The United Nations
University

UNU/IIST

International Institute for
Software Technology

Towards a Formal Semantics of Verilog using Duration Calculus

Gerardo Schneider and Xu Qiwen

March 20, 1998

UNU/IIST and UNU/IIST Reports

UNU/IIST is a Research and Training Center of the United Nations University. It was founded in 1992, and is located in Macau. UNU/IIST is jointly funded by the Governor of Macau and the Governments of China and Portugal through contribution to the UNU Endowment Fund.

The mission of UNU/IIST is to assist developing countries in the application and development of software technology.

UNU/IIST contributes through its programmatic activities:

1. advanced development projects in which software techniques supported by tools are applied,
2. research projects in which new techniques for software development are investigated,
3. curriculum development projects in which courses of software technology for universities in developing countries are developed,
4. courses which typically teach advanced software development techniques,
5. events in which conferences and workshops are organised or supported by UNU/IIST, and
6. dissemination, in which UNU/IIST regularly distributes to developing countries information on in ternational progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively partic ipate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU/IIST is on formal methods for software development. UNU/IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU/IIST produces a report series. Reports are either Research $\boxed{\text{R}}$, Technical $\boxed{\text{T}}$, Compendia $\boxed{\text{C}}$ or Administrative $\boxed{\text{A}}$. They are records of UNU/IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU/IIST or visit UNU/IIST home page: http://www.iist.unu.edu, if you would like to know more about UNU/IIST and its report series.

Zhou Chaochen, Director — 01.8.1997 – 31.7.2001

The United Nations University

UNU/IIST

International Institute for
Software Technology

P.O. Box 3058
Macau

Towards a Formal Semantics of Verilog using Duration Calculus

Gerardo Schneider and Xu Qiwen

Abstract

Verilog is a Hardware Description Language used for the design and description of hardware in a behavioral and structural way. The meaning of a Verilog program is given by a simulation semantics, which is based on events and lack the necessary formality. Clearly it would be desirable to have a logical formal semantics to reason about properties of Verilog programs and to prove, among other things, the equivalence between programs (specification and implementation). Verilog has some interesting features like concurrency, synchronism, shared variables, non-blocking assignments (scheduled assignments), timing controls, infinite computations, zero-time computations, etc., that makes it an interesting language to study. WDC (Duration Calculus with Weakly Monotonic Time), introduced by Pandya and Dang, seems to provide an appropriate semantical framework for formalising languages with zero-time delays and computations. We introduce a fixed point operator and infinite intervals in WDC obtaining μWDCI. We use μWDCI to give a compositional abstract semantics to the V^- subset of Verilog and in particular the fixed point operator is used to give semantics to the loop statements modelling infinite computation taking zero time (infinite in the discrete time) as well as infinite computation in the macro-time. We prove some results (at the semantical level) about the fixed point operator.

Contents

1 Introduction

Verilog is a hardware description language used for the design and description of hardware in a behavioral and structural way. Hence, it is possible to write an "abstract" program (that represents the behaviour of the system) and a more "concrete" (implementation, structural) one. The meaning of a Verilog program is given by a simulation semantics, which is based on events (changes in the values on wires and registers). As remarked by Gordon [7], Verilog is the most used HDL language (in 1995 Verilog was getting 66 percent of business with 25,000 designers). Even though Verilog is widely used, usual presentation of the simulation semantics lack the necessary formality, hence, equivalence between an abstract program and its implementation is given observing the output of a simulation cycle. Clearly it would be desirable to have a logical formal semantics to reason about properties of Verilog programs and to prove, among other things, equivalence between programs (specification and implementation). Verilog has some interesting features like concurrency, synchronism, shared variables, non-blocking assignments (scheduled assignments), timing controls, infinite computations, zero-time computations, etc., that makes it an interesting language to study.

WDC, introduced by Pandya and Dang [16], is an extension of Duration Calculus (DC) [9, 1] with Weakly Monotonic Time that provides an appropriate semantical framework for formalising languages with zero-time delays and computations. We extend WDC with fixed point operators and infinite intervals, to obtain μWDCI, and prove some results about fixed points in this logic (at the semantical level).

The idea of extending modal logics with fixed point operators is not new, some of the papers that treat them are [20, 18, 2]. In particular, Duration Calculus and some of its extensions were also extended with fixed point operators: DC (Duration Calculus) [14], MVD (Mean-Value Calculus) [17] and SDC (Super-Dense Duration Calculus) [21].

We give a compositional abstract semantics to the V^- subset of Verilog using μWDCI. In particular, the semantics of loop statements are given using the fixed point operator in such a way that we can capture finite and infinite computation taking zero time (infinite in the discrete time) as well as infinite computation in the macro-time. WDC was also used as a semantical framework for an ESTEREL-like language called SL [15], for shared-variable concurrency [16, Section 4.1] and for Timed CSP [16, Section 4.2].

This report is organised as follows. In the next section the syntax and semantics of WDC are briefly introduced. In the third section WDC is extended with fixed point operators (the semantics is presented in a set-theoretical approach). An overview of Verilog is given in Section 4 and an operational and logical semantics to V^- is defined in Section 5, including some results about the semantical formulas. In the following Section the difference between Pace's approach [12] and ours is discussed whereas in Section 7 the role of the idle_i formula is discussed. In Section 8 some results about fixed point formulae are presented and finally, the last section present some discussion about the current and future work.

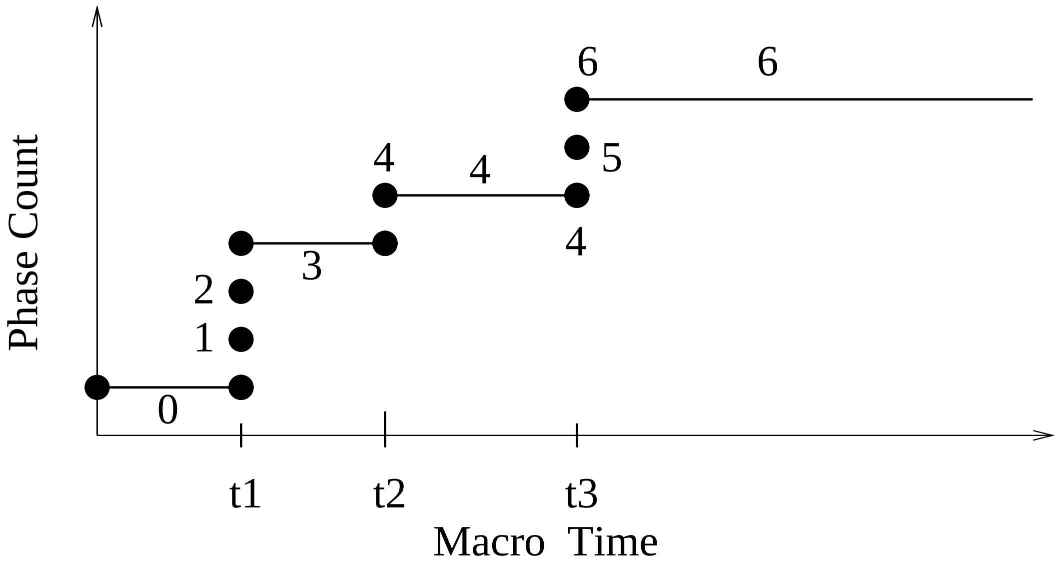

Figure 1: A time domain diagram.

2 Duration Calculus of Weakly Monotonic Time with Infinite Intervals (WDCI)

In this section we present basic concepts on $WDCI$. We do not discuss here the proof system and other deeper concepts. The logic presented here is practically the same as WDC but with the possibility of having infinite intervals, which is reflected syntactically just in the definition of the time domain and in the semantics of the "chop" ($\frown$) operator. For further reading on WDC see the original paper by Pandya and Dang [16]. The notation used here is slightly different from the one presented in [16].

2.1 Weakly-Monotonic Time

In WDC the first important features that distinguish it from DC is the *time domain* framework since we have a two dimensional time domain: 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. One graphic representation can be seen in Figure 1. Sometimes we can "flatted" a time domain in which case it looks like Figure 2-**a)** and when no ambiguity arise we do not even take in account the difference between the discrete and continuous time and will be represented as in Figure 2-**b)**. We have then the following definition,

Definition 2.1 *A* time domain *is a total order* $(\mathcal{T}, <)$*, where*

- $\mathcal{T} \subseteq (\Re^{+} \cup \{\infty\}) \times (\aleph \cup \{\infty\})$, *with* $(0,0) \in \mathcal{T}$
- $(r_1, n_1) < (r_2, n_2)$ *iff* $(r_1 \neq \infty \wedge r_1 \leq r_2 \wedge n_1 < n_2) \vee (r_1 < r_2 \wedge n_1 \neq \infty \wedge n_1 \leq n_2)$
- *for any* $t \notin \mathcal{T}$, *there exists* $t' \in \mathcal{T}$, *such that* $t \not\lesssim t'$ *and* $t' \not\lesssim t$.
- $lim((r_i, n_i)) \in \mathcal{T}$ □

With our definition of time domain we have two functions π_1 and π_2 that give the first and second projection of a time point respectively. That is, if $t = (r, n)$ then $\pi_1(t) = r$ and $\pi_2(t) = n$.

Let us analyse the definition of the time domain. The first and second conditions just define the domain and the order relationship ($<$). The third condition says that a time domain is maximal, in the sense that adding any other time point will cause the set to be no longer a total order. By the definition, exactly one of (r, ∞), (∞, n), and (∞, ∞) is in a time domain and it is the maximal element. The last condition stated that if the time domain has an infinite sequence of time points of the form $(r, n_0), (r, n_1), \ldots, (r, n_n), \ldots$ for some constant real number r, then (r, ∞) must be in the sequence. This is to avoid a sequence such as $(0, n_0), (0, n_1), \ldots, (0, n_n), \ldots, (1, \infty)$ for example, that without this condition would be allowed.

For any $t = (r, n)$, we write $t = \infty$ iff $n = \infty$ or $r = \infty$. An interval on $\mathcal{T}$ is a pair of time points $[b, e]$ of $\mathcal{T}$, where $b \neq \infty$ and $b \leq e$.

2.2 Syntax

It is important to emphasise here that our presentation of WDC is a bit different from the one presented in the original paper by Pandya and Dang [16].

The *symbols* are the following,

- Gvar: Integer global variables $x_i, y_i, z_i, \ldots$, real global variables $x_r, y_r, z_r, \ldots$ and boolean global variables $x_b, y_b, z_b, \ldots$ independent of time. In general we will use just $x, y, z, \ldots$ without distinguishing the type
- Tvar: Temporal variables, called *(State) durations*, with the form $\int S$ and $\hat{\int} S$, where S is a *State Expression.*
- Pletter: Temporal propositional letters $X, Y, \ldots$, that are truth-valued interval functions.
- Svar: State variables $P, Q, \ldots$.
- Sexp: State expressions $S, S_1, S_2, \ldots$.
- Fvar: Formulae variables $\alpha, \beta, \ldots, \phi, \psi, \ldots$

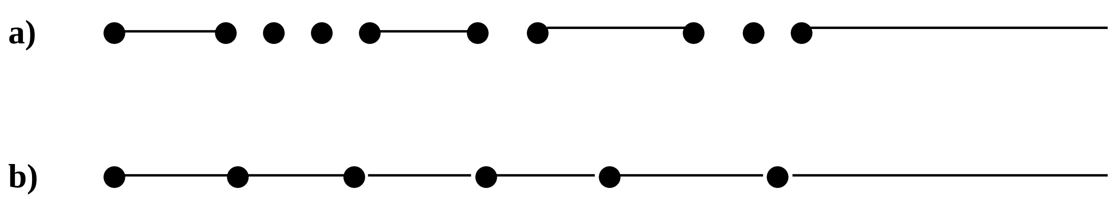

Figure 2: A "flatted" time domain diagram.

We do not introduce the global function and relation symbols because we will use just some real and integer operators $(+, -, \ldots)$as well as just the relation symbols $=$ and $<$ (considered in the State Expression as boolean operators).

We have a many sorted logic where the global variables (as a consequence the State Expressions and the Terms) can be of type boolean, integer and real, but for simplicity we will not make any difference between them and we will have a set $\mathsf{Values} = \Re \cup \aleph \cup \{0, 1\}$.

We will introduce first the syntax of the *State expressions* and of the *Terms* first.

State expressions:

$$S \ ::= \ c \ | \ x \ | \ P \ | \ S_1 \ op_1 \ S_2 \ | \ op_2 \ S$$

where $op_1 \in \{\vee, \wedge, +, -, *, =, <\}$ are the boolean, real and integer binary operators and $op_2 \in \{-, \neg\}$ are the unary operators (obviously $-$ for integer and reals and $\neg$ for boolean). Care must be taken using the operators between State expression of the same type.

Terms:

$$T \ ::= \ x \ | \ \int S \ | \ \hat{\int} S \ | \ T_1 \ op \ T_2 \ | \ \mathbf{b}.S \ | \ \mathbf{e}.S$$

where $op \in \{\vee, \wedge, +, -, *\}$ (for boolean, real and integer types). Care must be taken to apply the operators to terms of the same type. The durations of a state expression along real time and discrete time are denoted respectively as $\int S$ and $\hat{\int} S$, and consequently the lengths of an interval along real time and casual time are

$$\ell \stackrel{\text{def}}{=} \int 1$$
$$k \stackrel{\text{def}}{=} \hat{\int} 1$$

It is important to notice that we have separated the terms and the state expressions just to avoid non well-formed terms as $speed + \int speed$, where $speed$ is a state variable. On the other hand, we included the global variables x in both terms because it is possible to have $x_1 + speed$ and even $x_1 + \int speed$, but $x = speed$ it is a state expression whereas $x = \int speed$ is a formula (see the syntax of the formulas below).

Formulae:

$$\phi \ ::= \ X \ \mid \ T_1 = T_2 \ \mid \ T_1 < T_2 \ \mid \ \neg\phi \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \exists x \,.\, \phi \ \mid \phi_1 {}^\frown \phi_2$$

In the above, care must be taken to compare terms T_1, T_2 of the same type.

2.3 Semantics

A *behaviour* over $(\mathcal{T}, <)$ is a tuple $((\mathcal{T}, <), \mathcal{V}, \mathcal{I}, \mathcal{J})$ where

- $(\mathcal{T}, <)$ is the time domain as introduced in Definition 2.1
- $\mathcal{V}$: $\mathsf{GVar} \rightarrow \mathsf{Values}$ is a value assignment function for global variables, where $\mathsf{Values} = \Re \cup \aleph \cup \{0, 1\}$ (real numbers, natural numbers and $\{0, 1\}$)
- $\mathcal{I}$: $\mathsf{SVar} \rightarrow \mathcal{T} \rightarrow \mathsf{Values}$ is the interpretation of the state variables where $\mathsf{Values} = \Re^+ \cup \aleph \cup \{0, 1\}$
- $\mathcal{J}$: $\mathsf{Pletter} \rightarrow \mathbf{I}ntv \rightarrow \mathsf{Bool}$ is the interpretation of the Propositional variables over truth values.

Definition 2.2 *A* model $\mathbf{M}$ *is a tuple* $((\mathcal{T}, <), \mathcal{I}, \mathcal{J}, \mathcal{V}, [b, e])$*, where*

- $\mathcal{I} : \mathsf{SVar} \rightarrow \mathcal{T} \rightarrow \mathsf{Values}$
- $\mathcal{J} : \mathsf{Pletter} \rightarrow \mathbf{I}ntv \rightarrow \mathsf{Bool}$
- $\mathcal{V} : \mathsf{GVar} \rightarrow \mathsf{Values}$
- $[b, e]$ *is an interval on* $\mathcal{T}$ □

We shall often omit the time domain and simply write $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [b, e])$ for a model or even $[b, e]$.

Let $t, b, e \in \mathcal{T}$ be WDC time points (i.e. of the form (r, n) for given real number r and natural number n), then the semantics is inductively defined as follows.

The interpretation of state expressions is

- $[\![\, x \,]\!](t) \;=\; \mathcal{V}(x)$ (with $\mathcal{V}(x) \in \Re \cup \aleph \cup \{0,1\}$)
- $[\![\, P \,]\!](t) \;=\; \mathcal{I}(P)(t)$
- $[\![\, S_1 \; op_1 \; S_2 \,]\!](t) \;=\; [\![\, S_1 \,]\!](t) \; op_1^* \; [\![\, S_2 \,]\!](t)$
- $[\![\, op_2 \; S \,]\!](t) \;=\; op_2^* \; [\![\, S \,]\!](t)$

where op_1^* (op_2^*) is the interpretation of the operator op_1 (op_2) in the real, integer or $\{0,1\}$ domain.

The interpretation of terms is

- $[\![\, \int S \,]\!]((\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e])) \;=\; \int_{\pi_1(b)}^{\pi_1(e)} \mathcal{I}(S)(t) \; dt$
- $[\![\, \hat{\int} S \,]\!]((\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e])) \;=\; \sum_{\pi_2(b)+1}^{\pi_2(e)} \mathcal{I}(S)(t) \; dt$
- $[\![\, T_1 \; op \; T_2 \,]\!]((\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e])) \;=\; [\![\, T_1 \,]\!]((\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e])) \; op^* \; [\![\, T_2 \,]\!]((\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]))$
- $[\![\, \mathbf{b}.S \,]\!]((\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e])) \;=\; \mathcal{I}(S)(b)$
- $[\![\, \mathbf{e}.S \,]\!]((\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e])) \;=\; \mathcal{I}(S)(e)$

where op_1^* is the interpretation of the operator $(+, -, *)$ in the real or integer domain.

The semantics of the lengths of an interval along real time and casual time are

- $[\![\, \ell \,]\!]((\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e])) \;=\; \pi_1(e) - \pi_1(b)$
- $[\![\, k \,]\!]((\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e])) \;=\; \pi_2(e) - \pi_2(b)$

The semantics of the formulas is given by

- $[\![\, X \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \;=\; \mathcal{J}(X)$
- $[\![\, T_1 = T_2 \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \;=\; \mathsf{true}$ iff $[\![\, T_1 \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) = [\![\, T_2 \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e])$
- $[\![\, T_1 < T_2 \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \;=\; \mathsf{true}$ iff $[\![\, T_1 \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) < [\![\, T_2 \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e])$
- $[\![\, \neg\phi \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \;=\; \mathsf{true}$ iff $[\![\, \phi \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \;=\; \mathsf{false}$
- $[\![\, \phi_1 \wedge \phi_2 \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \;=\; \mathsf{true}$ iff $[\![\, \phi_1 \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) = \mathsf{true}$ and $[\![\, \phi_2 \,]\!](\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) = \mathsf{true}$

- $[\![\, \exists x\ .\ \phi \,]\!](\mathcal{I},\mathcal{J},\mathcal{V},[b,e]) \ = \ \mathsf{true}$ iff $[\![\, \phi \,]\!](\mathcal{I},\mathcal{J},\mathcal{V}',[b,e]) \ = \ \mathsf{true}$ for some $\mathcal{V}'$ which is x-equivalent to V, i.e $V(y) = V'(y)$ for every global variable $y (y \neq x)$
- $[\![\, \phi_1 \frown \phi_2 \,]\!](\mathcal{I},\mathcal{J},\mathcal{V},[b,e]) \ = \ \mathsf{true}$ iff there exists $c \in \mathcal{T}, b \leq c \leq e$, such that ($c \neq \infty$, $[\![\, \phi_1 \,]\!](\mathcal{I},\mathcal{J},\mathcal{V},[b,c]) = \mathsf{true}$ and $[\![\, \phi_2 \,]\!](\mathcal{I},\mathcal{J},\mathcal{V},[c,e]) = \mathsf{true}$) or ($e = \infty$ and $[\![\, \phi_1 \,]\!](\mathcal{I},\mathcal{J},\mathcal{V},[b,e]) = \mathsf{true}$).

The notions of *satisfaction* and *validity* are defined as usual

Definition 2.3 *A model* $(\mathcal{I},\mathcal{J},\mathcal{V},[b,e])$ satisfies *a formula* ϕ*, written as*

$$(\mathcal{I},\mathcal{J},\mathcal{V},[b,e]) \models \phi$$

iff

$$[\![\, \phi \,]\!](\mathcal{I},\mathcal{J},\mathcal{V},[b,e]) = \mathsf{true}$$

A formula ϕ *is* valid*, written as*

$$\models \phi$$

iff for all model $(\mathcal{I},\mathcal{J},\mathcal{V},[b,e])$

$$(\mathcal{I},\mathcal{J},\mathcal{V},[b,e]) \models \phi$$

□

In general we will use the satisfaction notation $(\mathcal{I},\mathcal{J},\mathcal{V},[b,e]) \models \phi$ instead of the semantical one $([\![\, \phi \,]\!](\mathcal{I},\mathcal{J},\mathcal{V},[b,e]))$.

The following formulas will be useful when defining the semantics of Verilog in Section 5.3:

$$\begin{aligned}
&\mathsf{fin} \stackrel{\mathrm{def}}{=} \ell < \infty \wedge k < \infty \\
&\mathsf{fin}^{\uparrow} \stackrel{\mathrm{def}}{=} k < \infty \\
&\mathrm{inf} \stackrel{\mathrm{def}}{=} \neg\mathsf{fin} \stackrel{\mathrm{def}}{=} \ell = \infty \vee k = \infty \\
&\mathsf{point} \stackrel{\mathrm{def}}{=} \ell = 0 \wedge k = 0
\end{aligned}$$

They characterise finite intervals, finite intervals in the discrete time only, infinite intervals and point intervals respectively. Let

$$\begin{aligned}
&\mathsf{dint} \stackrel{\mathrm{def}}{=} \ell = 0 \wedge k = 1 \\
&\mathsf{cint} \stackrel{\mathrm{def}}{=} \ell > 0 \wedge k = 0 \\
&\mathsf{unit} \stackrel{\mathrm{def}}{=} \mathsf{dint} \vee \mathsf{cint}
\end{aligned}$$

Intervals that satisfy cint and dint are called continuous and discrete respectively. If S is a boolean state expression, then we can define the following formulae:

$$\lceil S \rceil^0 \overset{\text{def}}{=} \mathsf{point} \wedge \mathbf{b}.S$$
$$\lceil S \rceil \overset{\text{def}}{=} \neg(\neg\mathsf{point} \frown \lceil \neg S \rceil^0 \frown \neg\mathsf{point})$$
$$\lceil\lceil S \rceil\rceil \overset{\text{def}}{=} \lceil S \rceil^0 \frown \lceil S \rceil \frown \lceil S \rceil^0$$
$$\lceil\lceil S \rceil\rceil^+ \overset{\text{def}}{=} \lceil\lceil S \rceil\rceil \vee \lceil S \rceil^0$$

The formula $\lceil S \rceil^0$ means that S is valid in a point interval. $\lceil S \rceil$ is read "everywhere inside (excluding the extremes of the interval), S" where $\lceil\lceil S \rceil\rceil$ means that "everywhere inside (including the extremes of the interval), S".

3 WDC with Fixed Point Operators and Infinite Intervals (μWDCI)

We extend the WDC introduced in Section 2 with infinite intervals and with *least fixed point* and *greatest fixed point* operators. The interpretation of a formula ϕ is given here using the set theoretical semantics approach $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi)$.

3.1 Syntax

The syntax of the state expressions, terms and formulas is the same as in Section 2.2 with the additional formula $\mu X.\phi_1$ corresponding to the *least fixed point* operator. Hence, the syntax of μWDCI formulas is

$$X \mid T_1 = T_2 \mid T_1 < T_2 \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x \,.\, \phi \mid \phi_1 \frown \phi_2 \mid \mu X.\phi$$

3.2 Semantics

We give the semantics of μWDCI using the set-theoretical approach, using the following semantical function:

$$\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}} : \mathsf{Fvar} \to \mathbf{I}ntv \to \mathsf{Bool}$$

where Fvar is the set of all μWDCI formulas, $\mathbf{I}ntv$ is the set of intervals and Bool is the Boolean set $\{\mathsf{true}, \mathsf{false}\}$. Sometimes we will use the notation $2^{\mathbf{I}ntv}$ instead of $\mathbf{I}ntv \to \mathsf{Bool}$. Then the above function give us the set of intervals that satisfy the given formula. The interpretation of the formulas is given by structural induction in the following way:

- $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(X) \stackrel{\text{def}}{=} \mathcal{J}(X)$
- $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(T_1 = T_2) \stackrel{\text{def}}{=} \{[b,e] \mid \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(T_1)(\mathcal{I},\mathcal{J},\mathcal{V},[b,e]) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(T_2)(\mathcal{I},\mathcal{J},\mathcal{V},[b,e])\}$
- $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(T_1 < T_2) \stackrel{\text{def}}{=} \{[b,e] \mid \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(T_1)(\mathcal{I},\mathcal{J},\mathcal{V},[b,e]) < \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(T_2)(\mathcal{I},\mathcal{J},\mathcal{V},[b,e])\}$
- $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\neg\phi) \stackrel{\text{def}}{=} 2^{\mathbf{I}ntv} - \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi) \stackrel{\text{def}}{=} \{[b,e] \mid [b,e] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi)\}$
- $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1 \wedge \phi_2) \stackrel{\text{def}}{=} \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \cap \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2)$
- $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\exists x \cdot \phi) \stackrel{\text{def}}{=} \bigcup_{a \in \mathsf{Values}} \mathcal{E}_{\mathcal{V}(x \mapsto a)}^{\mathcal{I},\mathcal{J}}(\phi)$
- $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1 \frown \phi_2) \stackrel{\text{def}}{=} \{[b,e] \mid \exists m \,.\, b \leq m \leq e \,.\, (m \neq \infty \wedge [b,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \wedge [m,e] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2)) \vee (e = \infty \wedge [b,e] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1))\}$
- $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X \cdot \phi) \stackrel{\text{def}}{=} \bigcap\{A \mid \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X \mapsto A)}(\phi) \subseteq A\}$

where $\mathcal{V}(x \mapsto a)$ means that the value (interpretation) of the global variable x is a, for $a \in \Re \cup \aleph \cup \{0,1\}$.

Note: We have introduced just the least fixed point operator since the greatest fixed point is easily defined as (see [2, p.25]):

$$\nu X \cdot \phi \stackrel{\text{def}}{=} \neg\mu Y \cdot \neg\phi[\neg Y/X]$$

where $\phi[\neg Y/X]$ means that we substitute $\neg Y$ for all the occurrences of the temporal propositional letter X in the formula ϕ. In this case the semantics of the greatest fixed point operator will be (see Proposition 8.1, page 57):

$$\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X \cdot \phi) \stackrel{\text{def}}{=} \bigcup\{A \mid A \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X \mapsto A)}(\phi)\}$$

For definitions and properties of fixed point operators in other logics see [13, p.9], [2, Section 12], [21], [18, Section 3].

The set theoretical semantics is related to satisfaction and validity in the following way: for any formula ϕ,

- $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [b, e]) \models \phi$ iff $[b, e] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi)$
- $\models \phi$ iff $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi) = 2^{\mathbf{I}ntv}$ for any model $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [b, e])$

For further results on the fixed point operators refer to Section 8 in page 57.

4 Overview of Verilog

Verilog is a Hardware Description Language (HDL) widely used to model the structure and behaviour of digital systems ranging from simple hardware building blocks to complete systems. In this section we will introduce Verilog briefly, discussing some semantical problems through examples. This section is based mainly on [7] (see also [12], [8], [10], [11] and [6] as well as http://www.cl.cam.ac.uk:80/users/mjcg/Verilog/ for some references about Verilog).

A specification in Verilog consists of a *main module*, that specifies a closed system containing both test data and hardware models, consisting of zero or more modules (or blocks). The specification can be *behavioural* or *structural* (or a combination of both). A behavioural specification defines the behaviour of a module using programming language constructs while the structural one expresses a module as a hierarchical interconnection of submodules. The modules have input and output *ports*, and events on the input ports cause events on the output ports. Verilog is the most used HDL that shows its importance to the industry. Its relevance is not only practical but also theoretical as it has some interesting features like for example:

- Normal programming language constructs (procedural assignments, loops, etc)
- Shared variables
- Synchronisation (Timing control: @(v), etc)
- Concurrency (modules, initial-statements, etc)
- Real-time constructs (Delay, Timing controls)
- Zero-duration computations
- Zero delays
- Non-blocking assignment
- Continuous assignments

An operational and logical semantics of a subset of Verilog (V^-) are given in Section 5 explaining most of these items. We will discuss briefly here just the non-blocking assignment since it is not included in the V^- subset. The non-blocking assignment, in contrast with the procedural (or blocking) assignment (or blocking assignment), does not "block" the execution of the process when the statement is reached, i.e. the control of the program continues with the next enabled statement and the non-blocking assignment is done after all the active events have been executed (see Example 4.1.4 and Section 4.2 for an explanation of the usual semantics of Verilog).

4.1 Some simple examples

We present here a few simple examples to show some features of Verilog. The first three examples can be found in [7]. All the examples were tested using a freeware version of the Veriwell/SPARC Simulator (Copyright 1994 Wellspring Solutions, Inc.).

4.1.1 Example

The following is a very simple example showing a behavioural specification of a `NAND` gate.

```
module NAND (i1,i2,o);
  input i1,i2; output o;

  assign o = ~(i1 & i2);
endmodule
```

The ports `i1,i2` and `o` are *wires*, as opposed to *procedural variables*, have no storage capacity: they can be continuously driven or left unconnected). The symbols `~` and `&` denote negation and conjunction respectively. The *continuous assignment* `assign o = ~(i1 & i2)` is activated whenever a change on `i1` or `i2` occurs and in this case the expression `~(i1 & i2)` is evaluated and the result is immediately propagated to the left hand side variable.

4.1.2 Example

The following example shows an edge-triggered Dtype register (see Figure 3).

Whenever there is a positive edge on the input `ck` (i.e. a change from 0 to 1) the value of the input `d` is copied to the output `q`. A behavioural specification is

```
module DTYPE (ck,d,q);
  input ck,d; output q; reg q;
```

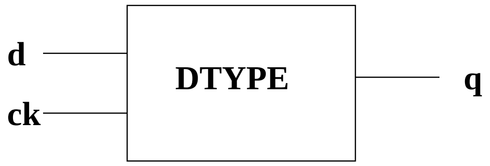

Figure 3: An edge-triggered Dtype register.

```
  always @(posedge ck) q = d;

endmodule
```

Here `ck` and `d` are wires and `q` is a *register*. Registers are variables that stores values (just like variables in imperative programming languages). The statement `always @(posedge ck) q =d` makes the timing control `@(posedge ck) q =d` be repeated forever. This timing control waits for a positive edge on `ck` and in this case the procedural assignment `q = d` is executed. The circuit shown in Figure 4 is an implementation of a Dtype register and it can easily be specified in Verilog:

```
module DTYPE_IMP (ck,d,q);
  input ck,d; output q; wire w1,w2,w3,w4,w5;

  NAND  M1(w2,d,w1);
  NAND3 M2(w3,ck,w1,w2);
  NAND  M3(w4,ck,w3);
  NAND  M4(w1,w3,w4);
  NAND  M5(w3,w5,q);
  NAND  M6(q,w2,w5);

endmodule
```

where `NAND3` is the following three-input NAND-gate:

```
module NAND3 (i1,i2,i3,o);
  input i1,i2,i3; output o;

  assign o = ~(i1 & i2 & i3);

endmodule
```

The module `DTYPE_IMP` shows two important characteristics of Verilog:

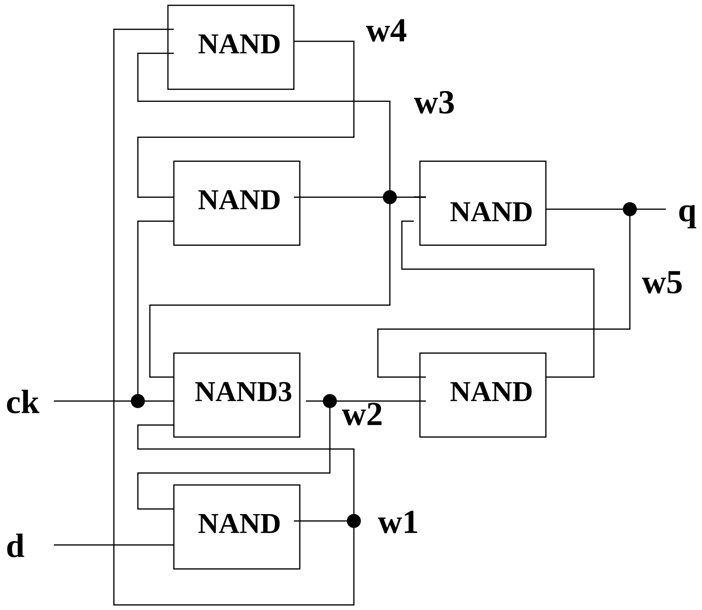

Figure 4: An implementation of a Dtype.

- Concurrency. All the `NAND` and the `NAND3` modules are executed in parallel in a separate thread of computation, where shared variables can lead to non-determinism (see Example 4.1.3)
- Instantiation of modules: the module `NAND` is instantiated five times here (distinguished by the name Mi) and in this case each instantiation acts as a different module

Verilog is used not only to specify hardware components and behaviours but also to specify test data. The module `DTYPE_TEST_DATA` specifies some test signals:

```
module DTYPE_TEST_DATA (ck,d);
  output ck,d; reg ck,d;

  initial begin
        ck = 0; // time 0
    #5  d  = 1; // time 5
    #5  ck = 1; // posedge ck at time 10
    #10 ck = 0; // negedge ck at time 20
    #5  d  = 0; // time 25
    #5  ck = 1; // posedge ck at time 30
    #5  d  = 1; // time 35
  end

endmodule
```

where `#5` is a `delay` statement, and for instance `#5 d = 1` means that the procedural assignment `d = 1` will be done after 5 simulation time units. Everything following \\ on a line is a comment.

The following module compares the behavioural and structural specification of the DTYPE

```
module DTYPE_TEST ();
  wire ck,d,q1,q2;

  DTYPE_TEST_DATA M1 (ck,d);
  DTYPE           M2 (ck,d,q1);
  DTYPE_IMP       M3 (ck,d,q2);

  initial
    $monitor ("Time = %0d, ck = %b, d = %b, q1 = %b, q2 = %b",
                       $time, ck, d, q1, q2);

endmodule
```

where `$monitor` is a directive to the simulator to print out the values on the wires whenever they change and `$time` is a system variable that prints out the simulation time (advanced by the `delay` (#) statement.

The output of the simulation is

```
Time = 0,  ck = 0, d = x, q1 = x, q2 = x
Time = 5,  ck = 0, d = 1, q1 = x, q2 = x
Time = 10, ck = 1, d = 1, q1 = 1, q2 = 1
Time = 20, ck = 0, d = 1, q1 = 1, q2 = 1
Time = 25, ck = 0, d = 0, q1 = 1, q2 = 1
Time = 30, ck = 1, d = 0, q1 = 0, q2 = 0
Time = 35, ck = 1, d = 1, q1 = 0, q2 = 0
```

This test shows that `DTYPE` and `DTYPE_IMP` are equivalent (with the test data provided) and also illustrates a theoretical (more specifically, semantical) challenge of Verilog: it would be desirable that this equivalence were formally proved, and for that a formal semantics is needed.

4.1.3 Example

This example shows that Verilog programs (having shared variables) can lead to a *race condition*: the semantics does not uniquely determine the result. In Verilog there is a main module which has *items* (or *blocks*) running in parallel in separate threads of computation. The items can be continuous assignments (see Example 4.1.1), instances of other modules (see module `DTYPE_IMP` in Example 4.1.2), `initial`-statements or `always`-statements (see module `DTYPE` in Example 4.1.2).

```
module INTERLEAVING ();
  integer x;

  initial
    begin
      x = 0;
      x = x+1;
    end

  initial x = 2;

endmodule
```

We have here two initial blocks running in parallel whose result can be `x=1` or `x=2` depending on which `initial` block is first executed, or even `x=3` if the second initial block is executed after the `x = 0` assignment of the first block, but before the execution of `x = x+1`.

4.1.4 Example

We present here an example of *non-blocking assignment* and blocking assignment with *zero delay*. Consider the following Verilog program:

```
module control;

reg [7:0]  x, y, z;

initial
  $display (" Time      x  Statement");

initial
begin
  x = #0 1;
  $write ("%5.0f  %5.0f", $time, x);
  $display ("      1");

  x <= #0 x + 1;
```

```
  $write ("%5.0f  %5.0f", $time, x);
  $display ("      2");

  x = 5;
  $write ("%5.0f  %5.0f", $time, x);
  $display ("      3");

  #1;
  $write ("%5.0f  %5.0f", $time, x);
  $display ("      4");

end

endmodule
```

We have two `initial` blocks running in parallel. The first one just writes the first line of the output[1] and in the second block we have four statements: a blocking assignment with zero delay, a non-blocking assignment with zero delay, a simple blocking assignment without delay and a `delay` statement. The `$write` command just output a string (between quotations marks) and the value of some variables (separated by commas) in the screen. The `$display` command is like `$write` but produce a new line. In the second block, the expression on the right hand side of the assignment `x= #0 1` is evaluated (just gives "1") and "blocks" the statement for 0 time units. After that, the second statement is executed: the expression `x + 1` is evaluated (the result is 2) but the result is not copied at this moment, it is enqueued for 0 time units and the assignment is done just at the end of the current slot time, i.e. after all the active and inactive events were executed (see the simulation semantics in Section 4.2). As the process is not "blocked" it can continue the execution with the next instruction (`x = 5`) and finally with the `delay` statement that advances the time in one time unit and the final value of `x` is displayed. The output is the following:

```
 Time       x  Statement
    0       1       1
    0       1       2
    0       5       3
    1       2       4
```

It is worth mentioning that a blocking assignment with delays can have the following two forms: $\#n\ v = e$ and $v = \#n\ e$. In the first case the process is blocked for n time units, then the expression e is evaluated and its value is copied to v whereas in the second case the expression e is evaluated in the current slot time and the value is copied to v after n time units. In this

[1] Even though the execution of the `initial` blocks is non-deterministic, the simulator first chooses one sequence of execution and will act deterministically after that: it will execute statements in the whole process always in the same order, unless we change their order in the source code.

example we use zero delay in the non-blocking assignment but the general form of the sentence is $v <= \#n\ e$ with $n \geq 0$. The important aspect to remember is that the expression (in the right hand side) of a non-blocking assignment is evaluated in the current simulation time, but the assignment is done after all the active and inactive events scheduled for time slot n.

4.2 Scheduling Semantics

The "normal" semantics of Verilog is based on the scheduling of events (changes in the values of wires and registers), that is, it is based on a simulation algorithm. This simulation (event or scheduling) semantics is explained in the Verilog IEEE Standard Report 1364 [10, Section 5]. According to this report, the simulation algorithm is based on a stratified queue of events segmented in the following five regions:

1. *Active events*: which occur at the current simulation time and can be processed in any order
2. *Inactive events*: which occur at the current simulation time but after all the active events are processed
3. *Non-blocking assignment update events*: which have been evaluated during some previous (or the same) simulation time, but must be assigned at this simulation time but after all the active and inactive events
4. *Monitor events*: which must be processed after all the above events
5. *Future inactive events* and *future non-blocking assignment update events*: which occur at some future simulation time.

Maybe one of the most important theoretical challenge of Verilog is its lack of a formal semantics that would allow, among other things, to prove some properties of programs or to develop verification conditions to ensure that two specifications are equivalent as for example the behavioural and structural version of the the DTYPE in Example 4.1.2. For other semantic challenges of Verilog refer to [7]).

There are not so many published papers attempting to give formal semantics to Verilog (as far as we know) and we will discuss briefly here just Gordon's [8] and Pace's [12] approach.

Gordon discuss in his draft ([8]) three different semantical approach to the $V0$ subset of Verilog: Event semantics, Trace semantics and Cycle semantics. The Trace semantics is based basically on the transformation of each statement of the module (the program is transformed to have just one module) applying algorithms and obtaining what he call the *steps* and finally the trace specification. The trace specification of a statement is a predicate where the program counter is existentially quantified and the time variable is totally quantified and the trace semantics of a module is the conjunction of the predicates corresponding to each statement. It is important to remark that Gordon's work is a draft and can not be taken as complete research but it looks complicated and it seems that it cannot be easily simplified.

Pace's approach [12] is based on the Relational Duration Calculus [12, Chapter 3] that seems to be equivalent to the Super-dense Duration Calculus (SDC) [13] (in fact both are "hybrid duration calculus"). These two logics are useful to give semantics to languages where what matters is the initial and final state at a given time point (each time point can be split only into two points representing the input and the output value of the variables at that moment). In case we want to express in the logic all the intermediate steps on instantaneous (zero-duration) computations we need a logic as WDC and this is one of the difference between Pace's approach and ours. Another difference is that he use a prefix-closed semantics whereas our semantics is maximal (see Section 6.1). It is important mentioning that Pace gives semantics to the non-blocking assignment but giving to it the same priority as the blocking assignment, i.e. if both $v_1 = \#n\ e_1$ and $v_2 <= \#n\ e_2$ are scheduled to be executed on the simulation time n, then one of them will be executed first in a non-deterministic way whereas in the standard Verilog (see the scheduling semantics above) the non-blocking statement should be executed after all the blocking assignment scheduled for that time. In fact, in Pace's semantics the order between two pending non-blocking assignment is not preserved, that is, if $v_1 <= \#n\ e_1$ and $v_2 <= \#n\ e_2$ are scheduled to be executed on the simulation time n, the assignment to v_2 can be done first, that contradicts the scheduling semantics. From the comments above it is easy to see that the semantics of the non-blocking assignment is rather complicated and that is why we excluded it from the V^- in a first treatment.

As a final remark we can say that both Gordon and Pace impose some syntactical restrictions on the language with the aim of facilitate its treatment.

5 The V^- subset

In this section we introduce the subset of Verilog called V^-. In the first section we give the syntax of the language, while a formal semantics is defined in the second and third sections: an operational semantics in the second one and a logical one, using WDC, in the third section.

5.1 The syntax

Gordon introduced the $V0$ ([8]) and V ([7],[12]) subset of Verilog and for simplicity we take a subset of V, called V^-, that has esentially the same statements as V except the function definition, the non-blocking assignment ($v <= e$), the `disable` statement and the delays in the assignment statements. The syntax is presented in Figure 5.

[1] In Verilog the parallel composition between blocks is written ";" but to avoid confusion with the sequential operator (that has the same denotation, ";") we use here the "‖" symbol

$$
\begin{array}{llll}
\langle number \rangle & & & \text{(Natural numbers)} \\
\langle variable \rangle & & & \text{(Variables)} \\
\langle exp \rangle & & & \text{(Arithmetical expressions)} \\
\langle bexp \rangle & & & \text{(Boolean expressions)} \\
\langle var-list \rangle & ::= & \langle variable \rangle & \text{(List of variables)} \\
 & \mid & \langle variable \rangle \, , \langle var-list \rangle & \\
\langle exp-list \rangle & ::= & \langle exp \rangle & \text{(List of expressions)} \\
 & \mid & \langle exp \rangle \, , \langle exp-list \rangle & \\
\langle guard \rangle & ::= & @(\langle variable \rangle) & \text{(Change of value)} \\
 & \mid & @(\texttt{posedge}\ \ \langle variable \rangle) & \text{(Positive edge)} \\
 & \mid & @(\texttt{negedge}\ \ \langle variable \rangle) & \text{(Negative edge)} \\
 & \mid & @(\langle variable \rangle\ \texttt{or}\ \ \ldots \texttt{or}\ \ \langle variable \rangle) & \text{(Compound sensitive list)} \\
\langle statement \rangle & ::= & \texttt{skip} & \text{(Empty statement)} \\
 & \mid & \langle variable \rangle = \langle exp \rangle & \text{(Assignment)} \\
 & \mid & \langle statement \rangle \, ; \langle statement \rangle & \text{(Sequential composition)} \\
 & \mid & \texttt{begin}\ \ \langle statement \rangle\ \ \texttt{end} & \text{(Block statement)} \\
 & \mid & \texttt{if}\ \ (\langle bexp \rangle)\, \langle statement \rangle & \\
 & & \quad \texttt{else}\ \ \langle statement \rangle & \text{(Conditional)} \\
 & \mid & \texttt{case}\ \ (\langle exp \rangle) & \text{(Case statement)} \\
 & & \quad \langle exp \rangle : \langle statement \rangle \, ; & \\
 & & \quad \vdots & \\
 & & \quad \langle exp \rangle : \langle statement \rangle \, ; & \\
 & & \quad \texttt{default} : \langle statement \rangle & \\
 & & \quad \texttt{endcase} & \\
 & \mid & \texttt{while}\ \ (\langle bexp \rangle)\ \ \langle statement \rangle & \text{(While-statement)} \\
 & \mid & \texttt{repeat}\ \ (\langle number \rangle)\ \ \langle statement \rangle & \text{(Repeat-statement)} \\
 & \mid & \texttt{for}\ \ (\langle variable \rangle = \langle exp \rangle \, ; \langle bexp \rangle \, ; & \\
 & & \qquad \langle variable \rangle = \langle exp \rangle)\, \langle statement \rangle & \text{(For-statement)} \\
 & \mid & \texttt{forever}\ \ \langle statement \rangle & \text{(Forever-statement)} \\
 & \mid & \#\, \langle number \rangle & \text{(Delay)} \\
 & \mid & \langle guard \rangle\, \langle statement \rangle & \text{(Timing control)} \\
\langle module \rangle & ::= & \texttt{initial}\ \ \langle statement \rangle & \text{(Initial block)} \\
 & \mid & \texttt{always}\ \ \langle statement \rangle & \text{(Always block)} \\
 & \mid & \texttt{assign}\ \ \langle variable \rangle = \langle exp \rangle & \text{(Continuous assignment)} \\
 & \mid & \langle module \rangle \parallel \langle module \rangle & \text{(Parallel composition)}
\end{array}
$$

Figure 5: The syntax of $V0_0$.

A complete specification in V^- consists of a single module of the general form:

$$\begin{array}{l} \texttt{module}\ \langle module - name\rangle\,(\langle port - name\rangle\,,\ldots,\langle port - name\rangle); \\ \quad \langle module\rangle \parallel \ldots \parallel \langle module\rangle \\ \ \ \texttt{endmodule} \end{array}$$

Notice that we have just top level parallelism, so we cannot have V^- programs like $(P_1 \parallel P_2); P_3$.

We have left under-specified the first four syntax categories (namely: numbers, variables, expressions and boolean expressions). We do not also make any difference between *wires* and *procedural variables* except for the case of continuous assignment where the variable in the left hand side must be a wire. An important remark here is that Verilog has the following *syntactical restriction*: a wire cannot be in the left hand side of a procedural assignment. Another syntactical restriction (of V^-) is that two continuous assignment to the same wire are not allowed. Even though this is not forbidden in Verilog, some simulators will give unexpected results and of course it does not have neither an informal nor a formal semantics.

Notation: v, v_i range over variables, w over wires, S, S_i over statements, n and i over natural numbers, eb over boolean expressions, e, e_i over arithmetic expressions and P, P_i over processes (modules, blocks or programs).

Note: In general in the examples that follow we will write S instead of `initial` S.

5.2 An operational semantics

We present here an operational semantics of V^- based on the work of Plotkin [19] but using the idea of distinguishing component and environment actions. Before giving the semantics of the statements we need to introduce some preliminaries concepts, notations and assumptions about the language. We need first the notion of *labelled transition systems.*

Definition 5.1 *A* labelled transition system *is a structure* $\langle \Gamma, A, \rightarrow \rangle$*, where* Γ *is a set of* configurations*,* A *is a set of* actions *(labels or operations) and* $\rightarrow$ *is a relation (called the* transition relation *such that:*

$$\rightarrow \subseteq \Gamma \times A \times \Gamma$$

□

In particular, $\Gamma = \langle P, \sigma \rangle$, where P is the program to be executed and σ is the state from which P is going to run. Here $\sigma \in \Sigma = Var \rightarrow Val$ denotes an assignment of values (reals and booleans) to variables. If x is a variable and v is a value, then the state resulting from assigning v to x

is denoted by $\sigma[v/x]$. The end of a program is represented by ϵ. A transition is represented by one of the following three forms:

$\langle P, \sigma \rangle \xrightarrow{i} \langle P', \sigma' \rangle$ for a state move from the component itself,
$\langle P, \sigma \rangle \xrightarrow{p} \langle P', \sigma' \rangle$ for a passive move,
$\langle P, \sigma \rangle \xrightarrow{t} \langle P', \sigma' \rangle$ for a time advanced step.

The standard way to give a compositional semantics to shared variable programs, suggested first by Aczel (cited e.g., in [4]), is to define the semantics as a set of labelled state transition sequences, where the label records whether the transition is from the environment or from the component.Our presentation is slightly different since we have let the environment transition be implicite (we do not have a label for it) mainly because we have only top level parallelism. The label p represents a "passive" transition that is particularly useful to distinguish transitions of processes that have to "progress simultaneously" with the parallel processes from the ones that can be represented with an "interleaving semantics". We have also introduced another label (t) to denote time advance, which only occur when both the component and the environment are executing some `delay` statement.

We have the following assumptions about the language:

- Only the `delay` statement ($\#n$) can take time. The other statements are executed in *zero* time (the time here is weakly monotonic as defined in Definition 2.1).
- The assignment is atomic, i.e., whenever $x = e$ and e is evaluated, then the value of e does not change before assigned to x.

The transition rules for the statements are as follows.

Skip. The `skip` command has the following operational semantics.

$$\langle \texttt{skip}, \sigma \rangle \xrightarrow{i} \langle \epsilon, \sigma \rangle$$

Its intuitive meaning is just that after execution, the state remains the same, and it takes no time.

Procedural assignment. The syntax of the procedural assignment is:

$$v = e$$

Its meaning is just that the expression e is evaluated and then its value is copied to the left-hand-side variable v, which means that the state has changed (just the variable v).

$$\frac{\langle e, \sigma\rangle \xrightarrow{i} \langle m, \sigma\rangle}{\langle v = e, \sigma\rangle \xrightarrow{i} \langle \epsilon, \sigma[m/v]\rangle}$$

where $m \in Val$.

Sequential composition. The syntax is:

`begin` $S_1; \ldots ; S_n$ `end`

The sequential composition is just the execution of the commands one after the other, but with the possibility of some interaction of the environment. That is why we have the first rule. Moreover, the transitions can take time and this is represented here allowing the label to be either i or t.

1. $$\frac{\langle S_1, \sigma\rangle \xrightarrow{\delta} \langle S_1', \sigma'\rangle \wedge S_1' \neq \epsilon}{\langle \texttt{begin } S_1; S_2; \ldots ; S_n \texttt{ end}, \sigma\rangle \xrightarrow{\delta} \langle \texttt{begin } S_1'; S_2; \ldots ; S_n \texttt{ end}, \sigma'\rangle}$$

2. $$\frac{\langle S_1, \sigma\rangle \xrightarrow{\delta} \langle \epsilon, \sigma'\rangle}{\langle \texttt{begin } S_1; S_2; \ldots ; S_n \texttt{ end}, \sigma\rangle \xrightarrow{\delta} \langle \texttt{begin } S_2; \ldots ; S_n \texttt{ end}, \sigma'\rangle}$$

where $\delta = \{i, t\}$ means that the transition is executed by the component or it takes time.

If-then-else. The `if-then-else` command,

`if` $(eb)\ S_1$ `else` S_2

has the usual meaning: if the expression eb evaluates to `true`, then S_1 is executed, otherwise S_2 is executed.

1. $$\frac{\langle eb, \sigma\rangle \xrightarrow{i} \langle \mathsf{true}, \sigma\rangle}{\langle \texttt{if } (eb)\ S_1 \texttt{ else } S_2, \sigma\rangle \xrightarrow{i} \langle S_1, \sigma\rangle}$$

2. $$\frac{\langle eb,\sigma\rangle \xrightarrow{i} \langle \mathsf{false},\sigma\rangle}{\langle \texttt{if}\ \ (eb)\, S_1\ \ \texttt{else}\ \ S_2,\sigma\rangle \xrightarrow{i} \langle S_2,\sigma\rangle}$$

Notice that the evaluation of the expression eb does not change the environment (assumption of the rules). The `if-then` conditional command is easily defined as

$$\texttt{if}\ \ (eb)\, S_1 \stackrel{\text{def}}{=} \texttt{if}\ \ (eb)\, S_1\ \ \texttt{else skip}$$

Case. The intuitive meaning of the `case` statement,

$$\texttt{case}\ \ (e)\, e_1 : S_1; \ldots ; e_n : S_n; \{\texttt{default} : S_{n+1}\}\ \ \texttt{endcase}$$

is that after evaluating the expression e, its value match with one of the e_i expression, then the S_i statement is executed, otherwise the statement S_{n+1} is executed (notice that the `default` part is optional).

1. $$\frac{\langle e,\sigma\rangle \xrightarrow{i} \langle e_h,\sigma\rangle}{\langle \texttt{case}\ \ (e)\, e_1 : S_1; \ldots ; e_n : S_n; \{\texttt{default} : S_{n+1}\},\sigma\rangle \xrightarrow{i} \langle S_h,\sigma\rangle}$$

2. $$\frac{\langle e,\sigma\rangle \xrightarrow{i} \langle e',\sigma\rangle\ \wedge\ e' \neq e_h\ \forall\, 1 \leq h \leq n}{\langle \texttt{case}\ \ (e)\, e_1 : S_1; \ldots ; e_n : S_n; \{\texttt{default} : S_{n+1}\},\sigma\rangle \xrightarrow{i} \langle S_{n+1},\sigma\rangle}$$

While. The `while` statement has the following syntax:

$$\texttt{while}\ \ (eb)\, S$$

where the boolean expression eb is evaluated (again without changing the values of the variables) and if it is true then the statement S is executed (first rule). The second rule express that whenever eb is evaluated to false then the command finish.

1. $$\frac{\langle eb,\sigma\rangle \xrightarrow{i} \langle \mathsf{true},\sigma\rangle}{\langle \texttt{while}\ \ (eb)\, S,\sigma\rangle \xrightarrow{i} \langle S; \texttt{while}\ \ (eb)\, S,\sigma\rangle}$$

2. $$\frac{\langle eb,\sigma\rangle \xrightarrow{i} \langle \mathsf{false},\sigma\rangle}{\langle \texttt{while}\ \ (eb)\, S,\sigma\rangle \xrightarrow{i} \langle \epsilon,\sigma\rangle}$$

For. The `for` statement

$$\texttt{for}\ (v_1 = e_1; eb; v_2 = e_2)\ S$$

can be seen as a particular case of the `while` statement with the following definition:

$$\texttt{for}\ (v_1 = e_1; eb; v_2 = e_2)\ S \stackrel{\text{def}}{=} v_1 = e_1; \texttt{while}\ (eb)\ \texttt{begin}\ S; v_2 = e_2\ \texttt{end}$$

So, the operational semantics is easily defined applying the semantics of the sequential composition and `while` .

Repeat. The syntax of the `repeat` statement is:

$$\texttt{repeat}\ (n)\ S;$$

The intuitive meaning is just that S is executed n times. If $n = 0$, then S is not executed at all.

1. $$\frac{n = 0}{\langle \texttt{repeat}\ (n)\ S, \sigma\rangle \xrightarrow{i} \langle \epsilon, \sigma\rangle}$$
2. $$\frac{n > 0}{\langle \texttt{repeat}\ (n)\ S, \sigma\rangle \xrightarrow{i} \langle S; \texttt{repeat}\ (n-1)\ S, \sigma\rangle}$$

Forever. Syntax:

$$\texttt{forever}\ S;$$

The forever statement means that the statement S is executed "forever" (it is the same as `while` (true) S).

$$\langle \texttt{forever}\ S, \sigma\rangle \xrightarrow{i} \langle S; \texttt{forever}\ S, \sigma\rangle$$

Delay. The `delay` statement is a timing control statement, but we present it in a separate paragraph because its operational semantics is different from the other timing control statements. It is the only statement that can introduce time passing in V^-. Its syntax is:

$\#n;$

Its meaning is that whenever this instruction is executed, then the process is blocked at the point the instruction appear and the next instruction (if exists) could be executed after n time units. It is important noting that we said "could be executed" because we allow infinite computations and can occur that one process enter in an infinite loop before the n time units. In this case the next instruction would never be executed. To represent this we have the following two rules:

1. $$\frac{\sigma \xrightarrow{n} \sigma'}{\langle \#n, \sigma\rangle \xrightarrow{n} \langle \epsilon, \sigma'\rangle}$$

2. $$\frac{\sigma \xrightarrow{n'} \sigma' \wedge n' < n}{\langle \#n, \sigma\rangle \xrightarrow{n} \langle \#(n-n'), \sigma'\rangle}$$

where $n = t$ and $\sigma \xrightarrow{n} \sigma'$ means that n time units have passed and the environment has possible changed the state. The first rule said that if no infinite computation has occurred (done by the environment), then after n time units the command is completed (ϵ), while the second one express that the time has advanced n' time units. The last one formalise the possibility of an interaction of the environment.

Timing control. We have four cases of Timing control: waiting for a variable change, a positive change (a greater value than the actual one), a negative change (a smaller value than the actual one) or a composite change (in one of the variables of the argument). Below we present the syntax and the rules for the operational semantics for each case.

Syntax:

$@(v)\ S;$

Semantics:

1. $$\frac{\sigma(v) \neq \sigma'(v)}{\langle @(v)\ S, \sigma\rangle \xrightarrow{p} \langle S, \sigma'\rangle}$$

2. $$\frac{\sigma(v) = \sigma'(v)}{\langle @(v)\ S, \sigma\rangle \xrightarrow{\delta} \langle @(v)\ S, \sigma'\rangle}$$

where $\delta = \{i, t\}$ is as explained before and p represent a "passive" transition that it is used to differentiate it from the component transition i in the parallel composition rule. Here $\sigma(v) = \sigma'(v)$ means that the variable v has not changed, whereas $\sigma(v) \neq \sigma'(v)$ has the obvious meaning. The environment transition (that changes the state) is not explicitly written, but it is understood from the premises of the rules. Note that we do not have a rule for time passing when the variable v is changed. This is not allowed because if v changes then the time control must be activated in the same instant.

In the rest of the cases no explanation is needed because the semantics is quite the same, it has just a few differences: instead of representing an arbitrary change we have to express a positive change in the first case, a negative one in the second and the change of one of the disjunctions in the last case.

Syntax of the positive change of the variable:

$@(\texttt{posedge}\ \ v)\ S;$

Semantics:

1. $$\frac{\sigma(v) < \sigma'(v)}{\langle @(\texttt{posedge}\ \ v)\ S, \sigma\rangle \xrightarrow{p} \langle S, \sigma'\rangle}$$

2. $$\frac{\sigma(v) \geq \sigma'(v)}{\langle @(\texttt{posedge}\ \ v)\ S, \sigma\rangle \xrightarrow{\delta} \langle @(\texttt{posedge}\ \ v)\ S, \sigma'\rangle}$$

Negative edge:

$@(\texttt{negedge}\ \ v)\ S;$

Semantics:

1. $$\frac{\sigma(v) > \sigma'(v)}{\langle @(\texttt{negedge}\ \ v)\ S, \sigma\rangle \xrightarrow{p} \langle S, \sigma'\rangle}$$

2. $$\frac{\sigma(v) \leq \sigma'(v)}{\langle @(\texttt{negedge}\ \ v)\ S, \sigma\rangle \xrightarrow{\delta} \langle @(\texttt{negedge}\ \ v)\ S, \sigma'\rangle}$$

Syntax of the compound timing control:

$$@(v_1 \texttt{or} \ \ldots \texttt{or} \ v_n)\, S;$$

Semantics:

1. $$\frac{\exists h, 1 \leq h \leq n \cdot \sigma(v_h) \neq \sigma'(v_h)}{\langle @(v_1 \texttt{or} \ \ldots \texttt{or} \ v_n)\, S, \sigma\rangle \xrightarrow{p} \langle S, \sigma'\rangle}$$

2. $$\frac{\forall h, 1 \leq h \leq n \cdot \sigma(v_h) = \sigma'(v_h)}{\langle @(v_1 \texttt{or} \ \ldots \texttt{or} \ v_n)\, S, \sigma\rangle \xrightarrow{\delta} \langle @(v_1 \texttt{or} \ \ldots \texttt{or} \ v_n)\, S, \sigma'\rangle}$$

We present now the semantics of the module items: `initial` block, parallel composition, continuous assignment and the `always` command.

Initial. The initial block is, as its name suggest, a block of commands that is executed initially. Its syntax is:

$$\texttt{initial} \ S;$$

Its semantics express just that S is executed once:

$$\langle \texttt{initial} \ S, \sigma\rangle \xrightarrow{i} \langle S, \sigma\rangle$$

Parallelism. Syntax:

$$P_1 \parallel P_2$$

We present here just the parallel composition between two processes for simplicity of presentation but it can be easily extended to an arbitrary number of processes. We model parallelism using interleaving semantics instead of true concurrency. The labelled transition system presented here with the labels ranging over component (i) movements, the time passing (t) and the passive transition (p) with the interleaving assumption, allow us to avoid the explicit naming of the shared variables. An operational semantics for shared variables with explicit mention about the read and written variables for each process is presented in [3] . The semantics is expressed using the following nine rules:

1. $$\frac{\langle P_1, \sigma\rangle \xrightarrow{i} \langle P_1', \sigma'\rangle}{\langle P_1 \parallel P_2, \sigma\rangle \longrightarrow \langle P_1' \parallel P_2, \sigma'\rangle}$$

2. $$\frac{\langle P_2, \sigma\rangle \xrightarrow{i} \langle P_2', \sigma'\rangle}{\langle P_1 \parallel P_2, \sigma\rangle \longrightarrow \langle P_1 \parallel P_2', \sigma'\rangle}$$

3. $$\frac{\langle P_1, \sigma\rangle \xrightarrow{i} \langle \epsilon, \sigma'\rangle}{\langle P_1 \parallel P_2, \sigma\rangle \longrightarrow \langle P_2, \sigma'\rangle}$$

4. $$\frac{\langle P_2, \sigma\rangle \xrightarrow{i} \langle \epsilon, \sigma'\rangle}{\langle P_1 \parallel P_2, \sigma\rangle \longrightarrow \langle P_1, \sigma'\rangle}$$

5. $$\frac{\langle P_1, \sigma\rangle \xrightarrow{t} \langle P_1', \sigma'\rangle \wedge \langle P_2, \sigma\rangle \xrightarrow{t} \langle P_2', \sigma'\rangle}{\langle P_1 \parallel P_2, \sigma\rangle \xrightarrow{t} \langle P_1' \parallel P_2', \sigma'\rangle}$$

6. $$\frac{\langle P_1, \sigma\rangle \xrightarrow{i} \langle P_1', \sigma'\rangle \wedge \langle P_2, \sigma\rangle \xrightarrow{p} \langle P_2', \sigma''\rangle}{\langle P_1 \parallel P_2, \sigma\rangle \longrightarrow \langle P_1' \parallel P_2', \sigma'\dagger\sigma''\rangle}$$

7. $$\frac{\langle P_1, \sigma\rangle \xrightarrow{p} \langle P_1', \sigma'\rangle \wedge \langle P_2, \sigma\rangle \xrightarrow{i} \langle P_2', \sigma''\rangle}{\langle P_1 \parallel P_2, \sigma\rangle \longrightarrow \langle P_1' \parallel P_2', \sigma''\dagger\sigma'\rangle}$$

8. $$\frac{\langle P_1, \sigma\rangle \xrightarrow{i} \langle \epsilon, \sigma'\rangle \wedge \langle P_2, \sigma\rangle \xrightarrow{p} \langle P_2', \sigma''\rangle}{\langle P_1 \parallel P_2, \sigma\rangle \longrightarrow \langle P_2', \sigma'\dagger\sigma''\rangle}$$

9. $$\frac{\langle P_1, \sigma\rangle \xrightarrow{p} \langle P_1', \sigma'\rangle \wedge \langle P_2, \sigma\rangle \xrightarrow{i} \langle \epsilon, \sigma''\rangle}{\langle P_1 \parallel P_2, \sigma\rangle \longrightarrow \langle P_1', \sigma''\dagger\sigma'\rangle}$$

We separate the explanation of the rules into two groups because the last four rules deal with the "passive" transition p. The first two rules model the case where P_1 (P_2) has been executed partially, which allow interleaving: another process can run before P_1 (P_2) has finished its execution. The third (forth) rule shows the case where P_1 (P_2) has already finish its execution and then just remain to execute P_2 (P_1). Rule 5. shows the case when the time advance for both process simultaneously.

The $\dagger$ (override) operator has the following meaning: $(\sigma'\dagger\sigma'')(x)$ is equal to $\sigma'(x)$ for all x such that $\sigma'(x) = \sigma''(x)$ and equal to $\sigma''(x)$ if $\sigma'(x) \neq \sigma''(x)$. The following four rules deals with passive transition, that can be done "simultaneously" with others and it is particularly useful in the semantics of the timing control and continuous assignment commands. These statements have the characteristic that when they are composed in parallel with another processes the may have to "advance together". For example, it would be desirable that the following transition were deduced from the operational semantics:

- $\langle (@(v)\ S \ \parallel\ v = v+1), \sigma\rangle \longrightarrow \ldots \longrightarrow \langle S, \sigma[(v+1)/v]\rangle$
- $\langle \texttt{assign}\ w = x \ \parallel\ x = 1; \#1; x = 2), \sigma\rangle \longrightarrow \ldots \longrightarrow \langle \texttt{assign}\ w = x, \sigma[2/x, 2/w]\rangle$

It is clear that with the first five rules above described it is not possible to guarantee these results. In particular, in the first case and using rules 1. to 5. we can have just the following results:

- $\langle (@(v)\ S \ \parallel\ v = v+1), \sigma\rangle \longrightarrow \ldots \longrightarrow \langle @(v)\ S, \sigma[(v+1)/v]\rangle$
- $\langle (@(v)\ S \ \parallel\ v = v+1), \sigma\rangle \longrightarrow \ldots \longrightarrow \langle (@(v)\ S \ \parallel\ v = v+1), \sigma\rangle$

that are both incorrect.

It is important to notice that we have here some unlabelled transition because we have considered just top level parallelism and then we can see a program in V^- as a set of processes running in parallel and as a closed system, in such a way that we do not need to say that the transition is done by the process.

Continuous assignment. Syntax:

$\texttt{assign}\ w = e;$

In the continuous assignment w is a *wire* and not a variable (we have not made any syntactical distinction between the wires and variables). The wires (as seen in Example 4.1.1) are like channels that do not store values, just conducted them. The continuous assignment "$\texttt{assign}\ w = e$" continuously watches for changes to variables in its right hand side (variables that occur on the expression e) and whenever a change happens the right hand side is re-evaluated and the result is immediately propagated to the left hand side (w). The operational semantics then is as follows:

1. $$\frac{\bigvee_{v\in FV(e)} \sigma(v) \neq \sigma'(v) \wedge \langle e, \sigma'\rangle \xrightarrow{i} \langle m, \sigma'\rangle}{\langle \texttt{assign}\ w = e, \sigma\rangle \xrightarrow{p} \langle \texttt{assign}\ w = e, \sigma'[m/w]\rangle}$$

2. $$\frac{\bigwedge_{v\in FV(e)} \sigma(v) = \sigma'(v)}{\langle \texttt{assign}\ w = e, \sigma\rangle \xrightarrow{t} \langle \texttt{assign}\ w = e, \sigma'\rangle}$$

where $FV(e)$ is the set of free variables appearing in the expression e. Notice that the first rule says that if the environment (expressed implicitly) change at least one of the variables occurring in e and it is evaluated to m (in the new state) then the wire w is equal to m. As in the timing control statements we have that the transtition is passive: after it we have the same `assign` statement. The second rule denote the time advancing case.

Always. Syntax:

always S;

The `always` block is like the `forever` statement, where S is always executed, then its semantics is:

$$\langle \texttt{always } S, \sigma\rangle \xrightarrow{i} \langle S; \texttt{always } S, \sigma\rangle$$

We finish here the operational semantics of V^-. In the next section we present a more abstract semantics based on *WDC*.

5.3 An abstract compositional semantics

In this section we give semantics to the V^- subset of Verilog allowing *interleaving* between sequential processes. The main features of the approach given here is that it is abstract (logical) and compositional. Our semantics is *maximal* (in the sense that the interval considered is that of the whole execution of the program) instead of *prefix-closed* as the ones of Pandya [15, Section 5] and Pace [12] (see also Section 6.1, page 51 of this work).

At the top level, a V^- program can be considered as a collection of concurrent processes, communicating through shared variables. The standard way to give a compositional semantics to shared variable programs, suggested first by Aczel (cited e.g., in [4]), is to define the semantics as a set of labelled state transition sequences, where the label records whether the transition is from the environment or from the component. This can be expressed in the setting of μWDCI by introducing a variable, say ∂, in the state, to record the transition agent. More precisely, let $\{P_i \mid i \in I_{Proc}\}$ be the set of processes, where I_{Proc} is a set of indexes. If there is a transition at $[(r,n),(r,n+1)]$, then the transition is from process P_i iff $\mathcal{I}(\partial)(r,n+1) = i$.

The semantics can be thought as for the program concerned defining a set of *runs*, each of which is a sequence of states $\sigma : \mathcal{T} \to \mathsf{Var}^+ \to \mathsf{Values}$ over a time domain, where $\mathsf{Var}^+ \overset{\text{def}}{=} \mathsf{Var} \cup \{\partial\}$ and *Var* is the set of all program variables. For example, consider the program $P_1 \parallel P_2$, where

$$
\begin{array}{ll}
P_1: & \#1; \\
 & x = 1; \\
 & y = 3; \\
 & x = x + 2; \\
 & \#3; \\
 & x = y + x;
\end{array}
$$

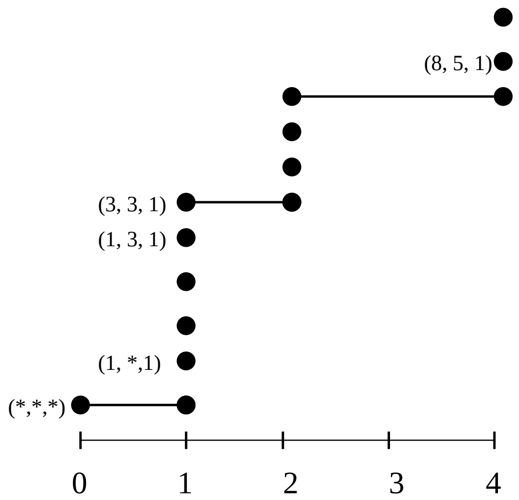

Figure 6: A run of P_1 as an Open System.

and

$$
\begin{array}{ll}
P_2: & \#2; \\
 & y = 5;
\end{array}
$$

That can produce the following sequence of states:

$$
\begin{array}{l}
\sigma(0,0) = \{x \mapsto *, y \mapsto *, \partial \mapsto *\}, \\
\sigma(1,0) = \{x \mapsto *, y \mapsto *, \partial \mapsto *\}, \\
\sigma(1,1) = \{x \mapsto 1, y \mapsto *, \partial \mapsto 1\}, \\
\sigma(1,2) = \{x \mapsto 1, y \mapsto 3, \partial \mapsto 1\}, \\
\sigma(1,3) = \{x \mapsto 3, y \mapsto 3, \partial \mapsto 1\}, \\
\sigma(2,3) = \{x \mapsto 3, y \mapsto 3, \partial \mapsto 1\}, \\
\sigma(2,4) = \{x \mapsto 3, y \mapsto 5, \partial \mapsto 2\}, \\
\sigma(4,4) = \{x \mapsto 3, y \mapsto 5, \partial \mapsto 2\}, \\
\sigma(4,5) = \{x \mapsto 8, y \mapsto 5, \partial \mapsto 1\},
\end{array}
$$

where $*$ is the undefined value.

Above we regard the system $P_1 \parallel P_2$ as closed, that is, they are the only processes in the system. It is desirable to define the semantics compositionally, namely, the semantics of a compound statement is defined as a function of the semantics of the components. Therefore, when giving the semantics of P_i, it should be viewed as an open system, and its semantics also contains all the potential actions from its environment processes.The diagram in Figure 6 shows a possible run of P_1, which contains transitions from some potential processes (including P_2). Runs with unmatching environment actions are removed by the semantics of the parallel composition.

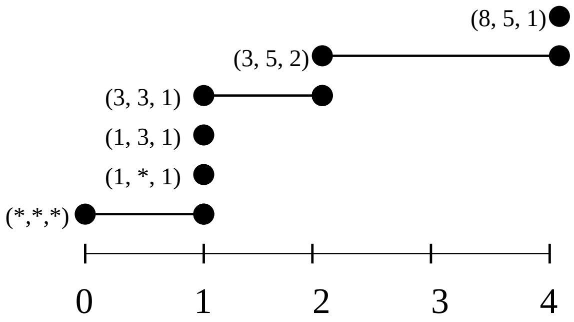

Figure 7: A run of $P_1 \parallel P_2$ as a Closed System.

Instead of defining the semantics as a set of runs directly, it is more effective to define it as a μWDCI formula characterising the runs. The semantics contains a general formula

$$\mathbf{Ax_1} : \forall x \in \mathsf{Var}^+ \ . \ \Box(\mathsf{cint} \Rightarrow \mathbf{b}.x = \mathbf{e}.x)$$

This says that none of the variables are changed if there are no transitions. The semantics of the whole system, considered as a closed system contains the formula:

$$\mathbf{Ax_2} : \ \exists i \in I_{Proc} \ . \ \Box(\mathsf{dint} \Rightarrow \mathbf{e}\partial = i),$$

This says that any transition is caused by one of the constituent process. Figure 7 represents the run of the example above as a closed system.

In the following, we assume that the considered sequential statements are from process P_i. As the semantics of an open system, it is necessary to include possible behaviours from other processes. The formula

$$\mathsf{idle_i} \stackrel{\mathrm{def}}{=} \Box(\mathsf{dint} \Rightarrow \mathbf{e}\partial \neq i)$$

says that over the interval, process P_i does not contribute any discrete transitions. There is no restriction over transitions by processes other than P_i, or in other words, transitions from other processes can be arbitrary.

Before defining the semantics of the statements and in spite of the under-specified syntax of the evaluation of boolean expressions, we will define their semantics first.

Boolean expressions. The meaning of the evaluation of a boolean expression is:

$$\mathcal{M}(B) \stackrel{\mathrm{def}}{=} \ell = 0 \wedge (\mathsf{idle_i}^\frown(\mathbf{b}.B \wedge \mathsf{dint} \wedge \mathit{unchange}_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i))$$

where $unchange_{\mathsf{Var}}$ means that the Global variables do not change.

Remark: In particular, V is the set of procedural variables and does not include the wires, otherwise there would be an inconsistence when trying to give semantics to the parallel composition and the continuous assignment (see the remark on page 39).

In particular we have

- $\mathcal{M}(\mathsf{false}) \stackrel{\text{def}}{=} \ell = 0 \land (\mathsf{idle_i} \frown \mathsf{false})$
- $\mathcal{M}(\mathsf{true}) \stackrel{\text{def}}{=} \ell = 0 \land (\mathsf{idle_i} \frown (\mathsf{dint} \land unchange_{\mathsf{Var}} \land \mathbf{e}.\partial = i))$

It would be worth to explain why we need to write an explicit semantics of the boolean expressions and do not let it under-specified. The main reason is because we want to allow interleaving between processes and avoid to use the $\mathsf{idle_i}$ preposition in non-atomic statements (see Section 7 in page 54). Another important remark here is the use of the $l = 0$ conjunct in the semantics of the boolean expressions. We find particularly useful to have it not only in the evaluation of boolean expressions but also in the atomic sentences because we want that these statements implies that $l = 0$ mainly when trying to prove some results using fixed point operators (see Section 8 in page 57). Another remark is that the evaluation of boolean expression takes one micro-time step (dint) because it is desirable to have $\mathcal{M}(\mathsf{true})$ equivalent to $\mathcal{M}(\mathtt{skip})$.

Skip. This is the simplest instruction: `skip` is executed by process i, takes one micro-time (dint) and does not change the value of the variables.

$$\mathcal{M}(\mathtt{skip}) \stackrel{\text{def}}{=} \ell = 0 \land (\mathsf{idle_i} \frown (\mathsf{dint} \land unchange_{\mathsf{Var}} \land \mathbf{e}.\partial = i))$$

Procedural assignment. The meaning of $v = e$ is the same as in imperative programming languages: the expression e is evaluated and then its value is assigned to the variable v (which has the usual meaning). This assignment is called *procedural* to distinguish it from the continuous assignment (`assign`).

$$\mathcal{M}(v = e) \stackrel{\text{def}}{=} \ell = 0 \land (\mathsf{idle_i} \frown (\mathsf{dint} \land \mathbf{e}.v = \mathbf{b}.e \land unchange_{\mathsf{Var}-\{v\}} \land \mathbf{e}.\partial = i))$$

Sequential composition. Thanks to the $\frown$ (chop) operator, the semantics of the sequential composition is easy to define: is just the chop of the statements involved. Notice that because of the $\mathsf{idle_i}$ at the beginning of the atomic statements, the interleaving is allowed between two sequential statements.

$$\mathcal{M}(\texttt{begin}\ S_1; S_2; \ldots; S_n\ \texttt{end}) \stackrel{\text{def}}{=} \mathcal{M}(S_1) \frown \mathcal{M}(\texttt{begin}\ S_2; \ldots; S_n\ \texttt{end})$$

If-then-else. We have two options in the `if-then-else` statement

- the current process is i, the boolean expression eb is true and the values of the variables does not change followed by the execution of S_1, or
- the current process is i, the boolean expression eb is false and the values of the variables does not change followed by the execution of S_2

Notice that we use the $\frown$ operator instead of the implication ($\Rightarrow$) because we allow the interleaving between the evaluation of the condition and the execution of the body of each branch of the conditional expression.

$$\mathcal{M}(\texttt{if}\ (eb)\, S_1\ \texttt{else}\ S_2) \stackrel{\text{def}}{=} (\mathcal{M}(eb) \frown \mathcal{M}(S_1)) \vee (\mathcal{M}(\neg eb) \frown \mathcal{M}(S_2))$$

In particular, the semantics of the `if-then` statement is defined as

$$\mathcal{M}(\texttt{if}\ (eb)\, S) \stackrel{\text{def}}{=} (\mathcal{M}(eb) \frown \mathcal{M}(S_1)) \vee \mathcal{M}(\neg eb)$$

Case. The `case` statement is just a resumed way of write nested `if-then-else` statements, but its semantics is not expressed here in terms of the simple conditional statement. If one of the e_i expressions is true then S_i is executed, otherwise the S_{n+1} statements take place.

$$\begin{aligned}&\mathcal{M}(\texttt{case}\ (e)\, e_1 : S_1; \ldots; e_n : S_n; \texttt{default} : S_{n+1}\ \texttt{endcase}) \stackrel{\text{def}}{=}\\ &\qquad ((\exists j\ .\ 1 \leq j \leq n\ .\ \mathcal{M}(e_j = \mathbf{b}.e) \wedge \mathbf{e}.\partial = i \wedge \mathit{unchange}_{\mathsf{Var}}) \frown \mathcal{M}(S_j)) \vee\\ &\qquad ((\forall j\ .\ 1 \leq j \leq n\ .\ \mathcal{M}(e_j \neq \mathbf{b}.e) \wedge \mathbf{e}.\partial = i \wedge \mathit{unchange}_{\mathsf{Var}}) \frown \mathcal{M}(S_{n+1}))\end{aligned}$$

While. The intuitive meaning is the usual one: if the boolean expression eb is true then the body of the `while` is executed. The semantics of the `while` statement is expressed using the *greatest fixed point operator* $\nu X.\phi$ in such a way that we can capture both the finite and the infinite behaviour. We use the expression $\nu X.((\phi_1 \frown X) \vee \phi_2)$ in the following way:

$$\mathcal{M}(\texttt{while}\ (eb)\, S) \stackrel{\text{def}}{=} \nu X.((\mathcal{M}(eb) \frown \mathcal{M}(S)) \frown X) \vee \mathcal{M}(\neg eb))$$

Note: Since the use of fixed point operator is sometimes cumbersome we explain here the intuition about the one used in the semantics of the loop statement. The semantics of the fixed point operator is (see Theorem 8.14)

$$\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}\left(\nu X.((\phi_1; X) \vee \phi_2)\right) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}\left(\mu X.((\phi_1; X) \vee \phi_2)\right) \cup E$$

where the $\mu X.((\phi_1 \frown X) \vee \phi_2)$ is the least fixed point and represent a finite sequence of intervals (say n), each formalising one iteration, where

- ϕ_1 is valid in the $n-1$ first and ϕ_2 is valid in the last one
- ϕ_1 is valid in all the intervals and the last one is infinite

The intuition of the E part is the following: there is an infinite sequence of intervals where ϕ_1 is valid. Clearly with this definition we can capture the following three different behaviours of loop including finite and infinite computations (see also Wang et al in [21, Section 11.2, page 77]):

1. The statement terminates within bounded time interval after finitely many iterations;
2. There are finitely many iterations but the last one does not terminate
3. There are infinitely many iterations and each of them takes finitely long time. Three kinds of behaviours may meet this situation:
   (a) The whole execution time is not bounded. It is generally called *divergence*. In $WDCI$ we can say that the end point of the interval is (∞, ∞) or (∞, n) for some natural number n.
   (b) The statement repeats forever and no time is spent in any iteration after certain point. In this case the endpoint of the interval (represented in $WDCI$) is (r, ∞) for some real number r.
   (c) The last behaviour is the so called *zero behaviour* in which the statement repeats forever and although each iteration takes a positive amount of time, the total execution time is bounded.

The last case of the third behaviour is not possible in Verilog. See Section 8 in page 57 to have a more exact definition of the fixed point operators and the E part.

Again the $\frown$ operator allows the interleaving between the evaluation of the condition of the `while` and the execution of the body. The same happens in the other iteration commands.

For. The `for` statement has the following intuitive semantics: first e_1 is assigned to the variable v_1, afterwards, if eb is true then the body of the iteration statement is executed and the second assignment takes place. This happens until eb becomes false. Clearly this iteration command is a particular case of the `while` statement.

$$\mathcal{M}(\texttt{for}\ \ (v_1 = e_1; eb; v_2 = e_2)\ S) \stackrel{\text{def}}{=} \mathcal{M}(v_1 = e_1) ^\frown \mathcal{M}(\texttt{while}\ \ (eb)\ \texttt{begin}\ \ S; v_2 = e_2\ \ \texttt{end})$$

Repeat. The intuition is that S is executed n times. It is expressed as a particular case of the `for` statement, where the initial value is 1, the condition to finish is that the counter be greater than n and the counter is incremented by one.

$$\mathcal{M}(\texttt{repeat}\ \ (n)\ S) \stackrel{\text{def}}{=} \mathcal{M}(\texttt{for}\ \ (v_1 = 1; v_1 \leq n; v_1 = v_1 + 1)\ S)$$

Forever. The forever statement is just a reduced written form of `while (true)` S. Its semantics is given using the greatest fixed point operator, representing infinite loop.

$$\mathcal{M}(\texttt{forever}\ \ S) \stackrel{\text{def}}{=} \nu X.(\mathcal{M}(S) ^\frown X)$$

See Example 5.3.2-3.

Delay. In Verilog the delay statement is the only (!?) way of advancing the time. The process that executes this statement ($\#n$), say i, is blocked in the current time slot (expressed in μWDCI as $\mathsf{idle_i}$) and it will be activated again just in some micro-time after n time units. In general n can be equal to zero, but for simplicity in this subset of the language (and as a first approach) we do not allow the case $n = 0$.

$$\mathcal{M}(\#n) \stackrel{\text{def}}{=} \mathsf{idle_i} \wedge l \leq n \wedge (\mathsf{fin}^\uparrow \Rightarrow \ell = n)$$

The semantics express that the process is *idle* (i.e. is blocked), the sum of all the macro-time intervals is less or equal to n and if all the micro-time intervals are finite then l must be equal to n. At first this formula can appear strange but we need to define the semantics of the delay in this way because we are using *maximal semantics* and we can have infinite intervals (in the macro-time as well as in the micro-time). Having in mind that we can have an infinite loop while the delay is taking place, we have to express that l will be equal to n just in case when we do not have such infinite micro-time sub-interval. That is why we need the last conjunction.

Timing control. Intuititively, when a timing control statement $@(v)\ S$ is reached then the process must wait until a change in v be produced in which case the S statement can be executed. It is possible that the timing control statement will never be activated. This situation can happen if other process remains in an infinite loop without changing the value of the variable v and this is modelled in μWDCI with the first expression of the disjunction.

$$\begin{aligned}\mathcal{M}(@(v)\ S) \stackrel{\text{def}}{=}\ & (\mathsf{idle_i} \wedge \Box(\mathbf{e}.v = \mathbf{b}.v)) \vee \\ & (\mathsf{idle_i} \wedge (\Box(\mathbf{e}.v = \mathbf{b}.v) \frown (\mathsf{dint} \wedge \mathbf{e}.v \neq \mathbf{b}.v) \frown \ell = o) \frown \mathcal{M}(S)))\end{aligned}$$

The $@(posedge\ v)\ S$ timing control statement is similar to the above but the change of v must be a "positive" one, i.e. the new value of v must be greater than the old one.

$$\begin{aligned}\mathcal{M}(@(posedge\ v)\ S) \stackrel{\text{def}}{=}\ & (\mathsf{idle_i} \wedge \Box(\mathbf{e}.v = \mathbf{b}.v)) \vee \\ & (\mathsf{idle_i} \wedge (\Box(\mathbf{e}.v = \mathbf{b}.v) \frown (\mathsf{dint} \wedge \mathbf{e}.v < \mathbf{b}.v) \frown \ell = o) \frown \mathcal{M}(S)))\end{aligned}$$

The $@(negedge\ v)\ S$ is similar to the $@(posedge\ v)\ S$ but with "negative" changes.

$$\begin{aligned}\mathcal{M}(@(negedge\ v)\ S) \stackrel{\text{def}}{=}\ & (\mathsf{idle_i} \wedge \Box(\mathbf{e}.v = \mathbf{b}.v)) \vee \\ & (\mathsf{idle_i} \wedge (\Box(\mathbf{e}.v = \mathbf{b}.v) \frown (\mathsf{dint} \wedge \mathbf{e}.v > \mathbf{b}.v) \frown \ell = o) \frown \mathcal{M}(S)))\end{aligned}$$

In the case of $@(v_1 \texttt{or}\ \ldots \texttt{or}\ v_n)\ S$ the process will continue with S if at least one of the v_i has its value changed.

$$\begin{aligned}\mathcal{M}(@(v_1 \texttt{or}\ \ldots \texttt{or}\ v_n)\ S) \stackrel{\text{def}}{=}\ & (\mathsf{idle_i} \wedge \textstyle\bigwedge_{i=1}^{n}(\Box(\mathbf{e}.v_i = \mathbf{b}.v_i))) \vee \\ & (\mathsf{idle_i} \wedge (\textstyle\bigwedge_{i=1}^{n}(\Box(\mathbf{e}.v_i = \mathbf{b}.v_i)) \frown (\mathsf{dint} \wedge \textstyle\bigvee_{i=1}^{n}(\mathbf{e}.v_i \neq \mathbf{b}.v_i) \frown \ell = o) \frown \mathcal{M}(S)))\end{aligned}$$

Remember that $\mathbf{b}.v$ and $\mathbf{e}.v$ are the initial and final value, respectively, of the variable v on the interval.

Initial. The semantics of `initial` S is

$$\mathcal{M}(\texttt{initial}\ S) \stackrel{\text{def}}{=} \mathcal{M}(S)$$

Parallelism. In V^- we have one main module that has many blocks or processes running in parallel (see Section 4, page 10). We model the parallelism operator of V^- using the conjunction ($\wedge$) in μWDCI. If we just interpret $\mathcal{M}(P_1 \parallel P_2)$ as $\mathcal{M}(P_1) \wedge \mathcal{M}(P_2)$ we can have some inconsistencies as we can see in the following example.

Let be the following parallel program:

$$\begin{array}{ccc} P_1 & & P_2 \\ x = 0 & \parallel & x = 5 \end{array}$$

The intuition said that we should have $x = 0$ or $x = 5$ as output, depending of which process is chosen first, but applying the interpretation above we have:

$$\begin{aligned} \mathcal{M}(x = 0 \parallel x = 5) = \quad & (l = 0 \wedge (\mathsf{idle}_1 \frown (\mathsf{dint} \wedge \mathbf{e}.x = 0 \wedge \mathit{unchange}_{\mathsf{Var}-\{x\}} \wedge \mathbf{e}.\partial = 1))) \wedge \\ & (l = 0 \wedge (\mathsf{idle}_2 \frown (\mathsf{dint} \wedge \mathbf{e}.x = 5 \wedge \mathit{unchange}_{\mathsf{Var}-\{x\}} \wedge \mathbf{e}.\partial = 2))) \end{aligned}$$

that means we have a sub-interval where x=0 and x=5! In order to avoid such inconsistence we have to guarantee that when one process is running the others are waiting ($\mathsf{idle_i}$), and the following seems to be a good definition of the semantics of the parallel composition:

$$\mathcal{M}(P_1 \parallel \ldots \parallel P_n) \stackrel{\text{def}}{=} \bigvee_{i=1}^{n} ((\mathcal{M}(P_1) \frown \mathsf{idle}_1) \wedge \ldots \wedge \mathcal{M}(P_i) \wedge \ldots \wedge (\mathcal{M}(P_n) \frown \mathsf{idle_n})) \tag{1}$$

Unfortunately, this definition does not work in the general case when at least one of the processes has the form `assign` $w = e$ (continuous assignment, see below), as shown in the following example.

Let P be the program

$$\begin{array}{l|l} P_1 & P_2 \\ \texttt{assign } x = y + 1 & y = 1; \\ & y = 2 \end{array}$$

With the definition (1) we would obtain that

$$\begin{aligned} \mathcal{M}(P_1 \parallel P_2) \stackrel{\text{def}}{=} \quad & (\mathcal{M}(\texttt{assign } x = y + 1) \frown \mathsf{idle}_1) \wedge \mathcal{M}(y = 1; y = 2) \vee \\ & \mathcal{M}(\texttt{assign } x = y + 1) \wedge (\mathcal{M}(y = 1; y = 2) \frown \mathsf{idle}_2) \end{aligned}$$

that is,

$$\mathcal{M}(P_1 \parallel P_2) \stackrel{\text{def}}{=} \begin{array}{l} (\lceil\lceil x = y+1 \rceil\rceil^+ {}^\frown \mathsf{idle}_1) \wedge \mathcal{M}(y=1; y=2) \vee \\ \lceil\lceil x = y+1 \rceil\rceil^+ \wedge (\mathcal{M}(y=1;y=2) {}^\frown \mathsf{idle}_2) \end{array}$$

The intension of the semantics of the continuous assignment `assign` $x = y+1$ is that the "equation" $x = y+1$ should be valid always (in the whole interval) and that is why the semantics of this statement was written using the "almost everywhere inside" construction. In the example we can see that this is not the case in the first disjunct $((\lceil\lceil x = y+1 \rceil\rceil^+ {}^\frown \mathsf{idle}_1) \wedge \mathcal{M}(y=1;y=2))$ where $\lceil\lceil x = y+1 \rceil\rceil^+$ is valid just in the first part of the interval (computation) and even though this is a valid formula it does not represent the desired behaviour. More complicated examples can be thought using the `delay` statement. Hence, the semantics of the parallel composition can be written as

$$\begin{array}{l} \mathcal{M}(P_1 \parallel \ldots \parallel P_j \parallel P_{j+1} \parallel \ldots \parallel P_n) \stackrel{\text{def}}{=} \\ \quad \bigvee_{i=1}^{j}((\mathcal{M}(P_1) {}^\frown \mathsf{idle}_1) \wedge \ldots \wedge \mathcal{M}(P_i) \wedge \ldots \wedge (\mathcal{M}(P_j) {}^\frown \mathsf{idle}_\mathsf{j}) \wedge \mathcal{M}(P_{j+1}) \wedge \ldots \wedge \mathcal{M}(P_n)) \end{array}$$

where $P_{j+1}, \ldots, P_n$ are processes of the form `assign` $w = e$ and $P_1, \ldots, P_j$ can be any other module statement (`initial` S, `always` S, $P_1 \parallel P_2$).

Note: in the examples that follow if no continuous assignment appears, then we will use the short form of the parallel composition semantics (1).

Remark: Suppose we have a continuous assignment (for example `assign` $w = 2$) running in parallel with another process (for example `initial` $x = 2$) then the semantics of the program is

$$\begin{array}{l} \mathcal{M}(\texttt{assign } w = 2) \parallel \texttt{initial } x = 2) \stackrel{\text{def}}{=} \\ \qquad \lceil\lceil w = 2 \rceil\rceil^+ \wedge \ell = 0 \wedge (\mathsf{idle}_2 {}^\frown (\mathsf{dint} \wedge \mathbf{e}.x = 2 \wedge \mathit{unchange}_{\mathsf{Var}-\{x\}} \wedge \mathbf{e}.\partial = 2)) \end{array}$$

We have a subinterval where $\mathit{unchange}_{\mathsf{Var}-\{x\}} \wedge \lceil\lceil w = 2 \rceil\rceil^+$ that seems a contradiction: we are saying that no variable change it value except x and at the same time we are claiming that the value of w is 2. This is no the case, since Var is the set of variables except the wires (see remark on page 33), hence in the example $\mathsf{Var} = \{x\}$ and we have that $\mathit{unchange}_{\mathsf{Var}-\{x\}} = \mathit{unchange}_{\emptyset}$.

Continuous assignment. In the left-hand-side of a continuous assignment we have a *wire* and not a variable. The wires (as seen in Example 4.1.1) are like channels that do not store values, just conducted them. The continuous assignment "`assign` $w = e$" continuously watches for changes to variables in its right hand side (variables that occur on the expression e) and whenever a change happens the right hand side is re-evaluated and the result is immediately propagated to the left hand side (w). In WDC we can easily express this behaviour with $\lceil\lceil w = \mathcal{M}(e) \rceil\rceil^+$ which is read "almost everywhere inside the interval, $w = \mathcal{M}(e)$" (holding for point and non-point intervals).

$$\mathcal{M}(\texttt{assign}\ \ w = e) \stackrel{\text{def}}{=} \lceil\lceil w = \mathcal{M}(e) \rceil\rceil^+$$

Always. The always is like the forever statement.

$$\mathcal{M}(\texttt{always}\ \ S) \stackrel{\text{def}}{=} \nu X.(\mathcal{M}(S) \frown X)$$

See Example 5.3.2-3.

5.3.1 Some results about the semantics formulae

We prove here some results related to inf, $\mathcal{M}(\mathsf{false})$, $\mathcal{M}(\mathsf{true})$, $\mathcal{M}(\texttt{skip})$, etc., in the context of μWDCI, as well as some results about finite and infinite sequence of time points (intervals). Let's first remember some useful definitions:

- $\mathcal{M}(\texttt{skip}) \stackrel{\text{def}}{=} \ell = 0 \wedge (\mathsf{idle_i} \frown (\mathsf{dint} \wedge \mathit{unchange}_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i))$
- $\mathcal{M}(\mathsf{false}) \stackrel{\text{def}}{=} \ell = 0 \wedge (\mathsf{idle_i} \frown \mathsf{false})$
- $\mathcal{M}(\mathsf{true}) \stackrel{\text{def}}{=} \ell = 0 \wedge (\mathsf{idle_i} \frown (\mathsf{true} \wedge \mathsf{dint} \wedge \mathit{unchange}_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i))$
- $\mathsf{fin} \stackrel{\text{def}}{=} \ell < \infty \wedge k < \infty$
- $\mathsf{inf} \stackrel{\text{def}}{=} \neg\mathsf{fin} \stackrel{\text{def}}{=} \ell = \infty \vee k = \infty$

The following Lemma will be used in the proof of Lemma 5.3-1.

Lemma 5.2 *If* $[b, e] \models \ell = 0$ *with* $b < \infty$ *and* $e = \infty$ *then* $[b, e] \models k = \infty$

Proof:

$[b,\infty] \models \mathsf{inf}$	(by hypothesis)
$[b,\infty] \models \ell = \infty \vee k = \infty$	(by definition of inf) (1)
$[b,\infty] \models (l = \infty \vee k = \infty) \wedge \ell = 0$	(by semantics of $\wedge$ from (1) and hypothesis)
$[b,\infty] \models (l = \infty \wedge \ell = 0) \vee (k = \infty) \wedge \ell = 0)$	(by distributivity)
$[b,\infty] \models k = \infty \wedge \ell = 0$	(because $(l = \infty \wedge \ell = 0) \Leftrightarrow \mathsf{false}$)
$[b,\infty] \models k = \infty$	(by semantics of $\wedge$) $\Box$

The Lemma below shows the validity of some formulae.

Lemma 5.3 *The following formulae holds in μWDCI:*

1. $\models \mathcal{M}(\mathsf{false}) \Leftrightarrow (\mathsf{idle_i} \wedge k = \infty \wedge \ell = 0)$
2. $\models \mathcal{M}(\mathsf{false})^\frown \psi \Leftrightarrow \mathcal{M}(\mathsf{false})$
3. $\models \mathcal{M}(\mathsf{false}) \Rightarrow \mathcal{M}(\mathsf{true})$
4. $\models (\mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\boldsymbol{skip})) \Leftrightarrow \mathcal{M}(\boldsymbol{skip})$
5. $\models (\phi^\frown \mathcal{M}(\mathsf{false})) \Rightarrow \mathsf{inf}$
6. *If* $\models \phi \Rightarrow \ell < \infty$ *then* $\models (\phi^\frown \mathcal{M}(\mathsf{false})) \Rightarrow k = \infty$
7. *If* $\models \phi \Rightarrow \ell = 0$ *then* $\models (\phi^\frown \mathcal{M}(\mathsf{false})) \Rightarrow (l = 0 \wedge k = \infty)$
8. $\models \mathsf{inf}^\frown \phi \Leftrightarrow \mathrm{inf}$
9. $\models (\phi \wedge \mathrm{inf})^\frown \psi \Leftrightarrow (\phi \wedge \mathsf{inf})$

Proof: We defined a model $\mathbf{M}$ as a tuple $((\mathcal{T},<),\mathcal{I},\mathcal{J},\mathcal{V},[b,e])$ (see Definition 2.2 in page 5). Because the above formulae are valid for all models we avoid to mention the tuple above, but we will mention the interval $([b,e])$ in some proofs where we need to talk about subintervals (as in the semantics of the "$\frown$" operator for example).

1.

	$[b,e] \models \mathcal{M}(\mathsf{false})$	
iff	$[b,e] \models \ell = 0 \wedge (\mathsf{idle_i} \frown \mathsf{false})$	(by definition)
iff	$[b,e] \models \ell = 0$ and	
	$[b,e] \models \mathsf{idle_i} \frown \mathsf{false}$	(by semantics of $\wedge$)
iff	$[b,e] \models \ell = 0$ and	
	$\exists m \,.\, b \le m \le e$.	
	$(([b,m] \models \mathsf{idle_i}$ and $[m,e] \models \mathsf{false})$ or	
	$(e = \infty$ and $[b,e] \models \mathsf{idle_i}))$	(by semantics of $\frown$)
iff	$[b,e] \models \ell = 0$ and	
	$(e = \infty$ and $[b,e] \models \mathsf{idle_i})$	(because $[m,e] \models \mathsf{false}$ is not possible)
iff	$[b,\infty] \models \mathsf{idle_i} \wedge \ell = 0$	(by semantics of $\wedge$) (1)
	$[b,\infty] \models k = \infty$	(by Lemma 5.2) (2)
	$[b,e] \models \mathsf{idle_i} \wedge \ell = 0 \wedge k = \infty$	(by (1) and (2) by semantics of $\wedge$)
Hence,	$\models \mathcal{M}(\mathsf{false}) \Leftrightarrow (\mathsf{idle_i} \wedge k = \infty \wedge l = 0)$	

2. By Lemma 5.3-9 we know that

$$\models (\phi \wedge \mathsf{inf}) \frown \psi \Leftrightarrow \phi \wedge \mathsf{inf}$$

Instantiating ϕ with $\mathsf{idle_i} \wedge \ell = 0$ we have

$$\models (\mathsf{idle_i} \wedge \ell = 0 \wedge \mathsf{inf}) \frown \psi \Leftrightarrow \mathsf{idle_i} \wedge \ell = 0 \wedge \mathsf{inf}$$

By definition of inf,

$$\models (\mathsf{idle_i} \wedge \ell = 0 \wedge (l = \infty \vee k = \infty)) \frown \psi \Leftrightarrow \mathsf{idle_i} \wedge \ell = 0 \wedge (l = \infty \vee k = \infty)$$

but

$$\models \ell = 0 \wedge (l = \infty \vee k = \infty) \Rightarrow \ell = 0 \wedge k = \infty$$

hence,

$$\models (\mathsf{idle_i} \wedge \ell = 0 \wedge k = \infty) \frown \psi \Leftrightarrow \mathsf{idle_i} \wedge \ell = 0 \wedge k = \infty$$

and by 1 we have that

$$\models \mathcal{M}(\mathsf{false}) \frown \psi \Leftrightarrow \mathcal{M}(\mathsf{false})$$

3.

$$
\begin{array}{lll}
 & \models \mathsf{false} \Rightarrow \mathsf{true} & \\
\text{iff} & \models \mathsf{false} \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i \Rightarrow & \\
 & \quad \mathsf{true} \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i & \text{(by monotonicity of } \wedge\text{)} \\
\text{iff} & \models \mathsf{idle_i}{}^\frown(\mathsf{false} \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i) \Rightarrow & \\
 & \quad \mathsf{idle_i}{}^\frown(\mathsf{true} \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i) & \text{(by monotonicity of } {}^\frown\text{)} \\
\text{iff} & \models \ell = 0 \wedge (\mathsf{idle_i}{}^\frown(\mathsf{false} \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i)) \Rightarrow & \\
 & \quad \ell = 0 \wedge (\mathsf{idle_i}{}^\frown(\mathsf{true} \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i)) & \text{(by monotonicity of } \wedge\text{)} \\
\text{iff} & \mathcal{M}(\mathsf{false}) \Rightarrow \mathcal{M}(\mathsf{true}) & \text{(by definition of M}(\mathsf{false}) \\
 & & \text{ and M}(\mathsf{true})\text{)}
\end{array}
$$

4. We will show first that $\models \mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\mathtt{skip}) \Rightarrow \mathcal{M}(\mathtt{skip})$ and then that $\models \mathcal{M}(\mathtt{skip}) \Rightarrow \mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\mathtt{skip})$.
$\Rightarrow$)

$$
\begin{array}{lll}
 & \models \mathcal{M}(\mathsf{false}) \Rightarrow \mathcal{M}(\mathsf{true}) & \text{(by 5.3-3)} \\
\text{implies} & \models \mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\mathsf{true}) \Rightarrow \mathcal{M}(\mathsf{true}) \vee \mathcal{M}(\mathsf{true}) & \text{(by monotonicity of } \vee\text{)} \\
\text{iff} & \models \mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\mathsf{true}) \Rightarrow \mathcal{M}(\mathsf{true}) & \text{(by idempotence of } \vee\text{)} \\
\text{iff} & \models \mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\mathtt{skip}) \Rightarrow \mathcal{M}(\mathtt{skip}) & \text{(because M}(\mathtt{skip}) = \text{M}(\mathsf{true})\text{)}
\end{array}
$$

$\Leftarrow$)

$$
\begin{array}{lll}
 & \models \mathcal{M}(\mathtt{skip}) \Rightarrow \mathcal{M}(\mathtt{skip}) & \\
\text{implies} & \models \mathcal{M}(\mathtt{skip}) \Rightarrow \mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\mathtt{skip}) & \text{(FOL property)}
\end{array}
$$

5.

$$
\begin{array}{lll}
 & \models \mathcal{M}(\mathsf{false}) \Leftrightarrow \mathsf{idle_i} \wedge k = \infty \wedge \ell = 0 & \text{(by 5.3-1)} \\
\text{implies} & \models \mathcal{M}(\mathsf{false}) \Rightarrow k = \infty \wedge \ell = 0 & \text{(FOL property) (1)} \\
 & \models k = \infty \wedge \ell = 0 \Rightarrow \mathsf{inf} & \text{(by definition of } \mathsf{inf}\text{) (2)} \\
 & \models \mathcal{M}(\mathsf{false}) \Rightarrow \mathsf{inf} & \text{(by transitivity from (1) and (2))} \\
\text{implies} & \models \phi{}^\frown\mathcal{M}(\mathsf{false}) \Rightarrow \phi{}^\frown\mathsf{inf} & \text{(by monotonicity of } {}^\frown\text{) (3)} \\
 & \models \phi{}^\frown\mathsf{inf} \Rightarrow \mathsf{inf} & \text{(4)} \\
 & \models \phi{}^\frown\mathcal{M}(\mathsf{false}) \Rightarrow \mathsf{inf} & \text{(by transitivity from (3) and (4))}
\end{array}
$$

6. Suppose that $\models (\phi{}^\frown\mathcal{M}(\mathsf{false})) \Rightarrow k = \infty$ is not valid. By 5.3-5 we know that

$$\models \phi{}^\frown\mathcal{M}(\mathsf{false}) \Rightarrow \mathsf{inf}$$

and by definition of inf we have that

$$\models \phi \frown \mathcal{M}(\mathsf{false}) \Rightarrow (k = \infty \vee \ell = \infty)$$

Because of our assumption we have that

$$\models \mathsf{inf} \Rightarrow \ell = \infty$$

and by transitivity,

$$\models \phi \frown \mathcal{M}(\mathsf{false}) \Rightarrow \ell = \infty$$

which, by definition of $\mathcal{M}(\mathsf{false})$ is the same as

$$\models \phi \frown (\mathsf{idle_i} \wedge \ell = 0 \wedge k = \infty) \Rightarrow \ell = \infty$$

in which case we have that

$$\models \phi \Rightarrow \ell = \infty$$

that contradicts the hypothesis $\models \phi \Rightarrow \ell < \infty$, so we must have that $\models (\phi \frown \mathcal{M}(\mathsf{false})) \Rightarrow k = \infty$, and we finished the proof.

7.

	$\models \mathsf{idle_i} \wedge \ell = 0 \wedge k = \infty \Rightarrow \ell = 0$	(FOL property)
implies	$\models \mathcal{M}(\mathsf{false}) \Rightarrow \ell = 0$	(definition of $\mathcal{M}(\mathsf{false})$) (1)
	$\models \phi \Rightarrow \ell = 0$	(by hypothesis) (2)
	$\models \phi \frown \mathcal{M}(\mathsf{false}) \Rightarrow \ell = 0 \frown \ell = 0$	(IL property from (1) and (2)) (3)
	$\models \ell = 0 \frown \ell = 0 \Rightarrow \ell = 0$	(by axiom IL-7 of Intervalar Logic) (4)
	$\models \phi \frown \mathcal{M}(\mathsf{false}) \Rightarrow \ell = 0$	(by transitivity from (3) and (4)) (5)
	$\models \phi \frown \mathcal{M}(\mathsf{false}) \Rightarrow k = \infty$	(by 5.3-6) (6)
	$\models \phi \frown \mathcal{M}(\mathsf{false}) \Rightarrow \ell = 0 \wedge k = \infty$	(FOL property, from (5) and (6)) $\square$

8. We have to prove that

$$\models \mathsf{inf} \frown \phi \Rightarrow \mathsf{inf}$$

and

$$\models \mathsf{inf} \Rightarrow \mathsf{inf} \frown \phi$$

$\Rightarrow$**)** Suppose that it is not true that $\models \mathsf{inf} \frown \phi \Rightarrow \mathsf{inf}$, i.e. exists a model $\mathbf{M} = (\mathcal{I}, \mathcal{J}, \mathcal{V}, [b, e])$ such that

$$\underbrace{(\mathcal{I}, \mathcal{J}, \mathcal{V}, [b, e]) \models \mathsf{inf} \frown \phi}_{[1]} \quad \text{and} \quad \underbrace{(\mathcal{I}, \mathcal{J}, \mathcal{V}, [b, e]) \not\models \mathsf{inf}}_{[2]}$$

By semantics of the chop operator [1] is equivalent to

$$\begin{array}{ll} \exists m, b \leq m \leq e \,.\, ((m \neq \infty \wedge (\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,m]) \models \mathsf{inf} \wedge (\mathcal{I}, \mathcal{J}, \mathcal{V}, [m,e]) \models \phi) & \text{or} \\ \quad (e = \infty \wedge (\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \models \mathsf{inf})) & \end{array}$$

But by definition of inf just $e = \infty \wedge (\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \models \mathsf{inf}$ can be true, and this contradicts [2]. Hence, we have that

$$\models \mathsf{inf}^\frown \phi \Rightarrow \mathsf{inf}$$

⇐) Let $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e])$ be a model that satisfies inf, then by semantics of inf and of infinite intervals, we have that

$$e = \infty \wedge (\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \models \mathsf{inf}$$

that implies

$$\begin{array}{ll} (e = \infty \wedge (\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \models \mathsf{inf}) & \text{or} \\ \exists m, b \leq m \leq e \,.\, (m \neq \infty \wedge (\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,m]) \models \mathsf{inf} \wedge (\mathcal{I}, \mathcal{J}, \mathcal{V}, [m,e]) \models \phi) & \end{array}$$

That is the same as

$$\begin{array}{ll} \exists m, b \leq m \leq e \,.\, ((m \neq \infty \wedge (\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,m]) \models \mathsf{inf} \wedge (\mathcal{I}, \mathcal{J}, \mathcal{V}, [m,e]) \models \phi) & \text{or} \\ \quad (e = \infty \wedge (\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \models \mathsf{inf})) & \end{array}$$

But this is the semantics of the chop operator, so

$$(\mathcal{I}, \mathcal{J}, \mathcal{V}, [b,e]) \models \mathsf{inf}^\frown \phi$$

Hence

$$\models \mathsf{inf} \Rightarrow \mathsf{inf}^\frown \phi$$

9. We can prove that

$$\models (\phi \wedge \mathsf{inf})^\frown \psi \;\Leftrightarrow\; (\phi \wedge \mathsf{inf})$$

in a similar way as Lemma 5.3-8. □

The next Lemma will be used in Lemmas 5.5 and 5.6.

Lemma 5.4 *If $a_0, a_1, \ldots, a_i, \ldots, a_m = b$ is a (possible infinite) non-descending sequence of time points such that*

1. $a = a_0$ and $a_i \leq b$ for all $0 \leq i \leq m$ (b can be infinite)

2. $(\mathcal{I},\mathcal{J},\mathcal{V},[a_i,a_{i+1}]) \models \ell = 0, \forall i \,.\, 0 \le i < m$

Then $(\mathcal{I},\mathcal{J},\mathcal{V},[a,b]) \models \ell = 0$

Proof: In μWDCI the time points a_i have the form (r_i,n_i) and by the hypothesis 2 we have that

$$(\mathcal{I},\mathcal{J},\mathcal{V},[(r_i,n_i),(r_{i+1},n_{i+1})]) \models l = 0 \;\text{ iff }\; r_i = r_{i+1} = r \text{ for some real number } r$$

We have two cases:

1. The sequence of time points is finite
2. The sequence of time points is infinite

1. Let $a = (r,n_0)$ and $b = (r,n_h)$ for some $h \in \aleph$, so we have that

$$\begin{array}{ll} & 0 = 0 \\ \text{iff} & r - r = 0 \\ \text{iff} & \pi_1(r,n_h) - \pi_1(r,n_0) = 0 \\ \text{iff} & \mathcal{V}(l)[(r,n_0),(r,n_h)] = 0 \\ \text{iff} & \mathcal{V}(l)[(r,n_0),(r,n_h)] = \mathcal{V}(0)[(r,n_0),(r,n_h)] \\ \text{iff} & (\mathcal{I},\mathcal{J},\mathcal{V},[(r,n_0),(r,n_h)]) \models \ell = 0 \\ \text{iff} & (\mathcal{I},\mathcal{J},\mathcal{V},[a,b]) \models \ell = 0 \text{ qed.} \end{array}$$

2. In particular, the property holds when b is infinite, since by definition of time domain (see Definition 2.1) $lim(r_i,n_i) = (r_i,\infty) = (r,\infty) \in \mathcal{T}$, and for all $i \ge 0, \pi_1(r,\infty) - \pi_1(r,n_i) = r - r = 0$. □

The next two Lemmas will be used in some of the examples presented in Section 5.3.2.

Lemma 5.5 *If $a_0, a_1, \ldots, a_i, \ldots$ is an infinite non-descending sequence of time points such that*

1. $a = a_0$ *and* $a_i \le b$ *for all* $i \ge 0$
2. $(\mathcal{I},\mathcal{J},\mathcal{V},[a_i,a_{i+1}]) \models \ell = 0 \wedge k \ge 1, \forall i \ge 0$

Then $(\mathcal{I},\mathcal{J},\mathcal{V},[a,b]) \models \ell = 0 \wedge k = \infty$

Proof: Since $a_0, a_1, \ldots, a_i, \ldots$ is an infinite sequence of time points of the form $a_i = (r, n_i)$ for some $r \in \Re^+$ (by hypothesis 2), then we have an infinite sequence of different time points (by hypothesis 2, $k \geq 1$) $(r, n_0), (r, n_1), \ldots, (r, n_i), \ldots$ that can be seen as an infinite sequence of natural numbers $n_0, n_1, \ldots, n_i, \ldots$. Hence, $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models k = \infty$ and by Lemma 5.4 we know that $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models \ell = 0$ and by the semantics of the $\wedge$ operator $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models \ell = 0 \wedge k = \infty$ and we finish the proof. □

Lemma 5.6 *If $a = a_0, a_1, \ldots, a_m = b$ is a finite non-descending sequence of time points such that*

1. $\forall 0 \leq i < m \;.\; a_i < b, (\mathcal{I}, \mathcal{J}, \mathcal{V}, [a_{i-1}, a_i]) \models \ell = 0 \wedge k \geq 1$
2. $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a_{m-1}, a_m]) \models \ell = 0 \wedge k = \infty$

Then $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models \ell = 0 \wedge k = \infty$

Proof: By Lemma 5.4 we know that $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models \ell = 0$, so just remain to show that $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models k = \infty$. By hypothesis 1:

$$(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a_0, a_{m-1}]) \models k \geq 1$$

By hypothesis 2,

$$(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a_{m-1}, a_m]) \models k = \infty$$

and by semantics of the $\frown$ operator we have that

$$(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a_0, a_m]) \models k \geq 1 \frown k = \infty$$

So,

$$(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models k = \infty$$

From the last expression and the previous result $((\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models \ell = 0)$ we have, by semantics of the $\wedge$ operator, that $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models \ell = 0 \wedge k = \infty$. □

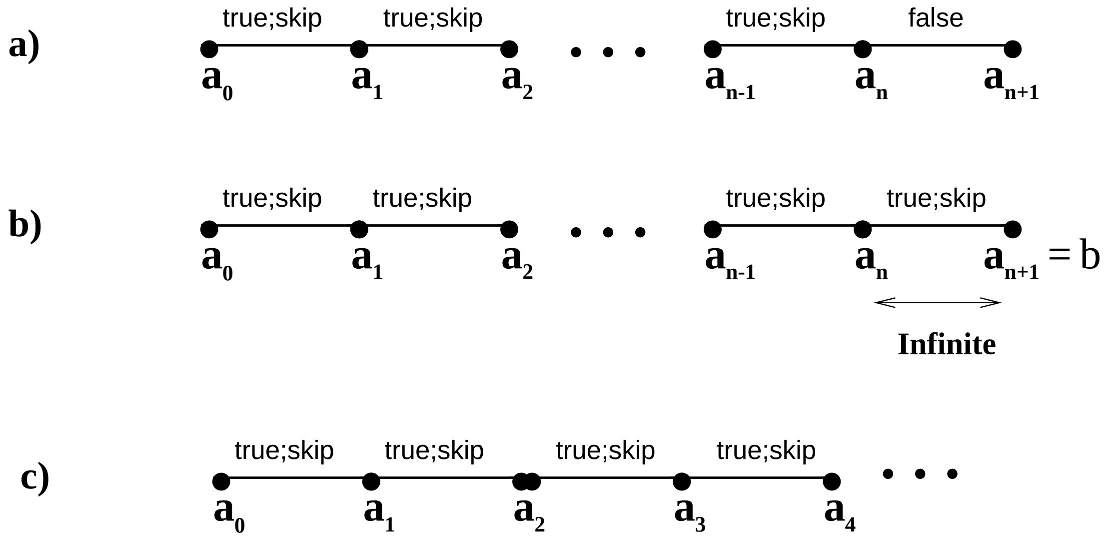

Figure 8: $\mathcal{M}(\texttt{while (true) skip})$.

5.3.2 Examples

1. $\mathcal{M}(\texttt{while (true) skip}) \Rightarrow \ell = 0 \wedge k = \infty$

   Proof: The semantics of a `while` statement (see Section 5.3, page 34) applied to this example gives the following expression

   $$\mathcal{M}(\texttt{while (true) skip}) \stackrel{\text{def}}{=} \nu X.((\mathcal{M}(\mathsf{true}) \frown \mathcal{M}(\texttt{skip})) \frown X) \vee \mathcal{M}(\mathsf{false}))$$

   and by Theorem 8.14 we know that

   $$\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}\Big(\nu X.((\phi_1; X) \vee \phi_2)\Big) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}\Big(\mu X.((\phi_1; X) \vee \phi_2)\Big) \cup E$$

   which behaviour can be described as in Figure 8 represented in a "flatted" time domain. By Lemma 5.3-1 we have that $\mathcal{M}(\mathsf{false}) \Rightarrow \ell = 0 \wedge k = \infty$ and because $\mathcal{M}(\mathsf{true}) \Rightarrow (\ell = 0 \wedge k \geq 1)$ and $\mathcal{M}(\texttt{skip}) \Rightarrow \ell = 0 \wedge k \geq 1$ we have that $\mathcal{M}(\mathsf{true}) \frown \mathcal{M}(\texttt{skip}) \Rightarrow \ell = 0 \wedge k \geq 1$. With these remarks we can obtain the following result (for each of the cases represented in Figure 8):

   (a) We have that $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a_0, a_{n-1}]) \models \phi \Rightarrow \ell = 0 \wedge k \geq 1$, where $\phi = \mathcal{M}(\mathsf{true}) \frown \mathcal{M}(\texttt{skip}) \frown \ldots \frown \mathcal{M}(\mathsf{true}) \frown \mathcal{M}(\texttt{skip})$, then by Lemma 5.3-7 we have that $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models \ell = 0 \wedge k = \infty$.

   (b) By Lemma 5.6 we have that $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models \ell = 0 \wedge k = \infty$.

   (c) By Lemma 5.5 we have that $(\mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b]) \models \ell = 0 \wedge k = \infty$.

   Hence, from the three cases above, $\mathcal{M}(\texttt{while (true) skip}) \Rightarrow \ell = 0 \wedge k = \infty$. qed.

2. $\mathcal{M}(\texttt{while}\ (\mathsf{false})\ P) = \mathcal{M}(\texttt{skip})$

   Proof: Applying again the semantics of a `while` statement (see Section 5.3, page 34) we obtain

   $$\mathcal{M}(\texttt{while}\ (\mathsf{false})\ P) \stackrel{\text{def}}{=} \nu X.((\mathcal{M}(\mathsf{false})^\frown\mathcal{M}(P))^\frown X) \vee \mathcal{M}(\mathsf{true}))$$

   By Lemma 5.3-2 we know that for a given formula ϕ, $\models \mathcal{M}(\mathsf{false})^\frown\phi \Leftrightarrow \mathcal{M}(\mathsf{false})$ holds. Instantiating ϕ with $\mathcal{M}(P)^\frown X$ we have

   $$\mathcal{M}(\mathsf{false})^\frown\mathcal{M}(P)^\frown X \Leftrightarrow \mathcal{M}(\mathsf{false})$$

   By monotonicity of the $\vee$ operator,

   $$(\mathcal{M}(\mathsf{false})^\frown\mathcal{M}(P)^\frown X) \vee \mathcal{M}(\mathsf{true}) \Leftrightarrow \mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\mathsf{true})$$

   and by monotonicity of the greatest fixed point operator (see Theorem 8.9) we obtain,

   $$\nu X.((\mathcal{M}(\mathsf{false})^\frown\mathcal{M}(P)^\frown X) \vee \mathcal{M}(\mathsf{true})) \Leftrightarrow \nu X.(\mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\mathsf{true}))$$

   which it is equivalent to

   $$\nu X.((\mathcal{M}(\mathsf{false})^\frown\mathcal{M}(P)^\frown X) \vee \mathcal{M}(\mathsf{true})) \Leftrightarrow \mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\mathsf{true})$$

   But the left-hand-side part of the double implication is the semantics of the `while` statement, so

   $$\mathcal{M}(\texttt{while}\ (\mathsf{false})\ P) \Leftrightarrow \mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\mathsf{true})$$

   and because the right-hand-side part is equivalent to $\mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\texttt{skip})$ (since $\mathcal{M}(\mathsf{true}) = \mathcal{M}(\texttt{skip})$) we have that

   $$\mathcal{M}(\texttt{while}\ (\mathsf{false})\ P) \Leftrightarrow \mathcal{M}(\mathsf{false}) \vee \mathcal{M}(\texttt{skip})$$

   and by Lemma 5.3-4 we finally obtain that

   $$\mathcal{M}(\texttt{while}\ (\mathsf{false})\ P) \Leftrightarrow \mathcal{M}(\texttt{skip})$$

   as stated.

3. $\mathcal{M}(\texttt{always}\ x = x + 1) \Rightarrow \ell = 0 \wedge k = \infty$

   Proof: The semantics of an `always` statement (see Section 5.3, page 40) applied to this example gives the following expression

   $$\mathcal{M}(\texttt{always}\ x = x + 1) \stackrel{\text{def}}{=} \nu X.(\mathcal{M}(x = x + 1)^\frown X)$$

and by definition of the semantics of $x = x + 1$,

$$\mathcal{M}(\texttt{always} \;\; x = x + 1) =$$
$$\nu X.(l = 0 \wedge (\mathsf{idle_i} ^\frown (\mathsf{dint} \wedge \mathbf{e}.x = \mathcal{M}(x + 1) \wedge unchange_{\mathsf{Var}-\{x\}} \wedge \mathbf{e}.\partial = i)) ^\frown X)$$

and because $l = 0 \wedge (\mathsf{idle_i} ^\frown (\mathsf{dint} \wedge \mathbf{e}.x = \mathcal{M}(x + 1) \wedge unchange_{\mathsf{Var}-\{x\}} \wedge \mathbf{e}.\partial = i)) \Rightarrow (l = 0 \wedge k \geq 1)$, by monotonicity of ν we have that

$$\nu X.(l = 0 \wedge (\mathsf{idle_i} ^\frown (\mathsf{dint} \wedge \mathbf{e}.x = \mathcal{M}(x + 1) \wedge unchange_{\mathsf{Var}-\{x\}} \wedge \mathbf{e}.\partial = i)) ^\frown X) \Rightarrow$$
$$\nu X.(l = 0 \wedge k \geq 1 ^\frown X)$$

Finally, by Lemma 5.5 we have that

$$\nu X.(l = 0 \wedge k \geq 1 ^\frown X) \Rightarrow \ell = 0 \wedge k = \infty$$

Hence we obtain the result claimed,

$$\mathcal{M}(\texttt{always} \;\; x = x + 1) \Rightarrow \ell = 0 \wedge k = \infty$$

4. $\mathcal{M}(\texttt{while} \;\; (\mathsf{true}) \; \texttt{skip} \parallel \#1) \Rightarrow \ell = 0 \wedge k = \infty \wedge \mathsf{idle_2}$

   Proof: Applying the semantics of the `parallel` statement (see Section 5.3, page 38) to this example we obtain the following expression

$$\mathcal{M}(\texttt{while} \;\; (\mathsf{true}) \; \texttt{skip} \parallel \#1) \stackrel{\text{def}}{=} \begin{array}{l} (\mathcal{M}(\texttt{while} \;\; (\mathsf{true}) \; \texttt{skip}) \wedge \mathcal{M}(\#1) ^\frown \mathsf{idle_2}) \vee \\ (\mathcal{M}(\texttt{while} \;\; (\mathsf{true}) \; \texttt{skip}) ^\frown \mathsf{idle_1} \wedge \mathcal{M}(\#1)) \end{array}$$

By the definition of the semantics of the `while` and `delay` statements the above is equal to

$$\nu X.(\mathcal{M}(\mathsf{true}) ^\frown \mathcal{M}(\texttt{skip}) ^\frown X \vee \mathcal{M}(\mathsf{false})) \wedge ((\mathsf{idle_2} \wedge l \leq 1 \wedge (fin^\uparrow \Rightarrow \ell = 1)) ^\frown \mathsf{idle_2})$$
$$\vee$$
$$(\nu X.(\mathcal{M}(\mathsf{true}) ^\frown \mathcal{M}(\texttt{skip}) ^\frown X \vee \mathcal{M}(\mathsf{false})) ^\frown \mathsf{idle_1}) \wedge (\mathsf{idle_2} \wedge l \leq 1 \wedge (fin^\uparrow \Rightarrow \ell = 1))$$

By Example 5.3.2-1 the above implies

$$(l = 0 \wedge k = \infty \wedge (\mathsf{idle_2} \wedge l \leq 1 \wedge (fin^\uparrow \Rightarrow \ell = 1) ^\frown \mathsf{idle_2}))$$
$$\vee$$
$$((l = 0 \wedge k = \infty) ^\frown \mathsf{idle_1}) \wedge (\mathsf{idle_2} \wedge l \leq 1 \wedge (fin^\uparrow \Rightarrow \ell = 1))$$

but

$$((\mathsf{idle_2} \wedge l \leq 1 \wedge (fin^\uparrow \Rightarrow \ell = 1)) ^\frown \mathsf{idle_2}) \Rightarrow \mathsf{idle_2}$$

and

$$(\mathsf{idle}_2 \wedge l \leq 1 \wedge (fin^{\uparrow} \Rightarrow \ell = 1)) \Rightarrow \mathsf{idle}_2$$

hence the above implies

$$(l = 0 \wedge k = \infty \wedge \mathsf{idle}_2) \vee ((l = 0 \wedge k = \infty {}^\frown \mathsf{idle}_1) \wedge \mathsf{idle}_2)$$

but $((l = 0 \wedge k = \infty) {}^\frown \mathsf{idle}_1) \Rightarrow ((\ell = 0 \wedge \mathsf{inf}) {}^\frown \mathsf{idle}_1)$ and by Lemma 5.3-9 implies $((\ell = 0 \wedge \mathsf{inf}) {}^\frown \mathsf{idle}_1) \Rightarrow \ell = 0 \wedge \mathsf{inf}$. By definition of inf we have $\ell = 0 \wedge \mathsf{inf} \Rightarrow \ell = 0 \wedge k = \infty$. We obtain then that

$$(\ell = 0 \wedge k = \infty \wedge \mathsf{idle}_2) \vee (\ell = 0 \wedge k = \infty \wedge \mathsf{idle}_2)$$

that gives

$$\ell = 0 \wedge k = \infty \wedge \mathsf{idle}_2$$

Hence,

$$\mathcal{M}(\texttt{while (true) skip} \parallel \#1) \Rightarrow \ell = 0 \wedge k = \infty \wedge \mathsf{idle}_2$$

6 *RDC* Vs. μWDCI as a semantical framework for Verilog

In this section we will compare our logical semantical approach to Verilog using μWDCI (see Section 5.3) with Pace's approach [12, Chapter 5]. The differences can be grouped in two classes:

- Differences of style: prefix-closed semantics vs. maximal semantics
- More substantial differences: scope, syntactical restrictions, etc.

In the following two subsections we will discuss briefly these two differences between Pace's approach and ours.

6.1 Maximal Vs. prefix-closed semantics

The differences pointed out in this section have to do with a style of presentation rather than with expressibility.

Pace gives semantics to Verilog using the hybrid duration calculus called Relational Duration Calculus, where the relational chop operator is crucial in his semantics since it allows the description of zero delayed assignment. He defines the meaning of primitive instructions in terms of continuation semantics and a set of variables it can write to, hence the semantics of a statement S in the language is $\mathcal{M}(S)_W(D)$: the semantics of S writing to variables W and followed by duration formula D. This continuation semantics is *prefix-closed* where partial (incomplete) executions are recording instead of complete ones. In this case, since infinite executions can be recorded by means of the set of its finite prefixes, there is no need to use infinite intervals in the logic.

In our approach we use a *maximal semantics*, i.e. we represent complete computations (the interval being considered is that of the whole computation) and we have infinite intervals in the logic. Of course, using the prefix closed approach implies in the introduction of some auxiliary variables and this lead to other differences discussed in the next section.

6.2 Other differences

We present here more substantial differences between Pace's and our approach.

Because of the prefix closed approach, Pace needed to introduce auxiliary variables to capture the termination of processes since the interpretation of a loop for instance is a set of intervals (each interval representing one iteration) instead of a single interval.

Another important feature is that Pace imposed some syntactical restrictions on the language [12, Section 5.2.1], because of "a good design practice". He also did not allow "super-steps" (the body of loops must take time) and "interference on shared variables" (global variables may not be read and written to simultaneously) among others. To prove some of the restrictions he had to define a predicate $\mathsf{dur}(S)$ to determine whether an instruction always takes time to execute or not (see [12, Section 5.5.2]).

The first restriction is due to the fact that *RDC* can just record the initial and final values of the variables at a given time point, so it is not possible to represent infinite loop where the body does not take time.

Because of the second syntactical restriction (no interference on shared variables) programs like

$$\begin{array}{l|l} P_1 & P_2 \\ x = x + 1; & y = x \\ x = x + 1 & \end{array}$$

are not allowed, but the following program would be

$$
\begin{array}{l|l}
P_1 & P_2 \\
x = x + 1; & \#1; \\
x = x + 1 & y = x
\end{array}
$$

The question that arise here is how to check this syntactical restriction, i.e. how can the compiler detect it?

Another interesting remark is that in the semantics using *RDC* the following is valid

$$x = x + 1; x = x + 1 \iff x = x + 2$$

and then we have that

$$x = x + 1; x = x + 1 \parallel y = x \iff x = x + 2 \parallel y = x$$

which is not true because the first implies that $y = 0 \vee y = 2$ and the second one that $y = 0 \vee y = 1 \vee y = 2$. This is why he need to impose such syntactical restriction about shared variables, otherwise the following property about parallelism (necessary for a compositional semantics)

$$P = Q \implies (P \parallel R) = (Q \parallel R)$$

would not be valid, as shown in the example above.

In our approach we can express not only the initial and final values of the variables but also the intermediate steps. Besides, we do not need auxiliary variables to asses the end of executions neither predicates for indicating if a process takes time or not. Moreover, we allow loops where the body does not take time as well as infinite computation in zero time (see the semantics of the `while` statement in page 5.3), easily represented because of our definition of the time domain framework (see Definition 2.1).

As a final remark we can claim that our approach is more clear and simpler than Pace's semantics (the relational chop operator is quite cumbersome [12, Section 3.4.3]). Moreover, our approach is more general because we do not impose any syntactical restriction, of course, we have to say that we do not include in V^- the non-blocking assignment that would complicate a bit the treatment even though the restrictions mentioned above have nothing to do with it.

7 The role of the $\mathsf{idle_i}$

In this section we will discuss briefly the role of the $\mathsf{idle_i}$ formula in the parallelism. Why the $\mathsf{idle_i}$ formula is needed in the atomic statements (skip and assignment) as well as in other statements? The answer is simple: to allow interleaving. We just present here few simple examples to show its use. Let P be the following program:

$$P: \ \texttt{skip}; \texttt{skip}$$

Suppose we had defined the semantics of the skip command in the following way:

$$\mathcal{M}(\texttt{skip}) = \ell = 0 \wedge \mathsf{dint} \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i$$

In this case the semantics of P would be

$$\begin{aligned}\mathcal{M}(\texttt{skip}; \texttt{skip}) = \quad & (l = 0 \wedge \mathsf{dint} \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i)^\frown \\ & (l = 0 \wedge \mathsf{dint} \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i)\end{aligned}$$

and no other process could be executed between the statements of P.

Since the use of the $\mathsf{idle_i}$ in the boolean expressions may not be so intuitive we show an example using the `if-then-else` statement. Let P be the following program, where the initial value of x is zero:

$$\begin{array}{lcl} P_1 & & P_2 \\ \texttt{if} \ (x \geq 0) \ x = 3 & \| & \texttt{if} \ (x = 0) \ y = 3 \end{array}$$

Remember that the semantics for the parallel composition and of the `if-then` statements are respectively:

$$\mathcal{M}(P_1 \parallel P_2) \stackrel{\text{def}}{=} (P_1 \wedge P_2{}^\frown \mathsf{idle}_2) \vee (P_1{}^\frown \mathsf{idle}_1 \wedge P_2)$$

$$\mathcal{M}(\texttt{if} \ (eb) \ S_1) \stackrel{\text{def}}{=} (\mathcal{M}(eb)^\frown \mathcal{M}(S_1)) \vee \mathcal{M}(\neg eb)$$

(we are using here the reduced form of the parallel composition semantics since our example does not have continuous assignment). Suppose that we let the semantics of the boolean expression under-specified, that means, that $\mathcal{M}(eb)$ is just true or false. In this case the semantics of the `if-then` would be (for simplicity we use just the part $\mathcal{M}(eb) \frown \mathcal{M}(S_1)$ because it is enough to show the problem):

$$\mathcal{M}(\texttt{if}\ \ (eb)\ S_1) \stackrel{\text{def}}{=} (eb \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = i) \frown \mathcal{M}(S_1)$$

In this case, applying the semantics of the parallel composition and of the `if-then` statements we would obtain:

$$\mathcal{M}(\texttt{if}\ \ (x \geq 0)\ x = 3 \parallel \texttt{if}\ \ (x = 0)\ y = 5) =$$

$$\begin{array}{l} \quad (((\mathcal{M}(x \geq 0) \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = 1) \frown \mathcal{M}(x = 3))\ \wedge \\ \quad ((\mathcal{M}(x = 0) \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = 2) \frown \mathcal{M}(y = 5) \frown \mathsf{idle_2})) \\ \vee \\ \quad (((\mathcal{M}(x = 0) \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = 2) \frown \mathcal{M}(y = 5)) \frown \mathsf{idle_2})\ \wedge \\ \quad ((\mathcal{M}(x \geq 0) \wedge unchange_{\mathsf{Var}} \wedge \mathbf{e}.\partial = 1) \frown \mathcal{M}(x = 3)) \end{array}$$

Let us see just the first part of the disjunction (the other case is symmetric). We have that the interval of the whole execution is divided (by the $\frown$ operator) in two subintervals, where in the first subinterval we have that $\mathbf{e}.\partial = 1$ and $\mathbf{e}.\partial = 2$ which is an inconsistence! We can solve this problem just using the $\mathsf{idle_i}$ in the semantics of the `if-then-else` statement (and in the `while` statement) or giving an explicit semantics to the evaluation of a boolean expression as we did (see Section 5.3 in page 30). It is easy to see that this problem is solved with our approach when using the `while` statement too.

Note: Under the closed system assumption it is easy to prove that taking off the $\mathsf{idle_i}$ from the atomic statements and introducing it in the parallel composition formula, we can obtain another compositional logical semantics to V^-.

7.1 Restricted interleaving semantics

Notice that we have presented a compositional semantics (both operational and abstract) allowing interleaving in the sequential composition without restriction (see also [7, Section 2.7] and [10, Section 5.4.2]). Some authors claim that the only statements that allows interleaving whenever a sequential block is being executed (`begin` $S_1; \ldots; S_n$ `end`) are the timing control. In an example ([6, Example 11.42, page 303]) Golze shows that the execution of two processes, each with `begin` $S_1; \ldots; S_n$ `end` statements, cannot be interleaved. The example presented by him is:

```
module event_control;

reg [4:0] N;

initial begin
  $display (``AAA'');
  $display (``BBB'');
end

initial
  for (N=0; N<=3; N=N+1)
    $display(N);

endmodule
```

The simulator can give the following output:

```
     0
     1
     2
     3
    AAA
    BBB
```

or even,

```
    AAA
    BBB
     0
     1
     2
     3
```

Golze said that the result

```
    AAA
     0
     1
     2
     3
    BBB
```

it is meaningful, and it is impossible to appear here, since both `initial` blocks do not contain time controls. In case we want to represent this reality we have to "play" with the $\mathsf{idle_i}$ preposition (see Section 7 above). Then, we have to change a bit our semantics: it is easy to see that if we do not use the $\mathsf{idle_i}$ preposition in the `skip` and procedural assignment and add it in the time control statements, we have the desire effect.

8 Some results about the Fixed Point formulae

We state here some important results about the fixed point operators defined in Section 3.2, where we interpreted a formula $\phi(X_1,\ldots,X_n)$ as an n-ary mapping on set of intervals. When defining $\mu X.\phi(X_1,\ldots,X_n,X)$ the intention is to interpret it as the operator that given sets of intervals $A_1,\ldots,A_n$ return the least fixed point of the mapping $\lambda A.\phi(A_1,\ldots,A_n,A)$ (or writing the function using the semantical function: $\lambda A.\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto A)}(\phi)$). We should have to prove that the above functional is monotonic (see [2, page 23], [21, page 15], [20, Section 4] and [18, Section 3.1]) otherwise we can not guarantee that the functional has a least fixed point. We will not prove here its monotonicity but we will impose a *syntatic monotonicity condition* stating that for the formula $\mu X.\phi$ to be well-formed any occurrence of X in ϕ must be within the scope of an even number of negation symbols, and then extending well-formedness to arbitrary formulas ψ by requiring all subformulas of ψ of the form $\mu X.\phi$ to be well-formed.

We prove first that defining the greatest fixed point as in Section 3.2 in page 8, the semantics is the intended one. Remember that $\nu X\cdot\phi$ was defined as:

$$\nu X\cdot\phi \stackrel{\text{def}}{=} \neg\mu Y\cdot\neg\phi[\neg Y/X]$$

The following Theorem shows that the semantics of the greatest fixed point is

$$\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X\cdot\phi) \stackrel{\text{def}}{=} \bigcup\{A \mid A\subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto A)}(\phi)\}$$

Proposition 8.1 *If* $\nu X\cdot\phi \stackrel{\text{def}}{=} \neg\mu Y\cdot\neg\phi[\neg Y/X]$ *then* $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X\cdot\phi) \stackrel{\text{def}}{=} \bigcup\{A \mid A\subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto A)}(\phi)\}$.

Proof:

$$\begin{array}{lll}
\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X \cdot \phi) & = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\neg \mu Y \cdot \neg \phi[\neg Y/X]) & \text{by definition of } \nu \\
& = \overline{\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu Y \cdot \neg \phi[\neg Y/X])} & \text{Semantics of } \neg \\
& = \overline{\bigcap\{A \mid \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y \mapsto A)}(\neg \phi[\neg Y/X]) \subseteq A\}} & \text{Semantics of } \mu \\
& = \bigcup\{\overline{A} \mid \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y \mapsto A)}(\neg \phi[\neg Y/X]) \subseteq A\} & \text{De Morgan law} \\
& = \bigcup\{\overline{A} \mid \overline{A} \subseteq \overline{\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y \mapsto A)}(\neg \phi[\neg Y/X])}\} & \text{set theory property} \\
& = \bigcup\{\overline{A} \mid \overline{A} \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y \mapsto A)}(\phi[\neg Y/X])\} & \text{Semantics of } \neg \\
& = \bigcup\{B \mid B \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y \mapsto \overline{B})}(\phi[\neg Y/X])\} & B = \overline{A} \\
& = \bigcup\{B \mid B \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X \mapsto B)}(\phi)\} & \text{by Lemma 8.2}
\end{array}$$

where we use the notation $\overline{A}$ instead of $2^{\mathbf{I}ntv} - A$ to represent the semantics of $\neg A$ (the complement in sets). □

The last step of the proof above is proved below in the following Lemma.

Lemma 8.2 *For any formula ϕ,*

$$\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y \mapsto \overline{B})}(\phi[\neg Y/X]) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X \mapsto B)}(\phi)$$

Proof: The proof is done by structural induction over the complexity of ϕ.

1. If $\phi = X$:

$$\begin{array}{lll}
\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y \mapsto \overline{B})}(\phi[\neg Y/X]) & = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y \mapsto \overline{B})}(\neg Y) & \text{by hypothesis} \\
& = \overline{\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y \mapsto \overline{B})}(Y)} & \text{semantics of } \neg \\
& = \overline{\{[c,d] \mid [c,d] \notin B\}} & \text{semantics of } Y \\
& = \{[c,d] \mid [c,d] \in B\} & \text{definition of complement} \\
& = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X \mapsto B)}(X) & \text{semantics of } X
\end{array}$$

2. If $\phi = Z$ and $Z \neq X$. Trivially true, since X is not in ϕ.

3. If $\phi = \phi_1 \wedge \phi_2$:

$$\begin{array}{lll}
 & \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y\mapsto\overline{B})}((\phi_1 \wedge \phi_2)[\neg Y/]) & \\
= & \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y\mapsto\overline{B})}(\phi_1[\neg Y/X]) \bigcap \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y\mapsto\overline{B})}(\phi_2[\neg Y/X]) & \text{semantics of } \phi_1 \wedge \phi_2 \\
= & \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto B)}(\phi_1) \bigcap \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto B)}(\phi_2) & \text{by Hypothesis of Induction} \\
= & \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto B)}(\phi_1 \wedge \phi_2) & \text{semantics of } \phi_1 \wedge \phi_2
\end{array}$$

4. If $\phi = \neg\phi_1$:

$$\begin{array}{lll}
\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y\mapsto\overline{B})}((\neg\phi_1)[\neg Y/X]) & = \overline{\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y\mapsto\overline{B})}(\phi_1[\neg Y/X])} & \text{semantics of } \neg \\
 & = \overline{\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto B)}(\phi_1)} & \text{by Hypothesis of Induction} \\
 & = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto B)}(\neg\phi_1) & \text{semantics of } \neg
\end{array}$$

5. If $\phi = \exists x \cdot \phi_1$:

$$\begin{array}{lll}
 & \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y\mapsto\overline{B})}((\exists x \cdot \phi_1)[\neg Y/X]) & \\
= & \bigcup_{a\in\mathsf{Values}} \mathcal{E}_{\mathcal{V}(x\mapsto a)}^{\mathcal{I},\mathcal{J}(Y\mapsto\overline{B})}(\phi_1[\neg Y/X]) & \text{semantics of } \exists \\
= & \bigcup_{a\in\mathsf{Values}} \mathcal{E}_{\mathcal{V}(x\mapsto a)}^{\mathcal{I},\mathcal{J}(X\mapsto B)}(\phi_1) & \text{by Hypothesis of Induction} \\
= & \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto B)}((\exists x \cdot \phi_1)) & \text{semantics of } \exists
\end{array}$$

6. If $\phi = \mu Z \cdot \phi_1$:

$$\begin{array}{lll}
 & \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y\mapsto\overline{B})}((\mu Z \cdot \phi_1[\neg Y/X]) & \\
= & \bigcap\{A \mid \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(Y,Z\mapsto\overline{B},A)}(\phi_1[\neg Y/X]) \subseteq A\} & \text{semantics of } \mu \\
= & \bigcap\{A \mid \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X,Z\mapsto B,A)}(\phi_1) \subseteq A\} & \text{by Hypothesis of Induction} \\
= & \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto B)}((\mu Z \cdot \phi_1) & \text{semantics of } \mu
\end{array}$$

   Notice that here $Z \neq X$ because Z is not free for X since Z is a "quantified" variable (it is bounded by μ). □

We show now the substitution Lemma for μWDCI. For two μWDCI formulas ϕ and ψ and a propositional letter X, we denote by $\phi[\psi/X]$ the formula obtained by replacing every *free occurrence* of X in ϕ by ψ. We require ψ to be free for X in ϕ (see any introduction book on logic to read more about substitution).

Lemma 8.3 *(Substitution Lemma) Let ϕ and ψ be two formulas and X a propositional letter such that ψ is free in ϕ for X. Then, for any propositional interpretation J,*

$$(\mathcal{I},\mathcal{J},\mathcal{V},[c,d]) \models \phi[\psi/X] \text{ iff } (\mathcal{I},\mathcal{J}',\mathcal{V},[c,d]) \models \phi(X)$$

where for all $[g,h], \mathcal{J}('X)[g,h] = 1$ iff $(\mathcal{I},\mathcal{J},\mathcal{V},[g,h]) \models \psi$ and for all $Y \neq X$. $\mathcal{J}('Y)[g,h] = \mathcal{J}(Y)[g,h]$.

Proof: The proof is done by structural induction over the complexity of the formulas. We will present here just the "chop" case.

If $\phi = \phi_1 \frown \phi_2$, then we have to prove that

$$(\mathcal{I},\mathcal{J},\mathcal{V},[c,d]) \models (\phi_1 \frown \phi_2)[\psi/X] \text{ iff } (\mathcal{I},\mathcal{J}',\mathcal{V},[c,d]) \models (\phi_1 \frown \phi_2)(X)$$

Then,

$$\begin{array}{lll}
& (\mathcal{I},\mathcal{J},\mathcal{V},[c,d]) \models (\phi_1 \frown \phi_2)[\psi/X] & \\
\text{iff} & (\mathcal{I},\mathcal{J},\mathcal{V},[c,d]) \models (\phi_1[\psi/X] \frown \phi_2[\psi/X] & \text{(by definition of substitution)} \\
\text{iff} & \exists m \in \mathcal{T}, c \leq m \leq d \,.\, (m \neq \infty \wedge (\mathcal{I},\mathcal{J},\mathcal{V},[c,m]) \models & \\
& \phi_1[\psi/X] \wedge (\mathcal{I},\mathcal{J},\mathcal{V},[m,d]) \models \phi_2[\psi/X]) & (1) \\
& \vee & \\
& (d = \infty \wedge (\mathcal{I},\mathcal{J},\mathcal{V},[c,d]) \models (\phi_1[\psi/X]) & \text{(semantics of } \frown\text{)} \quad (2)
\end{array}$$

We have to prove (1) and (2).

(1)

$$\begin{array}{lll}
& \exists m \in \mathcal{T}, c \leq m \leq d \,.\, (m \neq \infty \wedge (\mathcal{I},\mathcal{J},\mathcal{V},[c,m]) \models \phi_1[\psi/X] & \\
& \wedge(\mathcal{I},\mathcal{J},\mathcal{V},[m,d]) \models \phi_2[\psi/X]) & \\
\text{iff} & (\mathcal{I},\mathcal{J}'',\mathcal{V},[c,m]) \models \phi_1(X) \wedge (\mathcal{I},\mathcal{J}''',\mathcal{V},[m,d]) \models \phi_2(X)) & \text{(by H.I.)}
\end{array}$$

But by Hipothesis of Induction we know that

$\forall[g,h], \mathcal{J}''(X)[g,h] = 1$ iff $(\mathcal{I},\mathcal{J},\mathcal{V},[g,h]) \models \psi$ and

$\forall[g,h], \mathcal{J}'''(X)[g,h] = 1$ iff $(\mathcal{I},\mathcal{J},\mathcal{V},[g,h]) \models \psi$

iff $\mathcal{J}''(X)[g,h] = \mathcal{J}'''(X)[g,h]$

iff $\mathcal{J}'' = \mathcal{J}'''$ say $\mathcal{J}$'

iff $(\mathcal{I},\mathcal{J}',\mathcal{V},[c,m]) \models \phi_1(X) \wedge (\mathcal{I},\mathcal{J}',\mathcal{V},[m,d]) \models \phi_2(X))$

iff $(\mathcal{I},\mathcal{J}',\mathcal{V},[c,d]) \models (\phi_1 \frown \phi_2)(X)$ (by H.I.)

(2)

$d = \infty \wedge (\mathcal{I},\mathcal{J},\mathcal{V},[c,d]) \models \phi_1[\psi/X]$

iff $(\mathcal{I},\mathcal{J}',\mathcal{V},[c,d]) \models \phi_1(X)$ by Hypothesis of Induction

iff $(\mathcal{I},\mathcal{J}',\mathcal{V},[c,d]) \models \phi_1(X) \frown \phi_2(X)$ because $d = \infty$

iff $(\mathcal{I},\mathcal{J}',\mathcal{V},[c,d]) \models (\phi_1 \frown \phi_2)(X)$ □

Corollary 8.4 *If μWDCI-formula ϕ does not contain any free propositional variable symbols, then for any two $\mathcal{J}$-valuations $\mathcal{J}_1$ and $\mathcal{J}_2$,*

$$\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}_1}(\phi) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}_2}(\phi)$$

□

Corollary 8.5 *For any positive μWDCI-formula ϕ, any propositional variable symbol X,*

(1) if X doesn't appear in any scope of other propositional variable symbols in ϕ, then

$$\models \phi(\mu X.\phi) \Leftrightarrow \mu X.\phi$$

(2) if X is not a free propositional variable symbol of ϕ, then $\models \phi(\mu X.\phi) \Leftrightarrow \mu X.\phi$. □

We have similar results for $\nu X.\phi$.

Corollary 8.6 *For any μWDCI-formula $\nu X.\phi$,*

(1) If there is no propositional variable symbol with both free and bound occurrences in ϕ simultaneously, then $\models \phi(\nu X.\phi) \Leftrightarrow \nu X.\phi$.

(2) If ϕ has no free propositional variable symbol, then $\models \phi(\nu X.\phi) \Leftrightarrow \nu X.\phi$. □

Corollary 8.7 *Let ϕ be a μWDCI-formula and X a propositional variable symbol such that every occurrence of X in ϕ is positive. For any μWDCI-formula ψ, if*

(1) ψ *is free for* X *in* ϕ*;*

(2) $\models \phi[\psi/X] \Leftrightarrow \psi$.

then

$$\models \mu X.\phi \Rightarrow \psi \quad \textit{and} \quad \models \psi \Rightarrow \nu X.\phi$$

□

As a result we have that $\models \mu X.\phi \Rightarrow \nu X.\phi$. In particular, ψ is a fixed point of ϕ.

Theorem 8.8 *If $\overline{\phi}$ is a re-naming formula of some μWDCI-formula ϕ, then $\models \phi \Leftrightarrow \overline{\phi}$.* □

Theorem 8.9 *Let X be a propositional variable, ϕ_1 and ϕ_2 be two formulas such that X is positive in both. For any model* $\mathbf{M}$ *and valuation* $\mathcal{V}$*,*

$$\textit{if} \models \phi_1 \Rightarrow \phi_2,\ \textit{then} \quad \begin{array}{l} \models (\mu X.\phi_1) \Rightarrow (\mu X.\phi_2) \\ \models (\nu X.\phi_1) \Rightarrow (\nu X.\phi_2) \end{array}$$

Proof: For the validity of first formula, it is enough to prove $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi_1) \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi_2)$ for any $\mathcal{J}$-valuation $\mathcal{J}$.

By the assumption of this Theorem, $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2)$ for any $\mathcal{J}$-valuation $\mathcal{J}$.

Let $\mathcal{J}$ be a $\mathcal{J}$-valuation.

$$\begin{array}{c} \{\chi \mid \chi \in 2^{\mathbf{Intv}},\ \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi)}(\phi_2) \subseteq \chi\} \subseteq \{\chi \mid \chi \in 2^{Intv},\ \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi)}(\phi_1) \subseteq \chi\} \\ \bigcap\{\chi \mid \chi \in 2^{\mathbf{Intv}},\ \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi)}(\phi_2) \subseteq \chi\} \supseteq \bigcap\{\chi \mid \chi \in 2^{\mathbf{Intv}},\ \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi)}(\phi_1) \subseteq \chi\} \\ \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi_2) \supseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi_1) \end{array}$$

The second formula can be proved similarly. □

Theorem 8.10 *Let ϕ, ψ and β be μWDCI-formulas, X a propositional variable symbol. Both ψ and β are free for X in ϕ. If $\models \psi \Leftrightarrow \beta$, then $\models \phi[\psi/X] \Leftrightarrow \phi[\beta/X]$.* □

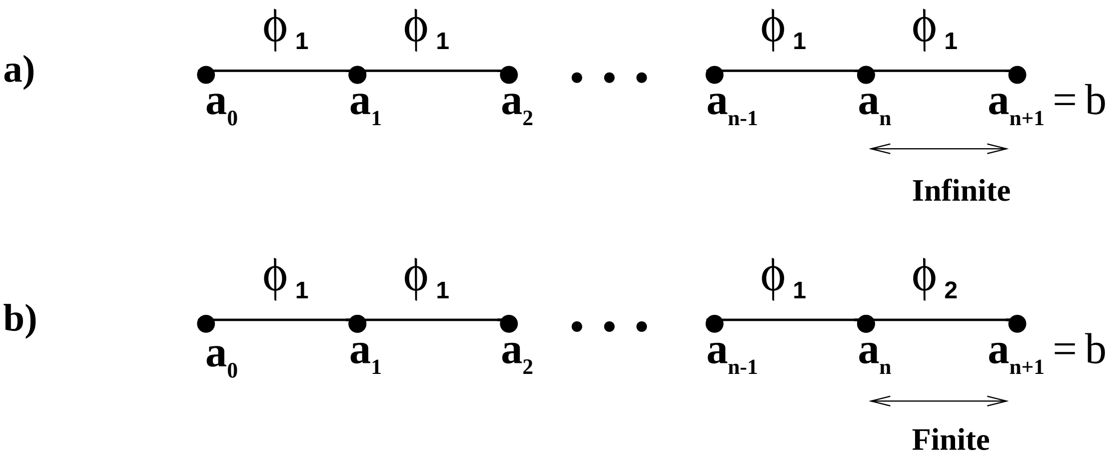

Figure 9: $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.(\phi_1 \frown X)) \vee \phi_2)$.

In the Theorems and Corollaries that follow, we will use $[b,e] \models \psi$ as a short for $(\mathcal{I},\mathcal{J},\mathcal{V},[b,e]) \models \psi$.

The following Theorem characterise the intervals that belongs to the fixed point formula $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.((\phi_1 \frown X)) \vee \phi_2)$. The intuitive meaning can be seen in Figure 9.

Theorem 8.11 *Let ϕ_1, ϕ_2 be two μWDCI-formulas. If both of ϕ_1 and ϕ_2 don't contain any free occurrence of the propositional variable symbol X, then for any model $\mathbf{M} = (\mathcal{I},\mathcal{J},\mathcal{V},[b,e])$, $\mathcal{J}$-valuation J and $\mathcal{V}$-valuation V, $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.(\phi_1 \frown X)) \vee \phi_2)$ iff the following condition $(*)$ is satisfied:*
there is a natural number n (may be 0) and a non-descending sequence of time points $a_0, a_1, \ldots, a_n, a_{n+1}$ such that

- *$a_0, a_1, \ldots, a_n \in \mathcal{T} - \{\infty\}$ and $a_{n+1} \in \mathcal{T}$;*
- *$a_0 = a$, $a_{n+1} = b$;* $\quad (*)$
- *$[a_i, a_{i+1}] \models \phi_1$ for all $i < n$;*
- *$[a_n, a_{n+1}] \models \phi_2$ or ($b = \infty$ and $[a_n, b] \models \phi_1$).*

Proof: Denote by χ_0 the set of intervals $[a,b]$ that satisfy the condition $(*)$ (the "if" condition), and denote $((\phi_1 \frown X) \vee \phi_2)$ by ϕ. It is sufficient to prove that $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi) = \chi_0$ (that is $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi) \subseteq \chi_0$ and $\chi_0 \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$) for any $\mathcal{J}$-valuation $\mathcal{J}$.

(1) To prove $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi) \subseteq \chi_0$, we need only to show $\chi_0 \in \{\chi \mid \chi \in 2^{\mathbf{I}ntv},\ \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X \mapsto \chi)}(\phi) \subseteq \chi\}$, that is, to show $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X \mapsto \chi_0)}(\phi) \subseteq \chi_0$.

Now suppose $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi)$. Because

$$\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi_1 \frown X) \cup \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi_2)$$

we have

$$[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi_1 \frown X) \ \text{ or } \ [a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi_2).$$

We will prove first that $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi_2)$ implies $[a,b] \in \chi_0$ and then that $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi_1 \frown X)$ implies $[a,b] \in \chi_0$

(1.1) If $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi_2)$, then, by Corollary 8.4, $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2)$ because ϕ_2 doesn't contain X, that is $[a,b] \models \phi_2$. By choosing n to be 0, we obtain that $[a,b] \in \chi_0$.

(1.2) If $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi_1 \frown X)$, because

$$\begin{aligned}
& \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi_1 \frown X) \\
= \ & \{[c,d] \mid \exists m \,.\, c \le m \le d \,.\, (m \ne \infty \wedge [c,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi_1) \wedge [m,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(X)) \vee \\
& \qquad\qquad (d = \infty \wedge [c,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi_1))\} \\
= \ & \{[c,d] \mid \exists m \,.\, c \le m \le d \,.\, (m \ne \infty \wedge [c,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \wedge [m,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(X))\} \bigcup \\
& \{[c,d] \mid d = \infty \wedge [c,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1)\}
\end{aligned}$$

it is enough to consider the following two cases:

(1.2.1) If $[a,b] \in \Big(\{[c,d] \mid \exists m \,.\, c \le m \le d \,.\, (m \ne \infty \wedge \underbrace{[c,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1)}_{[1]} \wedge \underbrace{[m,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(X))}_{[2]}\}\Big)$

and let $a = c = a_0, m = a_n, b = d = a_{n+1}$. Hence, by [1]: $[a_0,a_n] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1)$, so $[a_i,a_{i+1}] \models \phi_1 \forall i, 0 \le i \le n-1$ that implies $[a_0,a_n] \in \chi_0$. On the other hand, by [2] we have that $[a_n,a_{n+1}] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(X)$ iff $[a_n,a_{n+1}] \in \chi_0$ and then $[a,b] \in \chi_0$.

(1.2.2) If $[a,b] \in \{[c,\infty] \mid [c,\infty] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1)\}$ then let $a = c = a_0$ and $b = \infty = a_{n+1}$ and clearly $[a_i,a_{i+1}] \models \phi_1 \forall 0 \le i \le n-1$ and $(b = \infty,\ [a_n,b] \models \phi_1$. Hence $[a,b] \in \chi_0$.

Summing up above cases, we have $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi_0)}(\phi) \subseteq \chi_0$.

By definition we have that $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi) \subseteq \chi_0$ (qed).

(2) To prove $\chi_0 \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$ we start from investigating χ_0 and $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$.

(2.1) To describe χ_0, we define a sequence of μWDCI-formulas as follows:

Let $\psi_0 = \phi_2$;

$$\psi_n = \underbrace{\phi_1 \frown \phi_1 \frown \cdots \frown \phi_1}_{n \text{ numbers of } \phi_1} \frown \phi_2, \quad \text{for any natural number } n > 1.$$

It is not hard to see that $\chi_0 = \bigcup_{n \in \mathbf{N}} \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\psi_n)$.

(2.2) To discuss $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$, we define another sequence of μWDCI-formulas as follows:

Let $\phi^0 = \mu X.\phi$

$\phi^1 = \phi[\phi^0/X]$

$\ldots$

$\phi^{n+1} = \phi[\phi^n/X]$

$\ldots$

By Corollary 8.5-(1), we have $\models \phi^0 \Leftrightarrow \phi^1$.
By Theorem 8.10 , $\models \phi^1 \Leftrightarrow \phi^2$.
Using inductive method we can easily prove that $\models \phi^n \Leftrightarrow \phi^{n+1}$ for any $n \in \mathbf{N}$.

On the other hand,

$$\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi^1) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi[\phi^0/X]) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}((\phi_1 \frown \phi^0) \vee \phi_2) \supseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\psi_0).$$

$$\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi^2) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi[\phi^1/X]) \supseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi[\psi_0/X]) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}((\phi_1 \frown \psi_0) \vee \phi_2) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\psi_0 \vee \psi_1)$$

(by Theorem 8.10)

$\ldots$

$$\begin{aligned}
\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi^{n+1}) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi[\phi^n/X]) &\supseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}((\phi_1 \frown (\psi_0 \vee \cdots \vee \psi_{n-1})) \vee \phi_2) \\
&= \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(((\phi_1 \frown \psi_0) \vee \cdots \vee (\phi_1 \frown \psi_{n-1})) \vee \phi_2) \\
&= \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}((\psi_1 \vee \cdots \vee \psi_n) \vee \psi_0) \\
&= \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\psi_0 \vee \cdots \vee \psi_n)
\end{aligned}$$

$\ldots$

Hence $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi^0) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_{n+1}) \supseteq \bigcup_{n \in \mathbf{N}} \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\psi_0 \vee \cdots \vee \psi_n) = \bigcup_{n \in \mathbf{N}} \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\psi_n) = \chi_0$.

So, $\chi_0 \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$. (qed) $\square$

Corollary 8.12 *Let ϕ_1 be μWDCI-formula. If ϕ_1 doesn't contain any free occurrence of the propositional variable symbol X, then for any model $\mathbf{M} = ((\mathcal{T}, <), \mathcal{I}, \mathcal{J}, \mathcal{V}, [a,b])$, $[a,b] \models \mu X.(\phi_1 \frown X)$ iff the following condition can be satisfied:*

there is a natural number n (may be 0) and a non-descending sequence of time points $a_0, a_1, \ldots, a_n, a_{n+1}$ such that

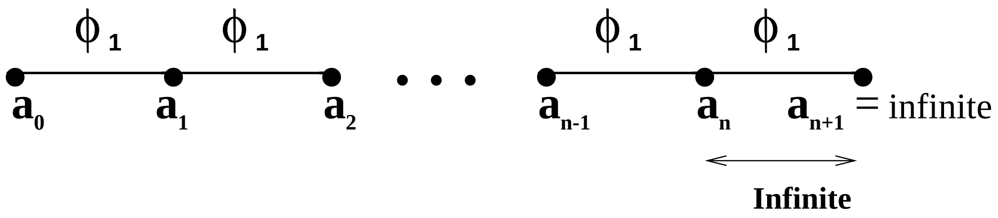

Figure 10: $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi \frown X)$

- $a_0, a_1, \ldots, a_n \in T - \{\infty\}$ *and* $a_{n+1} \in \mathcal{T}$*;*
- $a_0 = a,\ \ a_{n+1} = b = \infty$*;*
- $[a_i, a_{i+1}] \models \phi_1$ *for all* $i < n$*;*
- $[a_n, b] \models \phi_1$.

Proof: Let $\phi = (\phi_1 \frown X) \vee \mathsf{false}$. Then $\models \phi \Leftrightarrow (\phi_1 \frown X)$.

By Theorem 8.9, $\models \mu X.\phi \Leftrightarrow \mu X.(\phi_1 \frown X)$.

We can easily see that the Corollary is true by applying Theorem 8.11 to $\mu X.\phi$. □

See Figure 10 for the representation of the intervals of $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi \frown X)$.

Corollary 8.13 *Let* ϕ_1, ϕ_2 *be two* μ*WDCI-formulas. Both of* ϕ_1 *and* ϕ_2 *don't contain any free occurrence of the propositional variable symbol* X*. Denote* $(\phi_1 \frown X) \vee \phi_2$ *by* ϕ*. For any model* $\mathbf{M} = ((\mathcal{T}, <), \mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b])$*, if exists* $m \in T - \{\infty\}$ *such that*

- $a \le m \le b$*;*
- $[a, m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1)$ *and* $[m, b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$.

then $[a, b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$ □

Now we will investigate the greatest fixed-point operator of the formulas ϕ with the form $(\phi_1 \frown X) \vee \phi_2$. From Corollary 8.7 we know that $\models \mu X.\phi \Rightarrow \nu X.\phi$. What is the difference between $\nu X.\phi$ and $\mu X.\phi$? The following Theorem will answer this question.

Theorem 8.14 *Let* ϕ_1, ϕ_2 *be two* μ*WDCI-formulas. If both of* ϕ_1 *and* ϕ_2 *don't contain any free occurrence of the propositional variable symbol* X*, then for any model* $\mathbf{M} = ((\mathcal{T}, <), \mathcal{I}, \mathcal{J}, \mathcal{V}, [a, b])$*,*

$$\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}\Big(\nu X.((\phi_1 \frown X) \vee \phi_2)\Big) = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}\Big(\mu X.((\phi_1 \frown X) \vee \phi_2)\Big) \cup E$$

where, for each $[a,b] \in \mathbf{M}$, $[a,b] \in E$ *if only if*
there is an infinite non-descending sequence of time points $a_0, a_1, \ldots, a_n, \ldots$ $(n \geq 0)$ *such that:*

- $a_0 = a, \;\; a_n \leq b$ *for all* $n \geq 0$*;*
- $[a_n, a_{n+1}] \models \phi_1$ *for all* $n \geq 0$

Proof: Denote $((\phi_1 \frown X) \vee \phi_2)$ by ϕ, and $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi) \cup E$ by χ' (see Figure 9 for the representation of $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$ and Figure 11 for E). Hence, we have to prove that $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi) \subseteq \chi'$ and $\chi' \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$.

(1) For the direction $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi) \subseteq \chi'$, we divide the proof into two steps.

(1.1) We will show first that:
For any $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$, if $[a,b] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$, then there exist $m \in \mathcal{T} - \{\infty\}$ such that:

- $a \leq m \leq b$;
- $[a,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1)$;
- $[m,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$ but $[m,b] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$.

In fact, because $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$ is also a fixed point of $\lambda A.\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X \mapsto A)}(\phi)$, we have

$$\begin{aligned}
\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi) = \;& \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}\Big((\phi_1 \frown (\nu X.\phi)) \vee \phi_2\Big) \\
= \;& \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}\Big(\phi_1 \frown (\nu X.\phi)\Big) \;\bigcup\; \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2) \\
= \;& \{[c,d] \mid \exists c \leq m \leq d \;.\; (m \neq \infty \wedge [c,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \wedge [m,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)) \\
& \vee (d = \infty \wedge [c,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1))\} \;\cup\; \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2) \\
= \;& \{[c,d] \mid \exists c \leq m \leq d \;.\; (m \neq \infty \wedge [c,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \wedge [m,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi))\} \\
& \bigcup \{[c,d] \mid d = \infty \wedge [c,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1)\} \;\bigcup\; \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2)
\end{aligned}$$

So, for any $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$, $[a,b]$ must be in

$$\{[c,d] \mid \exists c \leq m \leq d \;.\; (m \neq \infty \wedge [c,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \wedge [m,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi))\}$$

or in

$$\{[c,d] \mid d = \infty \wedge [c,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1)\} \cup \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2)$$

But $\{[c,d] \mid d = \infty \wedge [c,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1)\} \cup \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2)$ is included in $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$. So, if $[a,b] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$, then $[a,b]$ must be in $\{[c,d] \mid \exists c \leq m \leq d\ .\ (m \neq \infty \wedge [c,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \wedge [m,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi))\}$. Hence there exists $m \in T - \{\infty\}$ such that:

- $a \leq m \leq b \wedge m \neq \infty$;
- $[a,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1)$;
- $[m,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$

If $[m,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$, then, by Corollary 8.13, $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$, that lead us to a contradiction. So $[m,b] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$, that is, m is the time point desired.

(1.2) Now we prove $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi) \subseteq \chi'$. it is sufficient to show that, for any $[a,b] \in \mathbf{M}$, if $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$ but $[a,b] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$, then $[a,b]$ must be in E. Let $[a,b]$ be any such an interval, we will construct an infinite non-descending sequence of time points $a_0, a_1, \ldots, a_n, \ldots$ $(n \geq 0)$ by induction as follows:

* $a_0 = a$;
* because $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$ and $[a,b] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$, by (1.1), there exists $m \in T - \{\infty\}$ such that:

  - $a \leq m \leq b$;
  - $[a,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1)$;
  - $[m,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$ but $[m,b] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$.

  Let $a_1 = m$.
* Provided $a_0 \leq \ldots \leq a_n$ have been defined with the following properties:

  - $a \leq a_i \leq b$ for all $i:\ 0 \leq i \leq n$;
  - $[a_i, a_{i+1}] \models \phi_1$ for all $0 \leq i < n$;
  - $[a_n,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$ but $[a_n,b] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$.

Now we will construct a_{n+1}. Because $[a_n,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$ and $[a_n,b] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$, by (1.1), there exist m such that

- $a_n \leq m \leq b$;
- $[a_n,m] \models \phi_1$;
- $[m,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$ and $[m,b] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$.

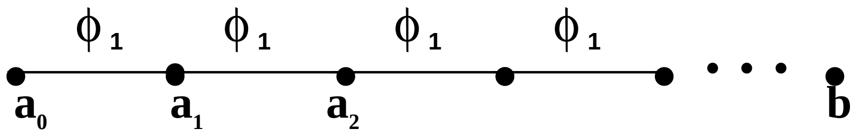

Figure 11: The representation of E.

Let $a_{n+1} = m$. Then

- $a_n \leq a_{n+1} \leq b$;
- $[a_n, a_{n+1}] \models \phi_1$;
- $[a_{n+1}, b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$ and $[a_{n+1}, b] \notin \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi)$.

We have finished the construction of a_n for $n \in \aleph$. With these a_n at hand, by the definition of E, we have that $[a, b] \in E$. (2) For the other direction $\chi' \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$, it is enough to prove $E \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$ (since by Corollary 8.7, $\mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\mu X.\phi) \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$). But we have that

$$\begin{array}{ll} & E \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto E)}(\phi) \\ \text{Implies} & E \in \{\chi \mid \chi \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi)}(\phi)\} \\ \text{Implies} & E \subseteq \bigcup\{\chi \mid \chi \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto\chi)}(\phi)\} \\ \text{Implies} & E \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi) \end{array}$$

Hence, we need only to prove that $E \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto E)}(\phi)$. But

$$\begin{array}{ll} \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto E)}(\phi) & = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto E)}((\phi_1 \frown X) \vee \phi_2) \\ & = \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto E)}(\phi_1 \frown X) \ \cup \ \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto E)}(\phi_2) \\ & = \{[c,d] \mid \exists c \leq m \leq d \ . \ (m \neq \infty \wedge [c,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto E)}(\phi_1) \wedge [m,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto E)}(X)) \\ & \qquad\qquad \vee \ (d = \infty \wedge [c,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto E)}(\phi_1))\} \\ & \ \cup \ \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}(X\mapsto E)}(\phi_2) \\ & = \{[c,d] \mid \exists c \leq m \leq d \ . \ (m \neq \infty \wedge [c,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \wedge [m,d] \in E) \ \vee \\ & \qquad (d = \infty \wedge [c,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1))\} \\ & \ \cup \ \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2) \end{array}$$

so, it is sufficient to show that $E \subseteq \{[c,d] \mid \exists c \leq m \leq d \ . \ (m \neq \infty \wedge [c,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \wedge [m,d] \in E) \vee (d = \infty \wedge [c,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1))\} \ \cup \ \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_2)$

For any $[a,b] \in \mathbf{M}$, if $[a,b] \in E$, then there is an infinite sequence of time points $a_0, a_1, \ldots, a_n, \ldots$ (with $a = a_0$)satisfying the conditions in the definition of E. Then

$$[a_0, a_1] \models \phi_1 \quad \text{and} \quad [a_1, b]) \in E$$

which is the same as

$$[a_0, a_1] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \quad \text{and} \quad [a_1, b]) \in E$$

Let $m = a_1$. Hence, $[a,b] \in \{[c,d] \mid \exists c \leq m \leq d \,.\, (m \neq \infty \wedge [c,m] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1) \wedge [m,d] \in E) \vee (d = \infty \wedge [c,d] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\phi_1))\}$, that is $[a,b] \in \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$. We have proved then that $E \subseteq \mathcal{E}_{\mathcal{V}}^{\mathcal{I},\mathcal{J}}(\nu X.\phi)$.
□

9 Discussion

This final section will be divided in two subsections: in the first we discuss the results of our work and in the second one we present some future works.

9.1 Results

We have given an operational and logical semantics to the V^- subset of Verilog. We have extended the Duration Calculus with Weakly Monotonic Time [16] with fixed point operators and infinite intervals obtaining μWDCI. Some results on fixed point operators at the semantical level were presented (Section 8).

WDC is useful as a logical framework for the semantics of languages that have zero-time computations. With our extension with infinite intervals and fixed point operators we have given a compositional semantics to V^- allowing infinite computations and in particular zero-time computations. The compositionality is due in part to our definition of the $\mathsf{idle_i}$ μWDCI-formula. It is possible to show that "playing" with the $\mathsf{idle_i}$, taking it off the atomic statements and introducing it in the parallel composition, we can obtain an alternative compositional semantics.

In particular the semantics of a V^- system was defined as the conjunction of three μWDCI formulas (see page 32): the semantics of the main program and two axioms, where the second axiom (about the changes on the variables and the execution of the processes) was established with the aim of extending our approach to hybrid systems (see the following section).

In Section 6 we have also compared our approach with that of Pace [12]. Pace semantics is based on a Relational Duration Calculus RDC, so he can record just the initial and final values of the variables at a given time point. His semantics is prefix closed with no infinite intervals and he cannot represent infinite computation in zero time (indeed he has a syntactical restriction). Our approach is based on μWDCI, we record not only the initial and final values of the variables but also the intermediate computations. It is a maximal semantics and we have infinite intervals allowing infinite computations that take no time.

We have presented some examples (using the `while` statement) in Section 5.3.2. A more complex example, the STARI, was specified in Verilog in [5].

9.2 Further work

We present here some directions for future work.

1. The V^- subset can be extended with `wait` statements and non-blocking assignment to obtain the V subset of Verilog. We can extend our approach introducing some auxiliary variables to capture the order of execution and the scheduling of the non-blocking assignment.
2. An axiomatisation of μWDCI can be given, developing a proof system for the extension of the logic with fixed point operators and infinite intervals.
3. The general assumption $\mathbf{Ax_1}$ that asserts that $\forall x \in \mathsf{Var}^+ \,.\, \Box(\mathsf{cint} \Rightarrow \mathbf{b}.x = \mathbf{e}.x)$ can be split into two axioms:

   $$\mathbf{Ax_1'} : \forall x \in Dvar \,.\, \Box(\mathsf{cint} \Rightarrow \mathbf{b}.x = \mathbf{e}.x)$$

   $$\mathbf{Ax_1''} : \forall x \in Cvar \,.\, \Box(\mathsf{cint} \Rightarrow \lceil\lceil D \rceil\rceil)$$

   where $\mathsf{Var}^+ = Dvar \cup Cvar$, $Dvar$ is the set of discrete variables, $Cvar$ is the set of continuous variables and D is the dynamics of the system. This can be useful in languages with clocks and/or system time variables that changes in the continuous time domain.

Acknowledgements We would like to thank He Jifeng for introducing Verilog to us, Gordon Pace for discussing his work, and Dang Van Hung, Paritosh Pandya and Zhou Chaochen for their useful comments in the initial stage of this work. The first author also thanks Victor Braberman and Pablo Giambiagi for hours of helpful discussions and comments during the fellowship.

References

[1] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 5(40):269–276, 1991.

[2] Mads Dam. Temporal Logic, Automata and Classical Theories: an Introduction, 1994. Notes for the Sixth European Summer School in Logic, Language and Information. Copenhagen.

[3] F.S. de Boer, H. Tej, W.-P. de Roever, and M. van Hulst. Compositionality in real-time shared variable concurrency.

[4] W.-P. de Roever. The quest for compositionality. In *Proc. of IFIP Working Conf., The Role of Abstract Models in Computer Science*. Elsevier Science B.V. (North-Holland), 1985.

[5] Pablo Giambiagi and Gerardo Schneider. A Verilog specification of STARI. Technical report, UNU/IIST, P.O.Box 3058, Macau, February 1998.

[6] Ulrich Golze. *VLSI Chip Design with the Hardware Description Language VERILOG: An Introduction Based on a Large RISC Processor Design*. Springer Verlag, 1996.

[7] Mike Gordon. The Semantic Challenge of Verilog HDL. In *Tenth Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, pages 136–145, June 1995. Revised version (April 11, 1996) of the paper published in the proceeding.

[8] Mike Gordon. Synthesizable Verilog: syntax and semantics. Available at http://www.cl.cam.ac.uk/users/mjcg/Verilog/V/V.html, June 1997.

[9] Michael R. Hansen and Zhou Chaochen. Lecture Notes on Logical Foundations for the Duration Calculus. Lecture Notes, 13, UNU/IIST, P.O.Box 3058, Macau, August 1993.

[10] IEEE Computer Society. *IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language (IEEE std 1364-1995)*, 1995.

[11] Rajeev Madhavan. Quick Reference for Verilog HDL. Automata Publishing Company, 1995. Available at ftp://ftp.siliconlogic.com:/pub/comp.lang.verilog/ref.tar.Z.

[12] Gordon Pace. Ph.D. thesis. Chapter 3 and 4, 1997.

[13] Paritosh Pandya, Hanpin Wang, and Qiwen Xu. Towards a Theory of Sequential Hybrid Programs. Technical Report 125, UNU/IIST, October 1997. Available at ftp://ftp.iist.unu.edu/pub/techreports/report125.ps.gz.

[14] Paritosh K. Pandya. A Recursive Duration Calculus. Technical Report CS-95/3, Computer Science Group, TIFR, Bombay, January 1995.

[15] Paritosh K. Pandya. A compositional semantics of SL. Technical report, DeTfoRS Group, UNU/IIST, October 1997.

[16] Paritosh K Pandya and Dang Van Hung. Duration Calculus with Weakly Monotonic Time. Technical Report 122, UNU/IIST, P.O. Box 3058 Macau, October 1997. Available at ftp://ftp.iist.unu.edu/pub/techreports/report122.ps.gz.

[17] Paritosh K. Pandya and Y.S. Ramakrishna. A Recursive Mean-Value Calculus. Technical report, Tata Institute of Fundamental Research, Bombay, 1995.

[18] I.C.C. Phillips. Recursion Theory. In *Handbook of Logic in Computer Science. Background: Mathematical Structures*, pages 79–187. 1992.

[19] Gordon D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[20] Colin Stirling. Modal and Temporal Logics. In *Handbook of Logic in Computer Science. Background: Computational Structures*, pages 477–563. 1992.

[21] Hanpin Wang, Qiwen Xu, and Paritosh Pandya. Infinite Duration Calculus with Fixed-point Operators. Technical report, UNU/IIST, 1997.