Duration Calculus of Weakly Monotonic Time

Paritosh K. Pandya[1] and Dang Van Hung[2]

[1] Tata Institute of Fundamental Research
Mumbai 400 005, India
email: pandya@tcs.tifr.res.in

[2] UNU/IIST
Macau
email: dvh@iist.unu.edu

Abstract. We extend Duration Calculus to a logic which allows description of *Discrete Processes* where several steps of computation can occur at the same time point. The resulting logic is called Duration Calculus of Weakly Monotonic Time (WDC). It allows effects such as *true synchrony* and *digitisation* to be modelled. As an example of this, we formulate a novel semantics of Timed CSP assuming that the communication and computation take no time.

1 Introduction

Many real-time systems are designed and analysed under the assumption that computation and communication do not take time. This assumption has been called **true synchrony** hypothesis [5,2]. Thus, only the waiting for external synchronisation or explicit delay statements take time. Such an abstraction provides an essential simplification in understanding the behaviour of real-time systems. Logics for real-time system must be capable of handling such abstractions.

One recent logic for real-time systems is the Duration Calculus (DC) [12]. The original duration calculus was intended as a logic for formulating requirements of real-time systems. At this level of description, the details of internal working of the system are hidden and only externally observable behaviour of the system is specified.

When behaviours under the true synchrony hypothesis are considered, it is quite natural that the system passes through a number of states within zero (negligible) time. To accommodate such behaviours, in this paper, we allow a discrete sequence of state changes to occur at a single "macro" time point. The resulting notion of time has been called in the literature as *weakly monotonic* [1]. We define an extension of Duration Calculus to weakly monotonic time including the notion of point-intervals from the mean-value calculus within our logic [13].

Our extension, called *Duration Calculus of Weakly Monotonic Time* (WDC), has been largely motivated by our attempts to capture the *low level* semantics of a wide spectrum of distributed and timed programming notations in Duration Calculus. These include synchronous languages like ESTEREL, SL and State charts, hardware description languages like Verilog and VHDL, and models such as timed automata. In most of these languages, a causally connected sequence of events may occur at the same time point.

We believe that many new interesting features such as true synchrony and digitisation of signals with discrete clocks [6] can be handled within the generalised setting. As an example of this, we give a novel semantics to Timed CSP under the true synchrony assumption.

The rest of the paper is organised as follows. Section 2 defines the notion weakly monotonic time and gives the syntax and semantics of *WDC*. Section 3 gives the semantics of Timed CSP under the assumption of true synchrony. The report ends with a brief discussion.

2 Duration Calculus of Weakly Monotonic Time

We shall assume that the system execution consists of a sequences of *phases* where each phase may last for zero, finite or infinite time. In order to describe such an execution, we introduce the notion of a *stepped time frame*. Figure 1 depicts the nature of stepped time-frame.

The horizontal axis represents *macro* time whereas the vertical axis represents the *phase-count*. Macro time refers to the external view of time with respect to which the real-time properties like response and delay are measured. An execution of the system gives rise to a *path* within the two-dimensional plane, which always extends by steps in either up or the right direction. We shall identify such a path by a *stepped time frame* consisting of the set of points on the path. Each such point is called a *micro* time point and represented by a tuple (t, i) where t is the macro-time and i is the phase count.

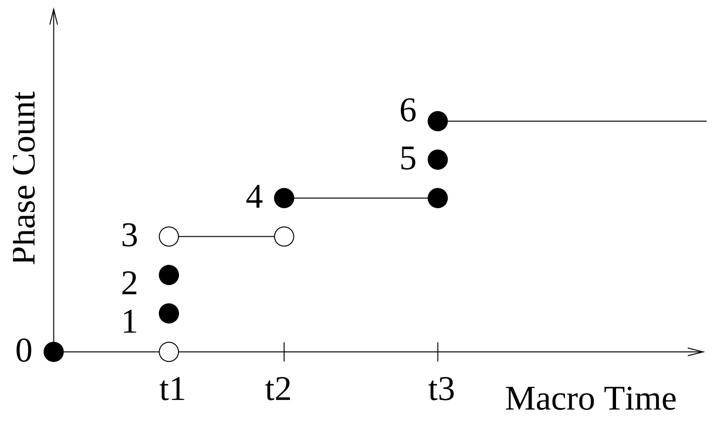

Fig. 1. Time Diagram

In the figure, solid circles indicates the presence of time-point whereas empty circles indicate the absence of time-point. We write the phase-count of each time point next to it. Thus, the diagram above represents the following stepped time frame:

$$\{(x,0) \mid 0 \leq x < t1\} \cup \{(t1,1),\ (t1,2)\} \cup \{(x,3) \mid t1 < x < t2\} \cup$$
$$\cup \{(x,4) \mid t2 \leq x \leq t3\} \cup \{(t3,5)\} \cup \{(x,6) \mid t3 \leq x\}$$

A behaviour of the system specifies the state of the system at each micro time point. Moreover, we shall assume the stability condition requiring that the state of the system remains unchanged throughout a phase. We shall now formalise the notion of a stepped time frame and behaviours over it.

2.1 Weakly-Monotonic Time

Macro-time We parameterise the logic with macro-time frame $TM = (T, <)$ where T is the set of time points and $<$ is "earlier than" order. It is assumed that TM is linearly ordered. The resulting logic is called $WDC(TM)$. In discrete time interpretation, TM is taken to be $(\omega, <)$, the set of natural numbers. In dense time interpretation, TM is taken to be $(\Re^0, <)$, the set of non-negative real numbers.

Every macro-time point is split into one or more micro-time point. Each micro-time point (t, i) consists of the macro-time t and a *phase-count* corresponding to the number of steps which have occurred *before* reaching this micro-time point.

Definition 1 (Stepped Time). Given a time frame TM, a stepped time frame w.r.t. TM is a pair $WTM = (WT, <)$ satisfying the following conditions:

- $WT \subseteq T \times \omega$. Let $\pi_1((t,i)) \overset{\text{def}}{=} t$ and $\pi_2((t,i)) = i$. Define $\pi_1(WT) = \{t \mid (t,i) \in WT\}$ and $\pi_2(WT) = \{i \mid (t,i) \in WT\}$.
- $<$ is the lexicographic ordering on WT, i.e. $(t_1, i_1) < (t_2, i_2)$ iff $t_1 < t_2 \vee (t_1 = t_2 \wedge i_1 < i_2)$. Thus, $(WT, <)$ is a total order.
- (**Phase Monotonicity**) $t_1 < t_2 \wedge (t_1, i_1) \in WT \wedge (t_2, i_2) \in WT \Rightarrow i_1 \leq i_2$.
- (**Progress**) $\pi_1(WT) = T \ \vee\ \pi_2(WT) = \omega$. This states that either the macro-time must increase indefinitely, or the micro-time must increase indefinitely. Here we allow macro-time to stop in case there are infinitely many phase changes which occur within finite time (finitely divergence [4]).
- (**Past closure**) $\pi_2(WT)$ is downward closed w.r.t. $<$ on ω, i.e. $i_1 \in \pi_2(WT) \wedge i_2 < i_1 \ \Rightarrow\ i_2 \in \pi_2(WT)$. Similarly, $\pi_1(WT)$ is downward closed w.r.t. $(T, <)$

We shall use b, e, m to range over WT. Let $Per(WT, i) \overset{\text{def}}{=} \{t \mid (t,i) \in WT\}$ denote the period of the ith phase.
Interpretation Let $Pvar$ be the set of propositional (state) variables. A behaviour over TM is a pair $I = (WTM, \theta)$, where WTM is a stepped time frame w.r.t. TM and the function θ assigns a boolean function of stepped time to each propositional variable $p \in Pvar$. Thus, $\theta(p) \in WT \rightarrow \{0, 1\}$. Moreover, θ

must satisfy the **stability** condition stating that for any $i \in \pi_2(WT)$, we have $\theta(p)(b) = \theta(p)(e)$ for all $b, e \in Per(WT, i)$.

We shall also assume that there are some global variables in the system, which do not change with time. Let $Gvar = Ivar \cup Rvar$ be the set global variables, which is partitioned into the disjoint subsets $Ivar$, of integer variables, and $Rvar$, of real variables. Finally, there are propositional interval variables, or formula variables, denoted by $X, Y \in Fvar$. Their value depends on the time interval.

The function θ also assigns a real-number to each global real variable $x \in Rvar$ and an integer to each integer variable. Thus,

$$\begin{aligned} \theta \in {} & (Pvar \to WT \to \{0,1\}) \times (Ivar \to \omega) \times \\ & \quad (Rvar \to \Re) \times (Fvar \to Intv \to \{0,1\}) \end{aligned}$$

For each proposition P, we can define a partial function $\theta_c(P) \in \pi_1(WT) \mapsto Bool$ of macro-time as follows.

$$\begin{aligned} \theta_c(P)(t) &= \theta(P)(t,i) && \{i \mid (t,i) \in WT\} \text{ is singleton} \\ &= \bot && \text{otherwise} \end{aligned}$$

Note that $\theta_c(P)$ is undefined at a discrete set of points. However, for $TM = (\Re^0, <)$, the function $\theta_c(P)$ is piecewise continuous (in fact, constant) and bounded. Hence we can define the Riemann integral $\int_{t_1}^{t_2} \theta_c(P)(t)dt$ as usual.

2.2 Logic *WDC*

Given a time frame TM, notation $WDC(TM)$ will denote Duration Calculus with weakly monotonic time over the frame TM. In the rest of this paper we shall assume that the time is dense, i.e. $TM = (\Re^0, <)$.

Syntax We have a two sorted logic with terms of type integer and real. Let P, Q range over states, $ti_1, ti_2, \ldots$ over terms of type integer, $tr_1, tr_2, \ldots$ over terms of type real, and D_1, D_2 over formulae. We shall use t to denote term of either type.

Real Terms have the form:

$$c \mid xr \mid \ell \mid \textstyle\int P \mid tr_1 \; op \; tr_2$$

where $op \in \{+, -, *\}$ and xr denotes a real-variable.

Integer terms have the form:

$$k \mid xi \mid \eta \mid tr_1 \; op \; tr_2$$

where $op \in \{+, -, *\}$ and xi denotes an integer variable.

Formulae have the form:

$$\lceil P \rceil^0 \mid Pt_l \mid t_1 = t_2 \mid t_1 < t_2 \mid \neg D \mid D_1 \wedge D_2 \mid \exists x.D \mid D_1 {}^\frown D_2$$

In the above, care must be taken to compare terms t_1, t_2 of the same type.
Semantics The set of intervals over stepped time frame $WTM = (WT, <)$ is defined as $Intv(WTM) = \{[b,e] \in WT \times WT \mid b \le e\}$

A *model* is a pair $(I, [b,e])$ where I is an interpretation (WTM, θ) and $[b,e]$ is an interval from $Intv(WTM)$. We shall denote the value of a term t in a model $(WTM, \theta, [b,e])$ by $\mathcal{V}(t)(WTM, \theta, [b,e])$, and the satisfaction of a formula D by $(WTM, \theta, [b,e]) \models D$. We shall typically abbreviate these by $\mathcal{V}(t)(\theta, [b,e])$ and $(\theta, [b,e]) \models D$.

The semantics is inductively defined as follows:

$$\begin{array}{lcl}
\mathcal{V}(\ell)(\theta,[b,e]) & = & \pi_1(e) - \pi_1(b) \\
\mathcal{V}(\eta)(\theta,[b,e]) & = & \pi_2(e) - \pi_2(b) \\
\mathcal{V}(\int P)(\theta,[b,e]) & = & \int_{\pi_1(b)}^{\pi_1(e)} \theta_c(P)\ dt \\
\mathcal{V}(x)(\theta,[b,e]) & = & \theta(x)
\end{array}$$

All the other cases are routine [12], and omitted.

$$\begin{array}{l}
(\theta,[b,e]) \models \lceil P \rceil^0 \quad \textbf{iff} \quad b = e \wedge \theta(P)(b) = 1 \\
(\theta,[b,e]) \models t_1 = t_2 \quad \textbf{iff} \quad \mathcal{V}(t_1)(\theta,[b,e]) \ = \ \mathcal{V}(t_2)(\theta,[b,e]) \\
(\theta,[b,e]) \models t_1 < t_2 \quad \textbf{iff} \quad \mathcal{V}(t_1)(\theta,[b,e]) \ < \ \mathcal{V}(t_2)(\theta,[b,e]) \\
(\theta,[b,e]) \models X \quad \textbf{iff} \quad \theta(X)[b,e] = 1 \\
(\theta,[b,e]) \models D_1 {}^\frown D_2 \quad \textbf{iff} \quad \exists m \in WT : b \leq m \leq e. \\
\qquad\qquad (\theta,[b,m]) \models D_1 \quad \wedge \quad (\theta,[m,e]) \models D_2
\end{array}$$

The usual set of derived operators are defined as usual. Notably, we have $Pt_i \stackrel{\text{def}}{=} \lceil 1 \rceil^0$ and $Ext_i \stackrel{\text{def}}{=} \neg Pt_i$. Also, $Pt_l \stackrel{\text{def}}{=} \ell = 0$ and $Ext_l \stackrel{\text{def}}{=} \neg Pt_l$. Let $Unit \stackrel{\text{def}}{=} Ext_i \wedge \neg(Ext_i {}^\frown Ext_i)$. Let $\lceil P \rceil^1 \stackrel{\text{def}}{=} \lceil P \rceil^0 {}^\frown Unit$. Also, $\lceil P \rceil \stackrel{\text{def}}{=} \neg(Ext_i {}^\frown \lceil \neg P \rceil^0 {}^\frown Ext_i)$. It is easy to see that $(\theta,[b,e]) \models \lceil P \rceil$ iff $b < e$ and $\theta(P)(m)$ for all $m : b < m < t$. Let $(WTM, \theta) \models D$ iff $(WTM, \theta, [b,e]) \models D$ for all $[b,e] \in Intv(WTM)$. Let $TM \models D$ iff $(WTM, \theta) \models D$ for all models (WTM, θ) over the macro time frame TM.

2.3 Extensions and Notation

We introduce some additional constructs and notational abbreviations which will be useful in the rest of the paper.

State Quantification It is convenient to include the state quantification construct $\exists p.D$ where p is a state variable in the language. Its semantics is given below (see [7] for a detailed discussion of the state quantification construct).

$(\theta,[b,e]) \models \exists p.D$ iff $(\theta',[b,e]) \models D$ for some p-variant θ' of θ.

The resulting logical may be called $QWDC$.

Notational Abbreviations

$$\begin{array}{ll}
\lceil P \rceil^1 \stackrel{\text{def}}{=} \lceil P \rceil^0 {}^\frown Unit & \lceil\lceil P \rceil \stackrel{\text{def}}{=} \lceil P \rceil^0 {}^\frown \lceil P \rceil \\
\lceil\lceil P \rceil \stackrel{\text{def}}{=} \lceil P \rceil^0 {}^\frown \lceil P \rceil & \lceil\lceil P \rceil\rceil \stackrel{\text{def}}{=} \lceil P \rceil^0 {}^\frown \lceil P \rceil {}^\frown \lceil P \rceil^0 \\
\lceil\lceil P \rceil^+ \stackrel{\text{def}}{=} \lceil\lceil P \rceil \vee \lceil P \rceil^0 & \lceil\lceil P \rceil^- \stackrel{\text{def}}{=} \lceil\lceil P \rceil \vee \lceil\ \rceil
\end{array}$$

Similarly $\lceil P \rceil\rceil$, $\lceil P \rceil\rceil^+$, $\lceil P \rceil\rceil^-$, $\lceil\lceil P \rceil\rceil^+$, $\lceil\lceil P \rceil\rceil^-$.

3 Compositional Semantics of Timed CSP

In this section, we give semantics to programming notations under the *true synchrony hypothesis*. In this hypothesis, it is assumed that *computation, communication/shared memory access* take no time. We interpret this to mean that they starts and finishes at the same *macro* time point. Any elapsing of macro time must be explicitly modelled using the delay t construct. Time can also elapse when the process waits to synchronise with its environment.

Using the true synchrony abstraction, we define a compositional semantics of Timed CSP, a well-known notation for distributed real-time programs, where processes synchronise by handshake communication. The semantics is low-level and captures the details of scheduling and interleaving of concurrent processes at the micro-step level.

In this section, we shall assume that the time-frames are *left-right continuous*, i.e. the following axiom holds

$$\Box(\eta > 0 \Rightarrow true^\frown Unit^\frown true)$$

Let S, T range over processes, a, c over channels and d over real constants. Let $cc, cc_1, \ldots$ range over communication commands of the form $c?$ or $c!$. A Timed CSP process has the following form.

$$\text{stop} \mid \text{skip} \mid \text{tick} \mid cc \mid \text{delay } d \mid S; T \mid S \sqcap T \mid [\Box_{i \in I}\, cc_i \rightarrow S_i] \mid$$
$$S \| T \mid S \setminus c \mid rec\, X.F(X) \mid S \overset{t}{\rhd} T \mid S \nabla_t T$$

Each process has the alphabet $\alpha = (In, Out)$ of input and output channels. Let $Ports(\alpha) = In \times \{?\} \cup Out \times \{!\}$.

Process stop represents a deadlocked process. skip terminates instantaneously without requiring any resources. On the other hand, tick takes one step of computation to terminate. Processes $c?$ and $c!$ represent input and output communication over channel c. Process delay d waits for d time units before terminating. The nondeterministic choice $S \sqcap T$, the guarded choice $[\ldots]$, the sequential composition $S; T$, the parallel composition $S \| T$, hiding $S \setminus c$ and recursion $rec\, X.F(X)$ function in the same manner as in CSP. The "tireout" construct $S \nabla_t T$ allows S to execute for t time units. After that the process is interrupted and it terminates. The timeout construct $S \overset{t}{\rhd} T$ is similar to the tireout, except that if process S performs any visible communication before time t then it is not interrupted. We refer the reader any standard work on Timed CSP, such as [3], for a detailed explanation of these constructs.

To record observable behaviour of a process, we use a state variable p for each $p \in Ports(\alpha)$ and a state variable c for each $c \in \alpha$. Variable p is true exactly when a process is waiting (ready) to communicate over the port p, and variable c is true exactly at those time points when a communication over c is taking place. We shall assume that communication takes no macro-time but requires one micro-step.

Notation For a set of states $C = \{c_1, \ldots, c_n\}$, let the state C denote $c_1 \vee c_2 \vee \ldots \vee c_n$. For a set of ports P and a set of channels C, let

$$\begin{aligned}
enonly_\alpha(P) &\stackrel{\text{def}}{=} \forall p \in P.\ p \quad \wedge \quad \forall p \in (Ports(\alpha) - P).\ \neg p \\
passive(C) &\stackrel{\text{def}}{=} \neg C \\
await_\alpha(P) &\stackrel{\text{def}}{=} enonly(P) \wedge passive(\alpha) \\
comm_\alpha(c) &\stackrel{\text{def}}{=} c \wedge passive(\alpha - c) \wedge enonly_\alpha(\emptyset) \\
Idle_\alpha &\stackrel{\text{def}}{=} enonly_\alpha(\emptyset) \wedge passive(\alpha)
\end{aligned}$$

For a process S over α we define a *WDC* formula $\mathcal{M}(S)$ capturing precisely the set of behaviours of S.Thus, $\mathcal{M}(S)$ gives the *strongest specification* of S. We use prefix-closed semantics. An interval temporal variable b records whether the execution is partial or terminated. Thus, $(\theta, [0, e]) \models \mathcal{M}(S)$ if and only if θ over the subinterval $[0, e]$ denotes a (partial) execution of S. Moreover, if $(\theta, [0, e]) \models \mathcal{M}(S) \wedge b$ then $(\theta, [0, e])$ denotes a complete execution where S has terminated. We shall use the following abbreviation:

$$\text{fstep}(D) \stackrel{\text{def}}{=} (Pt_i \wedge \neg b \quad \vee \quad Unit \frown D)$$

For example, the behaviour of process (delay $2; c?$) is given by the formula:

$$\begin{aligned}
&\lceil\lceil Idle \rceil\rceil^+ \wedge \ell < 2 \wedge \neg b \\
\vee\ &(\lceil\lceil Idle \rceil\rceil^+ \wedge \ell = 2 \frown \\
&\qquad \text{fstep}(\lceil\lceil await(c?) \rceil^- \frown \\
&\qquad\quad (\lceil await(c?) \rceil^0 \wedge \neg b \quad \vee \quad \lceil comm(c) \rceil^0 \frown \text{fstep}(Pt_i \wedge b)))\ )
\end{aligned}$$

The behaviour of $(c? \overset{t}{\rhd} d!)$ is given by:

$$\begin{aligned}
&\lceil\lceil await(c?) \rceil\rceil^+ \wedge \neg b \wedge \ell \leq t \\
\vee\ &(\lceil\lceil await(c?) \rceil^- \wedge \ell \leq t) \frown \lceil comm(c) \rceil^0 \frown \text{fstep}(Pt_i \wedge b) \\
\vee\ &(\lceil\lceil await(c?) \rceil\rceil^+ \wedge \ell = t) \frown \text{fstep}(\lceil\lceil await(d!) \rceil\rceil^+ \wedge \neg b \ \vee \\
&\qquad\qquad \lceil\lceil await(d!) \rceil^- \frown \lceil comm(d) \rceil^0 \frown \text{fstep}(Pt_i \wedge b))
\end{aligned}$$

Semantics

$$\begin{aligned}
\mathcal{M}(\text{stop}) &\stackrel{\text{def}}{=} \lceil\lceil Idle \rceil\rceil^+ \wedge \neg b \\
\mathcal{M}(\text{skip}) &\stackrel{\text{def}}{=} b \wedge Pt_i
\end{aligned}$$

tick takes one micro-step. It is an abstraction of statements like assignment which represent internal steps.

$$\mathcal{M}(\text{tick}) \stackrel{\text{def}}{=} (\lceil Idle \rceil^0) \frown \text{fstep}(Pt_i \wedge b)$$

Delay consumes d units of macro-time. Below we give a loose semantics to this construct where there is no restriction on the number of micro-steps it may take once time d has elapsed.

$$\begin{aligned}
\mathcal{M}(\text{delay } d) \stackrel{\text{def}}{=}\ &(\lceil\lceil Idle \rceil\rceil^+ \wedge \ell < t \wedge \neg b \\
&\vee (\lceil\lceil Idle \rceil\rceil^+ \wedge \ell = t) \frown \text{fstep}(Pt_i \wedge b)
\end{aligned}$$

An alternative semantics representing a "strong delay" construct can be given as

$$\begin{array}{l} \mathcal{M}(\text{delay } d) \overset{\text{def}}{=} (\lceil\lceil Idle \rceil\rceil^{+} \wedge \ell < t \wedge \neg b \\ \quad \vee \ \ (\lceil\lceil Idle \rceil\rceil^{+} \wedge \ell = t \wedge \neg(true^\frown(\ell = 0 \wedge Ext_i)))^\frown \text{fstep}(Pt_i \wedge b) \end{array}$$

The conjunct $\neg(true^\frown(\ell = 0 \wedge Ext_i)))$ ensures that there no micro-steps at the endpoint. This forces the delay d to terminate as soon as time d has passed without consuming any micro-steps.

$$\mathcal{M}(c?) \overset{\text{def}}{=} \begin{array}{l} \lceil\lceil await(c?) \rceil\rceil^{+} \wedge \neg b \\ \quad \vee \quad \lceil\lceil await(c?) \rceil^{-} {}^\frown \lceil comm(c) \rceil^{0} {}^\frown \text{fstep}(Pt_i \wedge b) \end{array}$$

The idea here is the process may be ready and wait for communication for arbitrary macro+micro time. The actual communication takes one micro-step.

$$\mathcal{M}(c!) \overset{\text{def}}{=} \begin{array}{l} \lceil\lceil await(c!) \rceil\rceil^{+} \wedge \neg b \\ \quad \vee \quad \lceil\lceil await(c!) \rceil^{-} {}^\frown \lceil comm(c) \rceil^{0} {}^\frown \text{fstep}(Pt_i \wedge b) \end{array}$$

$$\mathcal{M}(S; T) \overset{\text{def}}{=} \mathcal{M}(S) \wedge \neg b \quad \vee \quad \mathcal{M}(S)[true/b]^\frown \mathcal{M}(T)$$

$$\mathcal{M}(S \sqcap T) \overset{\text{def}}{=} \mathcal{M}(S) \vee \mathcal{M}(T)$$

In the following guarded choice construct, we assume that the guard cc_i is of the form $c_i?$ or $c_i!$, and $G = \{c_i \mid i \in I\}$.

$$\begin{array}{l} \mathcal{M}([\Box_{i \in I} \ cc_i \to S_i]) \overset{\text{def}}{=} \lceil\lceil await(G) \rceil\rceil^{+} \wedge \neg b \\ \quad \vee \ \bigvee_{i \in I} \quad \lceil\lceil await(G) \rceil^{-} {}^\frown \lceil comm(c_i) \rceil^{0} {}^\frown \text{fstep}(\mathcal{M}(S_i)) \end{array}$$

Parallel Composition Let $\mathcal{M}(PASS) \overset{\text{def}}{=} \lceil\lceil Idle \rceil^{-} \wedge b$. Then,

$$\begin{array}{l} \mathcal{M}(S \| T) \overset{\text{def}}{=} \exists b_1, b_2. \, (\mathcal{M}(S; PASS)[b_1/b] \wedge \mathcal{M}(T)[b_2/b] \\ \qquad\qquad\qquad\qquad \vee \ \ \mathcal{M}(S)[b_1/b] \wedge \mathcal{M}(T; PASS)[b_2/b] \ ) \\ \qquad\qquad\qquad \wedge \ \ (b \Leftrightarrow b_1 \wedge b_2) \ \ \wedge \ \ COND \end{array}$$

Here, *COND* axiomatises the underlying assumptions about the nature of concurrency. For example, we can choose one of the following.

- $COND_1 = true$ places no restrictions.
- $COND_2 = \lceil\lceil \neg(c? \wedge c!) \rceil\rceil^{+}$ states that the communication must occur as soon as it is possible (i.e. both sender and receiver become ready). Such semantics has been termed as maximum parallelism.
- $COND_3 = \Box(\lceil\lceil c? \wedge c! \rceil\rceil^{+} \Rightarrow \ell = 0)$ states that the communication can remain ready for many micro-steps but it must be completed within 0 macro-time. This allows us to represent causal connections between multiple communications which occur at the same time, while giving maximum parallelism at the level of macro-time.
- $COND_4 = COND_3 \wedge (\bigwedge_{c,d \in \alpha, c \neq d} \ \lceil\lceil c \Rightarrow \neg d \rceil\rceil^{+})$ gives maximum parallelism at the macro-time level. But it also requires that only one communication can occur in a micro-step. Thus it enforces interleaving semantics at the micro-step level.

Timing Related Constructs The "tireout" construct $S\nabla_t T$ allows S to execute for t time units. After that the process is interrupted and it terminates. We define the loose semantics of tire-out construct below where there is no restriction on the number of micro-steps the process S may execute after time t.

$$\begin{array}{rl}\mathcal{M}(S\nabla_t T) \overset{\text{def}}{=} & \mathcal{M}(S) \wedge \ell < t \;\;\vee\;\; \mathcal{M}(S) \wedge \ell = t \wedge b \\ & \vee\;\; ((\mathcal{M}(S)[false/b] \wedge \ell = t)^\frown \text{fstep}(\mathcal{M}(T))\end{array}$$

An alternative semantics defining the "strong tireout" construct can be given as follows, where process S may not perform any micro-steps once time t has elapsed. The control must immediately be passed to process T.

$$\begin{array}{l}\mathcal{M}(S\nabla_t T) \overset{\text{def}}{=} \;\; \mathcal{M}(S) \wedge \ell < t \\ \quad \vee\; ((\mathcal{M}(S)[false/b] \wedge \ell = t \wedge \neg(true^\frown(\ell = 0 \wedge Ext_i)))^\frown \text{fstep}(\mathcal{M}(T)))\end{array}$$

The timeout construct is similar to above, except that if process S performs any visible communication before time t then it is not interrupted. We give the loose semantics below. Strong semantics can also be defined.

$$\begin{array}{rl}\mathcal{M}(S \overset{t}{\rhd} T) \overset{\text{def}}{=} & \mathcal{M}(S) \wedge \ell \leq t \vee \mathcal{M}(S) \wedge (\ell \leq t^\frown \lceil \neg passive(\alpha) \rceil^0 {}^\frown true) \\ \vee & ((\mathcal{M}(S)[false/b] \wedge \ell = t \wedge \lceil\lceil passive(\alpha) \rceil\rceil^+)^\frown \text{fstep}(\mathcal{M}(T)))\end{array}$$

Hiding Hiding is easily modelled using the state quantification construct.

$$\mathcal{M}(S \setminus c) \overset{\text{def}}{=} \exists c?, c!, c.\;\; \mathcal{M}(S)$$

Recursion Recursion cannot be directly handled within the logic *WDC*. However, the logic can be extended with least and greatest fixed point operators as in Pandya [9]. This can be used to give semantics of recursion as follows.

We represent a process variable X by a *WDC* formula variable X, $\mathcal{M}(X) \overset{\text{def}}{=} X$. Then, recursion can be handled using the greatest fixed point operator.

$$\mathcal{M}(rec\ X.F(X)) \overset{\text{def}}{=} \nu X.\mathcal{M}(F(X))$$

It is easy to see that the following laws holds:

$$\begin{array}{l}\mathcal{M}(rec\ X.X) \;\Leftrightarrow\; true \\ \mathcal{M}(S; \text{skip}) \;=\; \mathcal{M}(S) \;=\; \mathcal{M}(\text{skip}; S)\end{array}$$

Similar law does not hold for tick.

4 Discussion

We have defined a notion of weakly monotonic time where several state changes can occur at the same (macro) time point. We have given a straightforward extension of Duration Calculus for weakly monotonic time. The extension makes it possible to model interesting new phenomena like the behaviour of synchronous programs and digitisation. We have illustrated this by giving a novel semantics of Timed CSP assuming the true synchrony hypothesis. A more substantial example of the use of *WDC* may be found in [8], where a compositional semantics of the synchronous programming language *SL* has been defined, in [11] where assumption-commitment style verification of shared variable programs is investigated, and in [10] where semantics of a subset of Verilog has been formulated. We have not dealt with digitisation [6] in this paper, which is a topic of our on-going work.

One significant feature of *WDC* is that it preserves almost all the properties of the original Duration Calculus. Thus, most of the proof rules for durations remain unchanged. Decidability properties of *WDC* are also similar to those of the original *DC*.

Acknowledgements This paper is a revised version of a technical report which has been available since August, 1996 as UNU/IIST technical report. We thank Anders Ravn, Hans Rischel, Xu Qiwen and Zhou Chaochen for their helpful comments which have helped in shaping this work to its present form.

References

1. R. Alur and T.A. Henzinger. Logics and models of real time: A survey. In *Proc. REX workshop on real-time: Theory in Practice*, volume 600 of *LNCS*. Springer-Verlag, 1991.
2. Gérard Berry and Georges Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19:87–152, 1992.
3. J. Davis and S. Schneider. A brief history of timed csp. *Theoretical Computer Science*, 138, 1995.
4. M.R. Hansen, P.K. Pandya, and Chaochen Zhou. Finite divergence. *Theoretical Computer Science*, 138:113–139, 1995.
5. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
6. T. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks. volume 623 of *LNCS*, pages 273–337. Springer-Verlag, 1992.
7. P.K. Pandya. Some extensions to mean-value calculus: expressiveness and decidability. In H. Kleine Buning, editor, *Proc. CSL'95*, volume 1092 of *LNCS*. Springer-Verlag, 1995.
8. P.K. Pandya. A compositional semantics of SL. Technical report, DeTfoRS Group, UNU/IIST, October 1997.
9. P.K. Pandya and Y.S. Ramakrishna. A recursive mean value calculus. Technical report, Computer Science Group, TIFR, Bombay, TCS-95/3, 1995.
10. G. Schneider and Qiwen Xu. Towards a formal semantics of Verilog using Duration Caculus. In *proc. of FTRTFT'98*, LNCS, 1998. (to appear).
11. Qiwen Xu and Swarup Mohalik. Compositional verification in duration calculus. Technical Report 136, UNU/IIST, P.O. Box 3058 Macau, 1997.
12. Chaochen Zhou, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.
13. Chaochen Zhou and Xiaoshan Li. A mean value calculus of durations. In *A Classical Mind: Essays in Honour of C.A.R. Hoare, A.W. Roscoe (ed)*, pages 431–451. Prentice Hall International, 1994.