ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

**He Jifeng, Oxford University Computing Laboratory, UK, John Cooke,
Loughborough University, UK, and Peter Wallis, University of Bath, UK (Eds)**

BCS-FACS 7th Refinement Workshop

Proceedings of the BCS-FACS 7th Refinement Workshop, Bath, 3-5
July 1996

Chopping a Point

Zhou Chaochen and Michael R. Hansen

Published in Collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

ISBN 3-540-76104-7

Chopping a Point

Zhou Chaochen^{*†}

Department of Information Technology, Technical University of Denmark
DK-2800 Lyngby, Denmark

Michael R. Hansen^{*}

Department of Information Technology, Technical University of Denmark
DK-2800 Lyngby, Denmark

Emails: {zcc, mrh}@it.dtu.dk

Abstract

This paper introduces a *super-dense chop* modality into the Duration Calculi. The super-dense chop can be used to specify a *super-dense computation*, where a number of operations happens simultaneously, but in a specific order. With this modality, the paper defines a real-time semantics for an OCCAM-like language. In the semantics, assignments and passing of messages in communications are assumed to be timeless operations.

1 Introduction

In a digital control system, a piece of program, hosted in a computer, acts as a controller in order to periodically receive sampled outputs of a plant, and to calculate and send controls to the plant. The program may be written in an OCCAM-like language as a loop of

$$sensor?x; \text{ } CAL(x, u); \text{ } actuator!u; \text{ } wait \text{ } T$$

where $CAL(x, u)$ stands for a program segment, which decides a current control from the current sampled data x and the previous control u . T is the sampling period. Typically, the time spent in the calculation and change of controls is negligible compared with the sampling period T . So control engineers make the comfortable assumption that the calculation program (i.e. $CAL(x, u)$) becomes a sequence of statements which are executed one by one, but consume no time. Similarly, the receiving and the sending (i.e. $sensor?x$ and $actuator!u$) are assumed to be a pair of consecutive operations which can happen instantaneously, if the partners (i.e. sensor and actuator) are willing to communicate. A computation of a sequence of operations which is assumed to be timeless is called a *super-dense computation*. A semantic model of super-dense computations is discussed e.g. in [10, Example 6.4.2] and [11], and is e.g. adopted by the languages Esterel [3] and Statecharts [9].

A super-dense computation is an abstraction of a real-time computation within a context with a grand time granularity. For instance, in the digital control system, the cycle time of the computer may be nanoseconds, while the sampling period of the plant may be seconds. In other words, the calculation ($CAL(x, u)$) and the communications ($sensor?x$ and $actuator!u$) may take micro- or milli-seconds, while the sampling period T (in $wait \text{ } T$) may take seconds. A computation time with fine time granularity is only negligible for computations not having infinite loops. Otherwise it is known as *Zeno* phenomenon, or *finite divergence* [5]. In this paper, finite divergence will not be taken into account.

^{*}This work is partially funded by the Danish research councils under the project Codesign.

[†]On leave of absence from International Institute for Software Technology, United Nations University, Macau, and from Software Institute, Academia Sinica, Beijing.

Another motivation to introduce super-dense computations arises from the program refinement area. One of the well-known algebraical laws for untimed programs is the *combine* law of assignments [16] (called the *merge assignment* law in [8]). The combine law can conclude that e.g. the two consecutive assignments

$$x := x + 1; x := x + 2$$

are equivalent to the single assignment

$$x := x + 3$$

In order to retain the combine law for real-time programs, one may simulate time consumption of assignments by wait statements, and assume that execution of assignment takes no time. Otherwise the execution time of two assignments may be twice the execution time of a single assignment, and the combine law can hardly be maintained. Under the assumption that assignments take no time, the following two consecutive assignments constitute a super-dense computation

$$x := x + 1; x := x + 2$$

This paper represents an attempt to treat super-dense computation in the framework of the Duration Calculi. The Duration Calculi [19] are extensions of Interval Temporal Logic [12]. They can be used to specify and reason about real-time systems. The real-time behavior of a system can be specified in terms of its *states*, where a state

$$S : \mathbf{R} \rightarrow \{0, 1\}$$

denotes a time dependent property of a system and $\mathbf{R}$ denotes the set of real numbers. States are assumed *finitely variable* (i.e. not finitely divergent). Thus, a state can alternate between being present ($S(t) = 1$) and absent ($S(t) = 0$) only finitely many times in any finite (bounded) interval.

Formulas of the Duration Calculus [20] express properties of the *duration* of states and can therefore not express point properties such as: “the value of variable x is 5 at time t ”, since this property of x has no duration. The Mean Value Calculus [21] is an extension of Duration Calculus where one can express point properties like the above one. But one cannot with the Mean Value Calculus express properties like: “the value of variable x is first 5 then 7 at time t ”, which are necessary in order to formalize super-dense computations. One approach to cope with this problem is to extend the model to allow several simultaneous events (e.g. assignments to x), thereby getting a model similar to that of [11]. In [15] traces as functions of time are used e.g. to model simultaneous events in a Duration Calculus framework.

Another approach is used in this paper, where we will introduce a new modality $\bullet$ (called *super-dense chop*) for expressing super-dense computations without changing the model. The idea, which is elaborated below, is to express state transitions as neighborhood properties and combine state transitions using $\bullet$ to specify super-dense computations.

We will define state transitions as certain *neighborhood* properties of a system. The neighborhood properties are designated as¹

$$\mathcal{N}_\leftarrow S \text{ and } \mathcal{N}_\rightarrow S$$

where $\mathcal{N}_\leftarrow S$ ($\mathcal{N}_\rightarrow S$) holds at time t if S is *stably* present to the left (right) side of t , i.e. present throughout some small *left (right)* neighborhood of t . A *state transition from S_1 to S_2* is expressed in terms of neighborhood properties as

$$\mathcal{N}_\leftarrow S_1 \wedge \mathcal{N}_\rightarrow S_2$$

That is, S_1 holds before the transition, and S_2 holds after the transition.

A variable x of a program can change its value during an execution of the program. One can interpret x as a function of time

$$x : \mathbf{R} \rightarrow Val$$

where Val stands for all possible values of x . The state

$$x = v, \quad (v \in Val)$$

¹The following notations are taken from [5].

Chopping a Point

is a time varying property ($\mathbf{R} \rightarrow \{0, 1\}$) of the program. It is reasonable to assume that the program is *timely progressive*, and thus a program variable can only change its value finitely many times in any finite period. Thus, the property $(x = v)$ is *finitely variable*. The assignment

$$x := x + 1$$

can be defined as the state transition

$$\exists v. \mathcal{R}(x = v) \wedge \mathcal{P}(x = v + 1)$$

This formula defines that the assignment first inherits a value (v) of x from its predecessor in the left neighborhood, and then passes the new value $(v + 1)$ to its successor in the right neighborhood. Similarly, the assignment

$$x := x + 2$$

can be defined as

$$\exists v. \mathcal{R}(x = v) \wedge \mathcal{P}(x = v + 2)$$

The stability assumption is against the concept of super-dense computation. A super-dense computation of the program

$$x := x + 1; x := x + 2$$

assumes that the value passing of x from $(x := x + 1)$ to $(x := x + 2)$ takes no time. Thus, the intermediate state $(x = v + 1)$ of the program is just unstable and invisible. In order to define super-dense computations in the Duration Calculi, one needs a new modality which can map a time instant in a grand time space into a non-point interval in a fine time space, so that an instant action (such as the value passing of x) in a grand time space can take some time in a fine one. In other words, an unstable intermediate state (such as $(x = v + 1)$) of a super-dense computation in the grand time space can become stable in the fine one.

We call the new modality for *super-dense chop* and designate it as $\bullet$. Consider two state transitions of the special forms: $\mathcal{R}S_1 \wedge \mathcal{P}S_2$ and $\mathcal{R}S_2 \wedge \mathcal{P}S_3$. The super-dense computation of these two state transitions can be modelled by

$$(\mathcal{R}S_1 \wedge \mathcal{P}S_2) \bullet (\mathcal{R}S_2 \wedge \mathcal{P}S_3)$$

Suppose that an *interpretation* $\mathcal{I}$ is given which associates a function $\mathcal{I}(S_i) : \mathbf{R} \rightarrow \{0, 1\}$ with each state $S_i, i = 1, 2, 3$. Then, this formula holds at a point t , iff there exists a *refined* interpretation of states (designated $\mathcal{I}'$), where the point t of $\mathcal{I}$ is expanded into an interval $[t, t + d], d > 0$, of $\mathcal{I}'$, such that $\mathcal{I}'$ satisfies $\mathcal{R}S_1 \wedge \mathcal{P}S_2$ at time t and $\mathcal{R}S_2 \wedge \mathcal{P}S_3$ at time $t + d$, and S_2 holds throughout the interval $(t, t + d)$, which links up the two transitions in $\mathcal{I}'$.

This situation is sketched in the following picture

Thus, a super-dense computation of the two consecutive assignments

$$x := x + 1; x := x + 2$$

can be modelled by

$$(\exists v. \kappa_{\mathbb{Q}}(x = v) \wedge \neg \rho^{\mathbb{A}}(x = v + 1)) \bullet (\exists v. \kappa_{\mathbb{Q}}(x = v) \wedge \neg \rho^{\mathbb{A}}(x = v + 2))$$

In this paper, a set of axioms and rules for the super-dense chop is developed, so that one can reason about super-dense computations, and prove, for example

$$\begin{aligned} & (\exists v. \kappa_{\mathbb{Q}}(x = v) \wedge \neg \rho^{\mathbb{A}}(x = v + 1)) \bullet (\exists v. \kappa_{\mathbb{Q}}(x = v) \wedge \neg \rho^{\mathbb{A}}(x = v + 2)) \\ \Leftrightarrow & (\exists v. \kappa_{\mathbb{Q}}(x = v) \wedge \neg \rho^{\mathbb{A}}(x = v + 3)) \end{aligned}$$

which formalizes the equivalence of the two consecutive assignments above and the single assignment $x := x + 3$.

The paper is organized as follows. A Duration Calculus with neighborhood formulas and super-dense chop is defined in Section 2. Axioms and theorems of this calculus are investigated in Section 3. In Section 4, a real-time semantics of an OCCAM-like language is presented, where assignments and message passings of communications are assumed to be timeless statements. Section 5 concludes the paper with some further discussions.

2 Duration Calculus with Chop and Neighborhoods

In this section we give a brief introduction to the syntax and semantics of Duration Calculus, where we focus on the new elements of neighborhood formulas and the super-dense chop.

2.1 Syntax

The three important syntactical categories are *states*, *terms* and *formulas*.

States: We assume that a set *State* of states is given. Examples of states from Section 1 are: $(x = v)$ and $(x = v + 1)$. Thus, these states have a structure which we will not elaborate on now, however. A state will denote a Boolean valued function of time.

Terms: The set *Term* of terms is constructed from the set of states and a given set of variables *Var* according to the following grammar, where $S \in \text{State}$, $v \in \text{Var}$, and $r \in \text{Term}$:

$$r ::= v \mid \int S \mid r + r \mid r - r \mid r \cdot r$$

A variable v is also called a *global* variable since it does not change its value over time. A term will denote a real valued interval function, and e.g. the term $\int S$ will denote the duration of S on the interval of consideration.

Formulas: The set *Formula* of formulas is obtained from the set of states and the set of terms according to the following grammar, where $S \in \text{State}$, $r \in \text{Term}$, and $\phi \in \text{Formula}$:

$$\phi ::= r = r \mid r > r \mid \kappa_{\mathbb{Q}} S \mid \nearrow S \mid \neg \phi \mid \phi \vee \phi \mid \exists v. \phi \mid \phi \bullet \phi$$

2.2 Semantics

States: An *interpretation* is a function

$$\mathcal{I} : \text{State} \rightarrow (\mathbf{R} \rightarrow \{0, 1\})$$

which associates a Boolean valued function of time $S_{\mathcal{I}} \hat{=} \mathcal{I}(S)$ with every state $S \in \text{State}$. Let *Intv* be the set of bounded and closed real intervals: $\{[a, b] \mid a, b \in \mathbf{R}, a \leq b\}$. We require that $S_{\mathcal{I}}$ is *finitely variable* on every interval $[a, b] \in \text{Intv}$, i.e. $S_{\mathcal{I}}$ is a piecewise continuous function, and hence integrable on every interval $[a, b]$.

When states have a structure, e.g. $S \vee S'$, then we assume that interpretations agree with this structure, e.g.

$$(S \vee S')_{\mathcal{I}}(t) = 1 \quad \text{iff} \quad S_{\mathcal{I}}(t) = 1 \text{ or } S'_{\mathcal{I}}(t) = 1$$

Terms: The meaning of a term r in an interpretation $\mathcal{I}$ is a real valued interval function: $\mathcal{I}(r) : Intv \rightarrow \mathbf{R}$. In the definition of $\mathcal{I}(r)$ we assume that the meaning of the function symbols: $+$, $-$, etc., are the associated functions of real arithmetic. Furthermore, variables occurring in terms are variables in the sense of first order logics. Their meaning is given by a *valuation* of type $Var \rightarrow \mathbf{R}$. We will not present details here, as this is standard from first order logic. The meaning of the term $\int S$ (called the *duration* of S) is given by:

$$\mathcal{I}(\int S) [a, b] \triangleq \int_a^b S_{\mathcal{I}}(t) dt$$

The duration of the constant state 1, i.e. $\int 1$, denotes the interval length $\mathcal{I}(\int 1) [a, b] = b - a$. This duration is used often, so it is abbreviated as $\ell \triangleq \int 1$.

Formulas: A formula ϕ denotes a truth value, given² an interpretation $\mathcal{I}$ and an interval $[a, b]$. By $\mathcal{I}, [a, b] \models \phi$ we denote that ϕ is true given $\mathcal{I}$ and $[a, b]$. The definition of $\models$ follows the structure of formulas. We only give the definitions for the neighborhood formulas and for the super-dense chop:

$$\begin{aligned} \mathcal{I}, [a, b] \models \lhd S & \quad \text{iff} \quad \int_{a-\delta}^a S_{\mathcal{I}}(t) dt = \delta \text{ for some } \delta > 0 \\ \mathcal{I}, [a, b] \models \rhd S & \quad \text{iff} \quad \int_b^{b+\delta} S_{\mathcal{I}}(t) dt = \delta \text{ for some } \delta > 0 \\ \mathcal{I}, [a, b] \models \phi_1 \bullet \phi_2 & \quad \text{iff} \quad \text{there exist } m \in [a, b], \delta > 0, \text{ and } \mathcal{I}': \\ & \quad \mathcal{I}', [a, m] \models \phi_1 \text{ and } \mathcal{I}', [m + \delta, b + \delta] \models \phi_2 \\ & \quad \text{and for every } S \in State: \\ & \quad S_{\mathcal{I}'}(t) = \begin{cases} S_{\mathcal{I}}(t) & \text{if } t \leq m \\ S_{\mathcal{I}}(t - \delta) & \text{if } t \geq m + \delta \\ S_{\mathcal{I}'}(m + \frac{\delta}{2}) & \text{if } m < t < (m + \delta) \end{cases} \end{aligned}$$

A formula ϕ is *valid* (written $\models \phi$), if $\mathcal{I}, [a, b] \models \phi$ holds for every interpretation $\mathcal{I}$ and every interval $[a, b]$.

Note, in the definition of $\bullet$, that the chop point m of $\mathcal{I}$ is mapped to a non-point interval $[m, m + \delta]$, and the interpretation $\mathcal{I}$ is changed to $\mathcal{I}'$, where $\mathcal{I}'$ gives a constant interpretation for each state within the inserted interval $[m, m + \delta]$, and otherwise $\mathcal{I}'$ agrees with $\mathcal{I}$.

Note that $\lhd S$ holds, given $\mathcal{I}$ and $[a, b]$, if there is a left neighborhood of a where S is 1 (almost) everywhere. It is allowed that $a < b$. The formula $\lhd_{\mathbb{Q}} S$ used in the introduction demands that $a = b$. This formula and $\lhd S$ can be defined as:

$$\lhd_{\mathbb{Q}} S \triangleq \ell = 0 \wedge \lhd S \quad \text{and} \quad \lhd S \triangleq \ell = 0 \wedge \lhd_{\mathbb{Q}} S$$

Since

$$\lhd S \Leftrightarrow \text{true} \bullet \lhd S \quad \text{and} \quad \lhd_{\mathbb{Q}} S \Leftrightarrow \lhd_{\mathbb{Q}} S \bullet \text{true}$$

we could have defined $\lhd$ and $\lhd_{\mathbb{Q}}$ in terms of $\lhd_{\mathbb{Q}}$ and $\lhd$ instead. We chose $\lhd$ and $\lhd_{\mathbb{Q}}$ as the basic neighborhood properties, since the axioms of neighborhood (see below) become slightly simpler in this case.

The following three abbreviations will be used later:

$$\begin{aligned} \lceil \rceil & \triangleq (\ell = 0) \\ \lceil S \rceil & \triangleq (\ell > 0) \wedge (\int S = \ell) \\ \lceil S \rceil^* & \triangleq \lceil \rceil \vee \lceil S \rceil \end{aligned}$$

The formula: $\lceil \rceil$ reads: “point interval”, and $\lceil S \rceil$ reads: “ S is 1 (almost) everywhere in a non-point interval”. Furthermore, we will use the standard abbreviations for the connectives of predicate logic: $\forall v. \phi, \phi_1 \wedge \phi_2, \phi_1 \Rightarrow \phi_2$, etc, and the relations $<, \leq$, etc. of arithmetic.

²We omit valuations for variables $v \in Var$, as it is standard for first order logic.

3 Axioms, Rules and Theorems

In this section, we present axioms and rules for neighborhood properties, super-dense chop and durations. We end this section with proofs of theorems concerning properties of super-dense computations.

3.1 Axioms and Rules for Neighborhoods

The following formulas can be taken as axioms and rules for neighborhood properties:

- N1.** $\lhd 1$ and $\lhd 1$
N2. $\lhd(S_1 \wedge S_2) \Leftrightarrow (\lhd S_1 \wedge \lhd S_2)$ and $\lhd(S_1 \wedge S_2) \Leftrightarrow (\lhd S_1 \wedge \lhd S_2)$
N3. $\lhd(S_1 \vee S_2) \Leftrightarrow (\lhd S_1 \vee \lhd S_2)$ and $\lhd(S_1 \vee S_2) \Leftrightarrow (\lhd S_1 \vee \lhd S_2)$
N4. $\lhd(\exists v. S(v)) \Leftrightarrow (\exists v. \lhd S(v))$ and $\lhd(\exists v. S(v)) \Leftrightarrow (\exists v. \lhd S(v))$
N5. $\lhd \neg S \Leftrightarrow \neg \lhd S$ and $\lhd \neg S \Leftrightarrow \neg \lhd S$
N6. If $S_1 \Rightarrow S_2$, then $\lhd S_1 \Rightarrow \lhd S_2$ and If $S_1 \Rightarrow S_2$, then $\lhd S_1 \Rightarrow \lhd S_2$

The law N4 is a generalization of N3. The validity of N3 can be derived from the assumption of the finite variability of any Boolean function, as one can prove that for any finitely varied functions S_1 and S_2 , $\lhd[S_1 \vee S_2]$ holds in an interval, iff the interval can be partitioned into finitely many subintervals such that $\lhd[S_1]$ or $\lhd[S_2]$ holds in each of the subintervals.

3.2 Axioms and Rule for Super-dense Chop

From the definition, it is obvious that $\bullet$ is *associative*, *distributive* (over $\vee$ and $\exists$) and *monotonic*.

Axiom 1 $\phi_1 \bullet (\phi_2 \bullet \phi_3) \Leftrightarrow (\phi_1 \bullet \phi_2) \bullet \phi_3$

Axiom 2 Suppose that v is not *free* in ϕ .

$$\begin{aligned} (\phi_1 \vee \phi_2) \bullet \phi_3 &\Leftrightarrow (\phi_1 \bullet \phi_3) \vee (\phi_2 \bullet \phi_3) \\ \phi_3 \bullet (\phi_1 \vee \phi_2) &\Leftrightarrow (\phi_3 \bullet \phi_1) \vee (\phi_3 \bullet \phi_2) \\ (\exists v. \phi_1) \bullet \phi &\Leftrightarrow \exists v. (\phi_1 \bullet \phi) \\ \phi \bullet \exists v. \phi_1 &\Leftrightarrow \exists v. (\phi \bullet \phi_1) \end{aligned}$$

Rule 1 If $(\phi_2 \Rightarrow \phi_3)$, then

$$(\phi_1 \bullet \phi_2) \Rightarrow (\phi_1 \bullet \phi_3) \text{ and } (\phi_2 \bullet \phi_1) \Rightarrow (\phi_3 \bullet \phi_1)$$

Axiom 3 If S is *consistent*³, $\lhd \phi_1$, and $\lhd \phi_2$, then

$$(\phi_1 \wedge \lhd S) \bullet (\lhd S \wedge \phi_2) \Leftrightarrow (\phi_1 \bullet \phi_2)$$

Axiom 4 $\lhd S \bullet \lhd \neg S \Leftrightarrow \text{false}$

Axiom 5 If $\lhd \phi_1$ and $\lhd \phi_2$, then

$$\phi_1 \bullet \lhd S \Leftrightarrow \phi_1 \wedge \lhd S \text{ and } \lhd S \bullet \phi_2 \Leftrightarrow \lhd S \wedge \phi_2$$

The intermediate state S can link up the two formulas $\phi_1 \wedge \lhd S$ and $\lhd S \wedge \phi_2$ in Axiom 3, since ϕ_1 and ϕ_2 place no demands in the intermediate states. No intermediate state exists which can link up $\lhd S$ and $\lhd(\neg S)$ in Axiom 4.

³That is, S is not equivalent to 0.

3.3 Axioms and Rules for Durations and Finite Variability

The axioms and rules for durations and finite variability are taken from the original duration calculus.

Axiom I $\int 0 = 0$

Axiom II $\int S \geq 0$

Axiom III $\int S_1 + \int S_2 = \int (S_1 \vee S_2) + \int (S_1 \wedge S_2)$

Axiom IV $(\int S = a_1 + a_2) \Leftrightarrow (\int S = a_1) \bullet (\int S = a_2) \quad a_1, a_2 \geq 0$

Observe that Axiom IV is valid since the formulas $\int S = a_1$ and $\int S = a_2$ place no demands on the intermediate states, which are introduced in the semantics of $(\int S = a_1) \bullet (\int S = a_2)$. This leads to a general remark (and rule) on the relationship with $\bullet$ and the chop operator $\frown$ of Interval Temporal Logic [12], which was used in the original Duration Calculus. It has the semantics:

$$\mathcal{I}, [a, b] \models \phi_1 \frown \phi_2 \text{ iff } \mathcal{I}, [a, m] \models \phi_1 \text{ and } \mathcal{I}, [m, b] \models \phi_2, \text{ for some } m \in [a, b]$$

Since Duration Calculus is based on Interval Temporal Logic [12], which has no neighborhood formulas, we have:

Rule 2: If $\neg, \nearrow \not\models \phi$ and $\phi[\frown/\bullet]$ is a theorem of Interval Temporal Logic, then ϕ is a theorem of Duration Calculus with super-dense chop. Here $\phi[\frown/\bullet]$ stands for the formula, which is obtained from ϕ by substituting each occurrence of $\bullet$ in ϕ with $\frown$.

The finite variability of states is formalized by two induction rules:

Induction Rule I

If $R(\lceil \cdot \rceil)$ holds, and $(R(X \vee X \bullet \lceil S \rceil) \wedge R(X \vee X \bullet \lceil \neg S \rceil))$ is provable from $R(X)$, then $R(\text{true})$ holds.

Induction Rule II

If $R(\lceil \cdot \rceil)$ holds, and $(R(X \vee \lceil S \rceil \bullet X) \wedge R(X \vee \lceil \neg S \rceil \bullet X))$, is provable from $R(X)$ then $R(\text{true})$ holds.

The previous four axioms and two induction rules constitute a (relatively) *complete* calculus [6] of the original Duration Calculus.

3.4 Theorems

The above axioms and rules will now be used to prove properties which relate to super-dense computations. First, we prove that $x := x + 1; x := x + 2$ is equivalent to $x := x + 3$, using the formula $\exists v. \neg \mathcal{Q}(x = v) \wedge \neg \mathcal{Q}(x = e(v))$ as semantics of the assignment $x := e$, where $e(v)$ denotes the expression obtained from e by replacing each occurrence of x in e with v .

Observe first that if $v' = v + 1$, then

$$\begin{aligned} & (\neg \mathcal{Q}(x = v) \wedge \neg \mathcal{Q}(x = v + 1)) \bullet (\neg \mathcal{Q}(x = v') \wedge \neg \mathcal{Q}(x = v' + 2)) \\ \Leftrightarrow & (\neg \mathcal{Q}(x = v) \wedge \neg \mathcal{Q}(x = v + 1)) \bullet (\neg \mathcal{Q}(x = v') \wedge \neg \mathcal{Q}(x = v' + 2)) && \text{def. } \neg \mathcal{Q}, \neg \\ \Leftrightarrow & \neg \mathcal{Q}(x = v) \bullet \neg \mathcal{Q}(x = v' + 2) && \text{Ax.3} \\ \Leftrightarrow & \neg \mathcal{Q}(x = v) \wedge \neg \mathcal{Q}(x = v' + 2) && \text{Ax.5} \\ \Leftrightarrow & \neg \mathcal{Q}(x = v) \wedge \neg \mathcal{Q}(x = v + 3) && \text{def. } \neg \end{aligned}$$

Furthermore, if $v' \neq v + 1$, then

$$(\neg \mathcal{Q}(x = v) \wedge \neg \mathcal{Q}(x = v + 1)) \bullet (\neg \mathcal{Q}(x = v') \wedge \neg \mathcal{Q}(x = v' + 2)) \Leftrightarrow \text{false}$$

because $x = v + 1 \Rightarrow \neg(x = v')$ and

$$\begin{aligned}
 & (\neg(x = v) \wedge \neg(x = v + 1)) \bullet (\neg(x = v') \wedge \neg(x = v' + 2)) \\
 \Rightarrow & (\neg(x = v) \wedge \neg\neg(x = v')) \bullet (\neg(x = v') \wedge \neg(x = v' + 2)) \\
 \Rightarrow & \neg\neg(x = v') \bullet \neg(x = v') \\
 \Rightarrow & \text{false}
 \end{aligned}$$

by the definitions of $\neg$, $\bullet$, N6, Rule 1 (twice) and Axiom 4.

The equivalence of $x := x + 1$; $x := x + 2$ and $x := x + 3$ can be derived using these two equivalences and Axiom 2:

$$\begin{aligned}
 & (\exists v. \neg(x = v) \wedge \neg(x = v + 1)) \bullet (\exists v. \neg(x = v) \wedge \neg(x = v + 2)) \\
 \Leftrightarrow & \exists v, v'. ((\neg(x = v) \wedge \neg(x = v + 1)) \bullet (\neg(x = v') \wedge \neg(x = v' + 2))) \\
 \Leftrightarrow & \exists v. (\neg(x = v) \wedge \neg(x = v + 3))
 \end{aligned}$$

Theorem 1. $\neg S_1 \bullet \neg S_2 \Leftrightarrow \neg S_1 \wedge \neg S_2$

Proof:

$$\begin{aligned}
 & \neg S_1 \bullet \neg S_2 \\
 \Leftrightarrow & (\neg S_1 \bullet \text{true}) \bullet (\text{true} \bullet \neg S_2) \quad \text{A5} \\
 \Leftrightarrow & ((\neg S_1 \bullet \text{true}) \bullet \text{true}) \bullet \neg S_2 \quad \text{A1} \\
 \Leftrightarrow & (\neg S_1 \bullet (\text{true} \bullet \text{true})) \wedge \neg S_2 \quad \text{A1, A5} \\
 \Leftrightarrow & (\neg S_1 \wedge (\text{true} \bullet \text{true})) \wedge \neg S_2 \quad \text{A5} \\
 \Leftrightarrow & \neg S_1 \wedge \neg S_2 \quad \text{R.2 since } \text{true} \wedge \text{true} \Leftrightarrow \text{true}
 \end{aligned}$$

Theorem 2. If $(S_2 \wedge S_3)$ is consistent, then

$$(\neg S_1 \wedge \neg S_2) \bullet (\neg S_3 \wedge \neg S_4) \Leftrightarrow (\neg S_1 \wedge \neg S_4)$$

Proof:

$$\begin{aligned}
 & (\neg S_1 \wedge \neg S_2) \bullet (\neg S_3 \wedge \neg S_4) \\
 \Leftrightarrow & (\neg S_1 \wedge \neg((S_2 \wedge S_3) \vee (S_2 \wedge \neg S_3))) \bullet (\neg S_3 \wedge \neg S_4) \quad \text{N6} \\
 \Leftrightarrow & (\neg S_1 \wedge \neg(S_2 \wedge S_3)) \bullet (\neg S_3 \wedge \neg S_4) \\
 & \vee (\neg S_1 \wedge \neg(S_2 \wedge \neg S_3)) \bullet (\neg S_3 \wedge \neg S_4) \quad \text{N3 \& Ax.2} \\
 \Leftrightarrow & (\neg S_1 \wedge \neg(S_2 \wedge S_3)) \bullet (\neg S_3 \wedge \neg S_4) \quad \text{N6, R.1 \& Ax.4} \\
 \Leftrightarrow & (\neg S_1 \wedge \neg(S_2 \wedge S_3)) \bullet (\neg((S_3 \wedge S_2) \vee (S_3 \wedge \neg S_2)) \wedge \neg S_4) \quad \text{N6} \\
 \Leftrightarrow & (\neg S_1 \wedge \neg(S_2 \wedge S_3)) \bullet (\neg(S_3 \wedge S_2) \wedge \neg S_4) \\
 & \vee (\neg S_1 \wedge \neg(S_2 \wedge S_3)) \bullet (\neg(S_3 \wedge \neg S_2) \wedge \neg S_4) \quad \text{N3 \& Ax.2} \\
 \Leftrightarrow & (\neg S_1 \wedge \neg(S_2 \wedge S_3)) \bullet (\neg(S_3 \wedge S_2) \wedge \neg S_4) \quad \text{N6, R.1 \& Ax.4} \\
 \Leftrightarrow & \neg S_1 \wedge \neg S_4 \quad \text{Ax.3, Th.1}
 \end{aligned}$$

Corollary. If S_2 is consistent, then

$$\neg S_1 \bullet \neg S_2 \Leftrightarrow \neg S_1 \quad \text{and} \quad \neg S_2 \bullet \neg S_1 \Leftrightarrow \neg S_1$$

Proof. We prove the first equivalence only:

$$\begin{aligned}
 & \neg S_1 \bullet \neg S_2 \\
 \Leftrightarrow & (\neg S_1 \wedge \neg 1) \bullet (\neg S_2 \wedge \neg 1) \quad \text{N1 \& Rule 1} \\
 \Leftrightarrow & \neg S_1 \wedge \neg 1 \quad \text{Th. 2} \\
 \Leftrightarrow & \neg S_1 \quad \text{N1}
 \end{aligned}$$

The formula $\exists v. \neg(x = v) \wedge \neg(x = v)$, abbreviated to $Cnt(x)$, expresses the continuity of x at a point and acts as a *unit* of $\bullet$ for states of x and their transitions:

Theorem 3.

$$\begin{aligned}
 Cnt(x) \bullet (\neg(x = v_1) \wedge \neg(x = v_2)) &\Leftrightarrow (\neg(x = v_1) \wedge \neg(x = v_2)) \\
 (\neg(x = v_1) \wedge \neg(x = v_2)) \bullet Cnt(x) &\Leftrightarrow (\neg(x = v_1) \wedge \neg(x = v_2)) \\
 Cnt(x) \bullet [x = v'] &\Leftrightarrow [x = v'] \\
 [x = v'] \bullet Cnt(x) &\Leftrightarrow [x = v']
 \end{aligned}$$

Proof Sketch:

Observe first that if $(v = v_1)$, then by the definitions of $\neg$ and $\neg$, Axioms 3 and 5, Rule 1, and Theorem 1:

$$\begin{aligned}
 &(\neg(x = v) \wedge \neg(x = v)) \bullet (\neg(x = v_1) \wedge \neg(x = v_2)) \\
 \Leftrightarrow &(\neg(x = v) \wedge \neg(x = v_2))
 \end{aligned}$$

Observe next that if $(v \neq v_1)$, then by the definitions of $\neg$ and $\neg$, Axiom 4, Rule 1, and N6:

$$(\neg(x = v) \wedge \neg(x = v)) \bullet (\neg(x = v_1) \wedge \neg(x = v_2)) \Leftrightarrow \text{false}$$

The first two parts of Theorem 3 follow from these observations. A proof of the last two parts can be given as

$$\begin{aligned}
 &Cnt(x) \bullet [x = v'] \\
 \Leftrightarrow &(\exists v. \neg(x = v) \wedge \ell = 0 \wedge \neg(x = v)) \bullet (\neg 1 \wedge [x = v']) && Cnt, \neg, \neg, N1, R.1. \\
 \Leftrightarrow &(\exists v. \neg(x = v)) \bullet [x = v'] && Th.2, \exists, \neg \\
 \Leftrightarrow &(\neg \exists v. (x = v)) \bullet [x = v'] && N4 \\
 \Leftrightarrow &\neg 1 \bullet [x = v'] && (\exists v. (x = v)) \Leftrightarrow 1 \\
 \Leftrightarrow &\ell = 0 \bullet [x = v'] && N1, \text{def. } \neg \\
 \Leftrightarrow &[x = v'] && R.2
 \end{aligned}$$

4 Real-Time Semantics

In this section, an OCCAM-like notation and a real-time semantics of the notation are presented, where assignment statements and message passing of communications are assumed to be *instant* actions.

4.1 Syntax

Let $t \in (0, \infty)$ stand for *time* delay, $x, y \in PVar$ for *program variables*, $c, d \in Chan$ for *channels*, $e \in Exp$ for *arithmetical expressions* of program variables, and $B \in Bexp$ for *Boolean expressions* of program variables. The syntax of the notation is given by the grammar:

$$\begin{aligned}
 \mathcal{S} ::= & x := e \mid c!e \mid c?x \mid \text{wait } t \mid \mathcal{S}; \mathcal{S} \mid B \rightarrow \mathcal{S} \mid (c?x \rightarrow \mathcal{S} \parallel d?y \rightarrow \mathcal{S}) \mid (c?x \rightarrow \mathcal{S} \parallel \text{wait } t \rightarrow \mathcal{S}) \mid \mu Y. \mathcal{S} \\
 \mathcal{P} ::= & \mathcal{S} \mid (\mathcal{P} \parallel \mathcal{P})
 \end{aligned}$$

where $\mathcal{S}$ stands for *sequential* processes, and $\mathcal{P}$ for *parallel* processes. The informal meaning of the statements is given as follows.

$x := e$ assigns the value of e to x .

$c!e$ sends the value of e on channel c .

$c?x$ receives a message from channel c , and assigns the received message to x .

$\text{wait } t$ delays for t ($t > 0$) time units.

$\mathcal{S}_1; \mathcal{S}_2$ is the sequential composition of $\mathcal{S}_1$ and $\mathcal{S}_2$.

$(B \rightarrow S)$ behaves like S , if the value of the program variables satisfies B . Otherwise, B is false, and the process terminates immediately.

$(c?x \rightarrow S_1 \parallel d?y \rightarrow S_2)$ is a choice. If a communication on c is completed no later than the one on d , the first branch S_1 will be chosen, and similarly if a communication on d is completed no later than one on c , then the second branch S_2 is chosen.

$(c?x \rightarrow S_1 \parallel \text{wait } t \rightarrow S_2)$ is also a choice. If a communication on c is completed within t time units, the first branch is chosen. Otherwise the second one is chosen.

$\mu Y.S$ is a conventional recursion, where Y is the name of the recursion process and Y may occur in S . As mentioned before, this paper will exclude finite divergent behavior of processes. Therefore we always assume that any occurrence of Y in S is guarded by a *wait* statement, so that a process will not be engaged in infinitely many assignments or communications in a finite period.

$(\mathcal{P}_1 \parallel \mathcal{P}_2)$ is a parallel combination of sequential processes. Shared variables are excluded in a parallel combination. The only interaction between sequential processes in a parallel combination are communications over channels. Each channel is uni-directed and owned by two sequential processes: one at each end.

4.2 Semantics

We assume that a process can be *observed* on the channels it controls and on its program variables. A semantic domain of a process can be composed by mathematical models of its *observables*:

- For each program variable x , an observable of x is a step function

$$x : \mathbf{R} \rightarrow Val$$

which records the value of x as a function of time.

- For each channel c , we define three observables $c?$, $c!$ and c

$$c?, c! : \mathbf{R} \rightarrow \{0, 1\} \quad \text{and} \quad c : \mathbf{R} \rightarrow \bigcup_{n=0}^{\infty} Val^n$$

where Val^n is the n -dimensional Cartesian product of Val . When $n = 0$, Val^n contains one element called the empty sequence which is written $\langle \rangle$.

The intention with these observables is: $c?$ is 1 if a process is willing to receive a message along the channel c . Otherwise $c?$ is 0. $c!$ is 1 if a process is willing to send a message along the channel c . Otherwise $c!$ is 0. c records the communication history of channel c , i.e. $c(t)$ is a sequence of messages passed over channel c up to t . c is a step function.

We will concentrate on how to define super-dense computations of a process using neighborhood properties and the super-dense chop modality. We will use a technique [8, 17], where each process $\mathcal{P}$ will be characterized by two formulas: $\llbracket \mathcal{P} \rrbracket_{ter}$ and $\llbracket \mathcal{P} \rrbracket$ defining the terminating and the complete behavior of $\mathcal{P}$, respectively. The formula $\llbracket \mathcal{P} \rrbracket$ is *prefix-closed*. By *prefix-closed*, we mean that for any interpretation $\mathcal{I}$ and interval $[a, b]$ if

$$\mathcal{I}, [a, b] \models \llbracket \mathcal{P} \rrbracket,$$

then for any c ($b > c \geq a$)

$$\mathcal{I}, [a, c] \models \llbracket \mathcal{P} \rrbracket.$$

The aim is to show the expressive power of neighborhood properties and super-dense chop, so we will *not* care about theoretical details e.g. to give a proof for the *continuity* of the recursions. Furthermore, for simplicity, we also assume that a process has only one variable, say x or y , and two channels, say c and d , it may communicate on. It is easy to generalize the semantics to more realistic cases by introducing process alphabets.

For each process below we state the kind of communications on c and d and the variable it controls.

$x := e$. The assignment terminates immediately. It inherits a value of x from its left neighborhood and passes the changed value to its right neighborhood. The communication histories of c and d do not change:

$$\begin{aligned} \llbracket x := e \rrbracket_{ter} &\hat{=} (\exists v. \kappa_{\mathbb{Q}}(x = v) \wedge \neg \mathcal{A}(x = e(v))) \wedge Cnt(c) \wedge Cnt(d) \\ \llbracket x := e \rrbracket &\hat{=} \llbracket x := e \rrbracket_{ter} \end{aligned}$$

$c?x$. We assume that this process controls input on d also. As soon as this process synchronizes with its partner, it will receive a message, update the communication history of channel c and assign the message to x instantly. However, it may wait forever, if its partner refuses to send. The communication history of d does not change during the execution of $c?x$:

$$\begin{aligned} \llbracket c?x \rrbracket_{ter} &\hat{=} \exists h_1, h_2. Sync_1(h_1, h_2) \bullet \exists v. Comm_1(h_1, h_2, v) \\ \llbracket c?x \rrbracket &\hat{=} \exists h_1, h_2. Sync_1(h_1, h_2) \vee \llbracket c?x \rrbracket_{ter} \end{aligned}$$

where

$$\begin{aligned} Sync_1(h_1, h_2) &\hat{=} \kappa_{\mathbb{Q}}((c = h_1) \wedge (d = h_2)) \wedge [c? \wedge \neg d? \wedge \neg c! \wedge (c = h_1) \wedge (d = h_2)]^* \\ Comm_1(h_1, h_2, v) &\hat{=} \neg \mathcal{A}((c = h_1 \cdot \langle v \rangle) \wedge (d = h_2) \wedge (x = v)) \\ \langle v_1, \dots, v_n \rangle \cdot \langle v_{n+1}, \dots, v_{n+m} \rangle &\hat{=} \langle v_1, \dots, v_n, v_{n+1}, \dots, v_{n+m} \rangle \end{aligned}$$

The formula $Sync_1(h_1, h_2)$ defines the behavior of the process while it is waiting to receive a value from channel c . The process first inherits the values (h_1 and h_2) of channel histories c and d , and keeps $c?$ to be 1 as long as its partner does not engage in the communication (i.e. $c! = 0$). During the waiting period (if any), the process will keep the channel histories of c and d constant, and no communication over d is possible as $d? = 0$.

The formula $Comm_1(h_1, h_2, v)$ describes the time instant when v is received over channel c and assigned to x . Note that the history for c is changed while that of d is kept constant.

$c!e$. Assume that this process controls output on d also and the variable y . It is described in a way similar to $c?x$:

$$\begin{aligned} \llbracket c!e \rrbracket_{ter} &\hat{=} \exists h_1, h_2, v. Sync_2(h_1, h_2, v) \bullet Comm_2(h_1, h_2, v) \\ \llbracket c!e \rrbracket &\hat{=} \exists h_1, h_2, v. Sync_2(h_1, h_2, v) \vee \llbracket c!e \rrbracket_{ter} \end{aligned}$$

where

$$\begin{aligned} Sync_2(h_1, h_2, v) &\hat{=} \kappa_{\mathbb{Q}}((c = h_1) \wedge (d = h_2) \wedge (y = v)) \wedge [c! \wedge \neg d! \wedge \neg c? \wedge (c = h_1) \wedge (d = h_2)]^* \\ Comm_2(h_1, h_2, v) &\hat{=} \neg \mathcal{A}((c = h_1 \cdot \langle e(v) \rangle) \wedge (d = h_2) \wedge (y = v)) \end{aligned}$$

$wait\ t$. We assume that this process controls the variable x and input from both c and d . The process always terminates, and until time $t > 0$ has elapsed there are no activities on the state variables it controls:

$$\begin{aligned} \llbracket wait\ t \rrbracket_{ter} &\hat{=} Idle_1 \wedge (\ell = t) \\ \llbracket wait\ t \rrbracket &\hat{=} \exists h_1, h_2. \left(\begin{array}{l} \kappa_{\mathbb{Q}}((c = h_1) \wedge (d = h_2)) \\ \wedge \quad [\neg c? \wedge \neg d? \wedge (c = h_1) \wedge (d = h_2)]^* \\ \wedge \quad (\ell < t) \end{array} \right) \vee \llbracket wait\ t \rrbracket_{ter} \end{aligned}$$

where

$$Idle_1 \hat{=} \exists h_1, h_2, v. \left(\begin{array}{l} \kappa_{\mathbb{Q}}((c = h_1) \wedge (d = h_2) \wedge (x = v)) \\ \wedge \quad [\neg c? \wedge \neg d? \wedge (c = h_1) \wedge (d = h_2)]^* \\ \wedge \quad \neg \mathcal{A}((c = h_1) \wedge (d = h_2) \wedge (x = v)) \end{array} \right)$$

$\mathcal{S}_1; \mathcal{S}_2$.

$$\begin{aligned} \llbracket \mathcal{S}_1; \mathcal{S}_2 \rrbracket_{ter} &\hat{=} \llbracket \mathcal{S}_1 \rrbracket_{ter} \bullet \llbracket \mathcal{S}_2 \rrbracket_{ter} \\ \llbracket \mathcal{S}_1; \mathcal{S}_2 \rrbracket &\hat{=} \llbracket \mathcal{S}_1 \rrbracket \vee (\llbracket \mathcal{S}_1 \rrbracket_{ter} \bullet \llbracket \mathcal{S}_2 \rrbracket) \end{aligned}$$

$(B \rightarrow S)$. Assume that the process controls the variable x and input from c and d .

$$\begin{aligned} \llbracket B \rightarrow S \rrbracket_{ter} &\hat{=} (\neg B \wedge \llbracket S \rrbracket_{ter}) \vee ((\neg B) \wedge Cnt(c) \wedge Cnt(d) \wedge Cnt(x)) \\ \llbracket B \rightarrow S \rrbracket &\hat{=} (\neg B \wedge \llbracket S \rrbracket) \vee ((\neg B) \wedge Cnt(c) \wedge Cnt(d) \wedge Cnt(x)) \end{aligned}$$

$(c?x \rightarrow \mathcal{S}_1 \parallel d?x \rightarrow \mathcal{S}_2)$. Assume that this process controls x and input from c and d .

$$\begin{aligned} \llbracket c?x \rightarrow \mathcal{S}_1 \parallel d?x \rightarrow \mathcal{S}_2 \rrbracket_{ter} &\hat{=} \\ &\quad \exists h_1, h_2. Sync_3(h_1, h_2) \bullet \exists v. Comm_1(h_1, h_2, v) \bullet \llbracket \mathcal{S}_1 \rrbracket_{ter} \\ &\quad \vee \exists h_1, h_2. Sync_3(h_1, h_2) \bullet \exists v. Comm_3(h_1, h_2, v) \bullet \llbracket \mathcal{S}_2 \rrbracket_{ter} \\ \llbracket c?x \rightarrow \mathcal{S}_1 \parallel d?x \rightarrow \mathcal{S}_2 \rrbracket &\hat{=} \\ &\quad \exists h_1, h_2. Sync_3(h_1, h_2) \\ &\quad \vee \exists h_1, h_2. Sync_3(h_1, h_2) \bullet \exists v. Comm_1(h_1, h_2, v) \\ &\quad \vee \exists h_1, h_2. Sync_3(h_1, h_2) \bullet \exists v. Comm_3(h_1, h_2, v) \\ &\quad \vee \exists h_1, h_2. Sync_3(h_1, h_2) \bullet \exists v. Comm_1(h_1, h_2, v) \bullet \llbracket \mathcal{S}_1 \rrbracket \\ &\quad \vee \exists h_1, h_2. Sync_3(h_1, h_2) \bullet \exists v. Comm_3(h_1, h_2, v) \bullet \llbracket \mathcal{S}_2 \rrbracket \end{aligned}$$

where

$$\begin{aligned} Sync_3(h_1, h_2) &\hat{=} \neg((c = h_1) \wedge (d = h_2)) \wedge [c? \wedge d? \wedge \neg c! \wedge \neg d! \wedge (c = h_1) \wedge (d = h_2)]^* \\ Comm_3(h_1, h_2, v) &\hat{=} \neg((c = h_1) \wedge (d = h_2 \cdot \langle v \rangle) \wedge (x = v)) \end{aligned}$$

$(c?x \rightarrow \mathcal{S}_1 \parallel wait\ t \rightarrow \mathcal{S}_2)$. Assume that this process controls x and input from $c?$ and $d?$.

$$\begin{aligned} \llbracket c?x \rightarrow \mathcal{S}_1 \parallel wait\ t \rightarrow \mathcal{S}_2 \rrbracket_{ter} &\hat{=} \\ &\quad \exists h_1, h_2. (Sync_1(h_1, h_2) \wedge (\ell < t)) \bullet \exists v. Comm_1(h_1, h_2, v) \bullet \llbracket \mathcal{S}_1 \rrbracket_{ter} \\ &\quad \vee (Idle_1 \wedge (\ell = t)) \bullet \llbracket \mathcal{S}_2 \rrbracket_{ter} \\ \llbracket c?x \rightarrow \mathcal{S}_1 \parallel wait\ t \rightarrow \mathcal{S}_2 \rrbracket &\hat{=} \\ &\quad \exists h_1, h_2. (Sync_1(h_1, h_2) \wedge (\ell < t)) \\ &\quad \vee \exists h_1, h_2. (Sync_1(h_1, h_2) \wedge (\ell < t)) \bullet \exists v. Comm_1(h_1, h_2, v) \\ &\quad \vee \exists h_1, h_2. (Sync_1(h_1, h_2) \wedge (\ell < t)) \bullet \exists v. Comm_1(h_1, h_2, v) \bullet \llbracket \mathcal{S}_1 \rrbracket \\ &\quad \vee (Idle_1 \wedge (\ell = t)) \bullet \llbracket \mathcal{S}_2 \rrbracket \end{aligned}$$

$\mu Y. \mathcal{S}(Y)$, where we write $\mathcal{S}(Y)$ to denote that Y may occur in $\mathcal{S}$. The terminating and complete behavior of $\mu Y. \mathcal{S}(Y)$ can be extracted from iterations of $\mathcal{S}$: Let $\mathcal{S}^i(Y) = \mathcal{S}(\mathcal{S}(\dots \mathcal{S}(Y)))$, with i occurrences of $\mathcal{S}$. We also introduce an additional syntactical entity, called *miracle*, with the definition:

$$\begin{aligned} \llbracket miracle \rrbracket_{ter} &\hat{=} \text{false} \\ \llbracket miracle \rrbracket &\hat{=} \text{false} \end{aligned}$$

The terminating and complete behavior of $\mu Y. \mathcal{S}(Y)$ can be defined, using *miracle* and iterations of $\mathcal{S}$, as:

$$\begin{aligned} \llbracket \mu Y. \mathcal{S}(Y) \rrbracket_{ter} &\hat{=} \exists n > 0. \llbracket \mathcal{S}^n(miracle) \rrbracket_{ter} \\ \llbracket \mu Y. \mathcal{S}(Y) \rrbracket &\hat{=} \exists n > 0. \llbracket \mathcal{S}^n(miracle) \rrbracket \end{aligned}$$

$(\mathcal{P}_1 \parallel \mathcal{P}_2)$. Assume that $\mathcal{P}_1$ controls x and input from $c?$ and $d?$, and that $\mathcal{P}_2$ controls y and can output messages on $c!$ and $d!$. $\mathcal{P}_1$ and $\mathcal{P}_2$ will synchronize the communication histories of the channels c and d first, by initializing them as $\langle \rangle$, and then run in parallel. Any terminating process will maintain its status (described by $Idle_1$ for $\mathcal{P}_1$) until the other one also terminates. The maintenance of status for $\mathcal{P}_2$ can be described by:

$$Idle_2 \triangleq \exists h_1, h_2, v. \left(\begin{array}{l} \neg((c = h_1) \wedge (d = h_2) \wedge (y = v)) \\ \wedge \quad [\neg c! \wedge \neg d! \wedge (c = h_1) \wedge (d = h_2)]^* \\ \wedge \quad \nearrow((c = h_1) \wedge (d = h_2) \wedge (y = v)) \end{array} \right)$$

The semantics of $(\mathcal{P}_1 \parallel \mathcal{P}_2)$ is:

$$\begin{aligned} \llbracket \mathcal{P}_1 \parallel \mathcal{P}_2 \rrbracket_{ter} &\triangleq \neg((c = \langle \rangle) \wedge (d = \langle \rangle)) \wedge \left(\begin{array}{l} \vee \quad (\llbracket \mathcal{P}_1 \rrbracket_{ter} \wedge (\llbracket \mathcal{P}_2 \rrbracket_{ter} \bullet Idle_2)) \\ \vee \quad ((\llbracket \mathcal{P}_1 \rrbracket_{ter} \bullet Idle_1) \wedge \llbracket \mathcal{P}_2 \rrbracket_{ter}) \end{array} \right) \\ \llbracket \mathcal{P}_1 \parallel \mathcal{P}_2 \rrbracket &\triangleq \neg((c = \langle \rangle) \wedge (d = \langle \rangle)) \wedge \left(\begin{array}{l} \vee \quad (\llbracket \mathcal{P}_1 \rrbracket \wedge (\llbracket \mathcal{P}_2 \rrbracket_{ter} \bullet Idle_2)) \\ \vee \quad ((\llbracket \mathcal{P}_1 \rrbracket_{ter} \bullet Idle_1) \wedge \llbracket \mathcal{P}_2 \rrbracket) \\ \vee \quad (\llbracket \mathcal{P}_1 \rrbracket \wedge \llbracket \mathcal{P}_2 \rrbracket) \end{array} \right) \end{aligned}$$

According to the semantics, the *partial correctness* of process $\mathcal{P}$ with Pre and $Post$ as its pre- and post-conditions can be formulated as

$$\neg Pre \wedge \llbracket \mathcal{P} \rrbracket_{ter} \Rightarrow \nearrow Post.$$

The (bounded) *termination* of $\mathcal{P}$ under pre-condition Pre holds, if there exists $r \geq 0$ such that

$$\neg Pre \wedge \llbracket \mathcal{P} \rrbracket \Rightarrow (\ell \leq r).$$

A process is *deadlock free*, if communication traces of the process can always be expanded. A (bounded) *liveness* is therefore to prove a upper bound of the time period where the communication traces of a process maintain constant. Let $\mathcal{P}$ be a process which controls channel c and d . $\mathcal{P}$ has $r > 0$ as its upper bound of liveness under pre-condition Pre , if

$$\neg Pre \wedge \llbracket \mathcal{P} \rrbracket \Rightarrow \Box((\exists h_1, h_2. [(c = h_1) \wedge (d = h_2)]) \Rightarrow (\ell \leq r)),$$

where

$$\begin{aligned} \Diamond \phi &\triangleq \text{true} \bullet \phi \bullet \text{true}, \\ \Box \phi &\triangleq \neg \Diamond \neg \phi. \end{aligned}$$

We will not present here examples to derive properties of a process directly from its semantics, since it is very tedious. Verification technique is therefore necessary.

5 Discussion

We have shown how super-dense computations can be expressed using neighborhood properties and a new modal operator called super-dense chop. The advantage of this approach is that super-dense computations can be modelled in the framework of Duration Calculus without extending the basic model of Duration Calculus (cf. [19, 7, 14]).

The super-dense chop ($\bullet$) modality, like the *projection* modality of [13], is distinguished from the other interval modalities by defining not only accessibility relations among intervals, but also projection relations among interpretations. The novelty of $\bullet$ demands further research into its axiomatization. We do not know whether the completeness results of [4] and [18] can be easily adopted in interval logics with super-dense chop.

An interesting generalization to our approach may be to introduce $\neg$ and $\nearrow$ as modalities. That is, $\neg$ and $\nearrow$ can be applied to arbitrary formulas. The definition of $\neg \phi$ (or $\nearrow \phi$) can be given as

$$\mathcal{I}, [a, b] \models \neg \phi \text{ (or } \nearrow \phi)$$

iff there exists $\delta > 0$ such that

$$\mathcal{I}, [a - \delta, a] \models \phi \text{ (or } \mathcal{I}, [b, b + \delta] \models \phi)$$

With these two modalities, first order quantification, and the interval length (ℓ), one can define the well-known thirteen unary modal operators [2] and three binary modal operators [18] of intervals.

Another generalization of this work concerns the technique we have used to define semantics to an OCCAM-like language with parallel processes communicating over channels. This technique can also be used to give a compositional semantics for *shared variable* processes, such as

$$\mathcal{P}_1 \parallel \mathcal{P}_2$$

where $\mathcal{P}_1$ includes assignment $x := e_1$ and $\mathcal{P}_2$ includes $x := e_2$, and thus the variable x is shared by $\mathcal{P}_1$ and $\mathcal{P}_2$. A kind of trace (designated $\hat{x}$) to record the history of assignments of x (similar to *labelled process transitions* in [1]) can be used to give semantics to $x := e_1$:

$$\exists h. \kappa_{\mathcal{Q}}(\hat{x} = h) \wedge \exists h_1 \in \bigcup_{n=0}^{\infty} Val_2^n. \not\exists(\hat{x} = h \cdot h_1 \cdot \langle (e_1(lt(h \cdot h_1) \downarrow Val))_1 \rangle)$$

where $Val_i \hat{=} \{(v)_i | v \in Val\}$ is the set of labelled values, $lt(h)$ is the last element of sequence h , $(v)_i \downarrow Val$ is v , i.e. the value projection of a labelled value, and

$$e_1(lt(h \cdot h_1) \downarrow Val)$$

is therefore the value of e_1 when the concurrent process has just finished all assignments of x recorded in $h \cdot h_1$.

Similarly, the semantics of $x := e_2$ can be given as

$$\exists h. \kappa_{\mathcal{Q}}(\hat{x} = h) \wedge \exists h_1 \in \bigcup_{n=0}^{\infty} Val_1^n. \not\exists(\hat{x} = h \cdot h_1 \cdot \langle (e_2(lt(h \cdot h_1) \downarrow Val))_2 \rangle)$$

Acknowledgments: The first author would like to thank Xu Qiwen for raising the question of how to define with DC a real time semantics of assignments, where the combine law is retained. Furthermore, we would like to thank Anders P. Ravn, Michael Schenke, and Simon Mørk for valuable discussions to explore possible approaches to specifying superdense computations. We thank Martin Fränzle and Markus Müller-Olm to point out an origin of introducing the notation $\llbracket \mathcal{P} \rrbracket_{ter}$ and $\llbracket \mathcal{P} \rrbracket$. We also thank Ole-Johan Dahl, He Jifeng, Yassine Lakhneche, Kees Middleburg, Hans Rischel, and Willem-Paul de Roever for their comments.

References

- [1] M. Abadi and L. Lamport: Composing Specifications. In *ACM Trans. on Program. Lang. Syst.*, Vol. 15, pp. 73-132, 1993.
- [2] James F. Allen: Towards a General Theory of Action and Time. In *Artificial Intelligence*, Vol. 23, pp. 123-154, Elsevier, 1984.
- [3] Gérard Berry and Georges Gonthier: The Esterel Synchronous Programming Language: Design, Semantics, Implementation. In *Science of Computer Programming*, Vol.19, pp. 87-152, Elsevier, 1992.
- [4] B. Dutertre: Complete Proof Systems for First Order Interval Logic. In *Tenth Annual IEEE Symp. on Logic in Computer Science*, pp. 36-43, IEEE Press, 1995.
- [5] M.R. Hansen, P.K. Pandya and Zhou Chaochen: Finite Divergence. In *Theoretical Computer Science*, Vol.138, pp 113-139, 1995.

- [6] M.R. Hansen and Zhou Chaochen: Semantics and Completeness of Duration Calculus. In *Real-Time: Theory in Practice, REX Workshop*, LNCS 600, J.W. de Bakker, C. Huizing, W.-P. de Roever and G. Rozenberg (Eds.), pp. 209-225, Springer-Verlag, 1992.
- [7] M.R. Hansen and Zhou Chaochen: *Lecture Notes on the Logical Foundations of Duration Calculus*, Department of Computer Science, Technical University of Denmark, ID-TR: 1995-166, 1995.
- [8] He Jifeng: *PROVABLY CORRECT SYSTEMS: Modelling of Communication Languages and Design of Optimized Compilers*. McGraw-Hill, 1995.
- [9] C. Huizing, R. Gerth and W.-P. de Roever: Modelling Statecharts behaviour in a fully abstract way. In *CAAP'88*, M. Dauchet and M. Nivat (Eds.), LNCS 299, Springer-Verlag, pp. 271-294, 1988.
- [10] R. Koymans: *Specifying Message Passing and Time-Critical Systems with Temporal Logic*. LNCS 651, Springer-Verlag, 1992.
- [11] O. Maler, Z. Manna and A. Pnueli: From Timed to Hybrid Systems. In *Real-Time: Theory in Practice*, J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg (Eds.), LNCS 600, Springer-Verlag, pp. 447-484, 1992.
- [12] B. Moszkowski: A Temporal Logic for Multilevel Reasoning about Hardware. In *IEEE Computer*, Vol. 18, No. 2, pp. 10-19, 1985.
- [13] B. Moszkowski: Compositional Reasoning about Projected and Infinite Time. In *Proc. of the First IEEE Intl. Conf. on Engineering of Complex Computer Systems*, IEEE Press, 1995.
- [14] A.P. Ravn: *Design of Embedded Real-Time Computing Systems*, Department of Computer Science, Technical University of Denmark, DK-2800 Lyngby, Denmark, Doctoral Dissertation, ID-TR: 1955-170, 1995.
- [15] A.P. Ravn, H. Rischel and K.M. Hansen: Specifying and Verifying Requirements of Real-Time Systems. In *IEEE Trans. Software Eng.*, Vol. 19, No. 1, pp. 41-55, January 1993.
- [16] A.W. Roscoe and C.A.R. Hoare: The laws of OCCAM programming. In *Theoretical Computer Science* vol. 60, 1988, pp. 177-229.
- [17] M. Schenke: *Requirements to Programs: A Development Methodology for Real Time Systems Part 2*, Technical Report, Oldenburg University, 1995.
- [18] Y. Venema: A Modal Logic for Chopping Intervals. In *Journal of Logic and Computation*, Vol. 1, pp. 453-476, Oxford University Press, 1991.
- [19] Zhou Chaochen: Duration Calculi: An Overview. In *Proceedings of Formal Methods in Programming and Their Applications*, LNCS 735, D. Bjørner, M. Broy and I.V. Pottosin (Eds.), pp. 256-266, Springer-Verlag, July 1993.
- [20] Zhou Chaochen, C.A.R. Hoare and A.P. Ravn: A Calculus of Durations. In *Information Processing Letters*, Vol. 40, No. 5, pp. 269-276, 1991.
- [21] Zhou Chaochen and Li Xiaoshan: A Mean Value Calculus of Durations. In *A Classical Mind (Essays in Honour of C.A.R. Hoare)*, A.W. Roscoe (Ed.), Prentice-Hall, pp. 431-451, 1994.