The Definition of a Temporal Clock Operator

Cindy Eisner[1] Dana Fisman[1,2] John Havlicek[3]
Anthony McIsaac[4] David Van Campenhout[5] ⋆

[1] IBM Haifa Research Laboratory [2] Weizmann Institute of Science
[3] Motorola, Inc. [4] STMicroelectronics, Ltd. [5] Verisity Design, Inc.

Abstract. Modern hardware designs are typically based on multiple clocks. While a singly-clocked hardware design is easily described in standard temporal logics, describing a multiply-clocked design is cumbersome. Thus it is desirable to have an easier way to formulate properties related to clocks in a temporal logic. We present a relatively simple solution built on top of the traditional LTL-based semantics, study the properties of the resulting logic, and compare it with previous solutions.

1 Introduction

Synchronous hardware designs are based on a notion of discrete time, in which the flip-flop (or latch) takes the system from the current state to the next state. A flip-flop or latch is a memory element, which passes on some function of its inputs to its outputs, but only when its clock input is active. The signal that causes the flip-flop (or latch) to transition is termed the *clock*. In a singly-clocked hardware design, the behavior of hardware in terms of the clock naturally maps to the notion of the next-time operator in temporal logics such as LTL[10] and CTL[2], so that the following LTL formula:

$$\mathsf{G}(p \rightarrow \mathsf{X}\, q) \tag{1}$$

can be interpreted as "globally, if p then *at the next clock cycle*, q". Mapping between a state of a model for the temporal logic and a clock cycle of hardware can then be dealt with by the tool which builds a model from the source code (written in some hardware description language, or HDL).

Modern hardware designs, however, are typically based on multiple clocks. In such a design, for instance, some flip-flops may be clocked with *clka*, while others are clocked with *clkb*. In this case, the mapping between states and clock cycles cannot be done automatically; rather, the formula itself must contain some indication of which clock to use. For instance, a clocked version of Formula 1 might be:

$$(\mathsf{G}(p \rightarrow \mathsf{X}\, q))@clka \tag{2}$$

We would like to interpret Formula 2 as "globally, if p during a cycle of *clka*, then at the next cycle of *clka*, q". In LTL we can express this as:

$$\mathsf{G}\ ((clka \wedge p) \rightarrow \mathsf{X}[\neg clka\ \mathsf{W}\ (clka \wedge q)]) \tag{3}$$

⋆ *E-mail addresses*: eisner@il.ibm.com (C. Eisner), dana@wisdom.weizmann.ac.il (D. Fisman), john.havlicek@motorola.com (J. Havlicek), anthony.mcisaac@st.com (A. McIsaac), dvc@verisity.com (D. Van Campenhout)

Thus, we would like to give semantics to a new operator @ such that Formula 2 is equivalent to Formula 3. The issue of defining what such a solution should be for LTL is the problem we explore in this paper.

We present a relatively simple solution built on top of the traditional LTL-based semantics. Our solution is based on the idea that the only role of the clock operator should be to define a projection of the path onto those states where the clock "ticks"[1]. Thus, $\neg(f@clk)$ should be equivalent to $(\neg f)@clk$, that is, the clock operator should be its own dual. Achieving this introduces a problem for paths on which the clock never ticks. We solve this problem by introducing a propositional strength operator that extends the semantics from non-empty paths to empty paths in the same way that the strong next operator [8] extends the semantics from infinite to finite paths. We present the resulting logic LTL$^@$, and show that we meet the goal of the "projection view", as well as other design goals presented below. To show that the clock and propositional strength operators add no expressive power to LTL, we provide a set of rewrite rules to translate an LTL$^@$ formula to an equivalent LTL formula.

2 Related Work

Many modeling languages, such as Lustre [5] and Signal, incorporate the idea of a clock. However, in this paper we are interested in the addition of a clock operator to temporal logic. The work described in this paper is the result of discussions in the LRM subcommittee of the Accellera Formal Verification Technical Committee (FVTC). All four languages (Sugar2.0, ForSpec, Temporal e, CBV) examined by the committee enhance temporal logic with clock operators. Many of these languages distinguish between *strong* and *weak* clock operators, in a similar way as LTL distinguishes between strong and weak until.

Sugar2.0 supports both strong and weak versions of a clock operator. As originally proposed [3], a strongly clocked Sugar2.0 formula requires the clock to "tick long enough to ensure that the formula holds", while a weakly clocked formula allows it to stop ticking before then.

In ForSpec [1], which also supports strong and weak clocks, a strongly clocked formula requires only that the clock tick at least once, after which the only role of the clock is to define the projection of the path onto those states where the clock ticks. A weakly clocked formula, on the other hand, holds if the clock never ticks; if it does tick, then the role of the clock is the same as for a strongly clocked formula.

In Temporal e [9], which also supports multiple clocks, clocks are not attributed with strength. This is consistent with the use of Temporal e in simulation, in which behaviors are always finite in duration. Support for reasoning about infinite length behaviors is limited in Temporal e.

In CBV [6], clocking and alignment of formulas are supported by separate and independent sampling and alignment operators. The sampling operator is self-dual and

[1] Actually, referring to a projection of the path is not precisely correct, as we allow access to states in between consecutive states of a projection in the event of a clock switch. However, the word "projection" conveys the intuitive function of the clock operator in the case that the formula is singly-clocked. Use of the word "projection" when describing the clocks of Sugar2.0 and ForSpec below is similarly imprecise.

determines the projection in the singly-clocked case. It is similar to the clock operator of LTL$^{@}$. The CBV alignment operators come in a strong/weak dual pair that take us to the first clock event, without affecting the sampling clock. The composition of the sampling operator with a strong/weak alignment operator on the same clock is provided by the CBV synchronization operators, which behave like the ForSpec strong/weak clock operators.

Clocked Temporal Logic [7], confusingly termed CTL by its authors, is another temporal logic that deals with multiple clocks. However, in their solution a clock is a predetermined subset of the states on a path, and their approach is to associate a clock with each atomic proposition, rather than to clock formulas and sub-formulas.

Wang, Mok and Emerson have defined APTL [11], which enhances temporal logic with multiple real-time clocks. In this work, we are concerned with hardware clocks, which determine the granularity of time in a synchronous system, rather than with clocks in the sense of [11] that measure real time in an asynchronous system. Thus, for example [11] assumes the clock ticks infinitely often, while we are trying to face the problems arising when such an assumption is not adopted.

3 Issues in Defining the Clock Operator

We begin by trying to set the design requirements for the clock operator. What is the intuition it should capture? What are the problems involved?

The projection view When only a single clock is involved we would like that a clocked formula $f@clk$ hold on a path π if and only if the unclocked formula f holds on a path π' where π' is π projected onto those states where clk holds.

Non-accumulation of clocks In many hardware designs, large chunks of the design work on some main clock, while small pieces work on a secondary clock. Rather than require the user to specify a clock for each sub-formula, we would like to allow clocking of an entire formula on a main clock, and pieces of it on a secondary clock, in such a way that the outer clock (which is applied to the entire formula) does not affect the inner clock (which is applied to one or more sub-formulas). That is, we want a nested clock operator to have the effect of "changing the projection", rather than further projecting the projected path.

Finite and empty paths The introduction of clocks requires us to deal with finite paths, since the projection of an infinite path may be finite. For LTL, this means that the single *next operator* X no longer suffices. To see why, consider an atomic proposition p and a path where the clock stops ticking. On the last state of the path, do we want $(\mathsf{X}\ p)@clk$ to hold or not? Whatever we do, assuming we want to preserve the duality $\neg(\mathsf{X}\ p) = \mathsf{X}(\neg p)$ under clocks, and thus obtain a definition under which $\neg((\mathsf{X}\ p)@clk)$ is equivalent to $(\mathsf{X}(\neg p))@clk$, the result is unsatisfactory. For instance, if $(\mathsf{X}\ p)@clk$ holds when the clock stops ticking, then $\neg((\mathsf{X}\ p)@clk)$ does not. Letting $p = \neg q$, we get that $(\mathsf{X}\ q)@clk$ does not hold if the clock stops ticking, which is a contradiction.

Thus, the addition of clocks to LTL-based semantics introduces problems similar to those of defining LTL semantics for finite paths. In particular, it requires us to make a decision as to the semantics of the next operator on the last clock tick of a path, with the result that the next operator is not dual to itself. Instead, we end up with two next operators, strong and weak, which are dual to each other [8].

Not only may the projection of an infinite path be finite, it may be empty as well. For LTL, this means that the duality problem exists not only for the next operator, but also for atomic propositions. Whatever choice we make for the semantics of $p@clk$ (where p is an atomic proposition) on an empty path, we cannot achieve the duality $\neg(p@clk) = (\neg p)@clk$ without adding something to the logic.

A natural solution for the semantics of a formula over a path where the clock does not tick is to take the strength from the temporal operator. Under this approach, for example, a clocked strong next does not hold on a path with no ticks, while a clocked weak next does hold on such a path. This solution breaks down in the case of a formula with no temporal operators. One way to deal with this is to make a decision as to the semantics of the clock operator on a path with no ticks, giving two clock operators which are dual to each other, rather than a single clock operator that is dual to itself. Below we discuss this issue in more detail.

Avoiding the problems of existing distinctions between strong and weak clocks Three of the languages considered by the FVTC make a distinction between strong and weak clocks. Each has significant drawbacks that we would like to avoid.

In Sugar2.0 as originally proposed [3], a strongly clocked formula requires the clock to "tick long enough to ensure that the formula holds", while a weakly clocked formula allows it to stop ticking before then. Thus, for instance, the formula $(\mathsf{F}\ p)@clk!$ (where $@$ is the clock operator, clk is the clock, and the ! indicates that it is strong) requires there to be enough ticks of clk so that p eventually holds, whereas the formula $(\mathsf{F}\ p)@clk$ (which is a weakly clocked formula, because there is no !) allows the case where p never occurs, if it "is the fault of the clock", i.e., if the clock ticks a finite number of times. Negation switches the clock strength, so that $\neg(f@clk) = (\neg f)@clk!$ and we get that $(\mathsf{G}\ q)@clk!$ holds if the clock ticks an infinite number of times and q holds at every tick, while $(\mathsf{G}\ q)@clk$ holds if q holds at every tick, no matter how many there are. Although initially pleasing, this semantics has the disadvantage that the formula $(\mathsf{F}\ p) \wedge (\mathsf{G}\ q)$ cannot be satisfactorily clocked for a finite path, because $((\mathsf{F}\ p) \wedge (\mathsf{G}\ q))@clk!$ does not hold on any finite path, while $((\mathsf{F}\ p) \wedge (\mathsf{G}\ q))@clk$ makes no requirement on p on such a path. Since our intent is to define a semantics that can be used in simulation (where every path is finite) as well as in model checking, this is unacceptable.

In ForSpec, a strongly clocked formula requires only that the clock tick at least once, after which the only role of the clock is to define the projection of the path onto those states where the clock ticks. A weakly clocked formula, on the other hand, holds if the clock never ticks; if it does tick, then the role of the clock is the same as for a strongly clocked formula. Thus, the only difference between strong and weak clocks in ForSpec is on paths whose projection is empty. This leads to the strange situation that a liveness formula may hold on some path π, but not on an extension of that path, $\pi\pi'$. For instance, if p is an atomic proposition, then $(\mathsf{F}\ p)@clk$ holds if there are no ticks of clk, but does not hold if there is just one tick, at which p does not hold.

In CBV, there is a self-dual clock operator, the sampling operator, according to which all temporal advances are aligned to the clock. However, the sampling operator causes no initial alignment. Therefore, sampled booleans are evaluated immediately; sampled next-times align to the next strictly future tick of the clock; and so forth. As a result, the projection defined by the CBV sampling operator includes the first state of a path, regardless of whether it is a tick of the clock. The CBV alignment and synchronization operators come in strong/weak dual pairs. The latter behave like the ForSpec strong/weak clock operators and therefore suffer from the same disadvantages.

Under the solutions described above, the clock or synchronization operator is given the role of determining the semantics in case the path is empty. As a result, the operator cannot be its own dual, resulting in two kinds of clocks. Our goal is to define a logic where the only role of the clock operator is to determine a projection. Thus, we seek a solution which solves the problem of an empty path in such a way that the clock operator is its own dual, eliminating the need for two kinds of clocks.

Equivalence and substitution We would like the logic to adhere to an equivalence lemma as well as a substitution lemma. Loosely speaking, an equivalence lemma requires that two equivalent LTL formulas remain equivalent after the application of the clock operator. The substitution lemma guarantees that substituting sub-formula g for an equivalent sub-formula h does not change the truth value of the original formula.

Motivating example We would like our original motivating example from the introduction to hold.

Goals

To summarize, our goals composed in light of the discussion above, are as follows:

1. When singly-clocked, the semantics should be that of the projection view.
2. Clocks should not accumulate.
3. The clock operator should be its own dual.
4. There should be a clocked version of $(\mathsf{F}\ p) \wedge (\mathsf{G}\ q)$ that is meaningful on paths with a finite number of clock ticks.
5. For any atomic proposition p, if $(\mathsf{F}\ p)@clk$ holds on a path, it should hold on any extension of that path.
6. For any clock c, two equivalent LTL formulas should remain equivalent when clocked with c.
7. Substituting sub-formula g for an equivalent sub-formula h should not change the truth value of the original formula.
8. The truth value of LTL$^{@}$ Formula 2 should be the same as the truth value of LTL Formula 3 for every path.

4 The Definition of LTL$^{@}$

We solve the problem of finite paths introduced by clocks in LTL-based semantics by supplying both strong and weak versions of the next operator ($\mathsf{X!}$ and X).

We solve the problem of empty paths by introducing a new, propositional strength operator. Thus, if p is an atomic proposition, then $p!$ is as well. While p is a weak atomic proposition, and so holds on an empty path, $p!$ is a strong atomic proposition, and does not hold on such a path. The intuition behind this is that the role of the strength of a temporal operator is to tell us how far a finite path is required to extend. For strong until, as in $[f \mathsf{U} g]$, we require that g hold somewhere on the path. For strong next, as in $\mathsf{X}!\ f$, we require that there be a next state. Intuitively then, we get that a strong proposition, as in $p!$, requires that there be a current state.

Without clocks, there is never such a thing as not having a current state, so the problem of an empty path does not come up in traditional temporal logics. But for a clocked semantics, there may indeed not be a first state. In such a situation, putting the responsibility on the atomic proposition gives a natural extension to the idea of the formula itself telling us how far a finite path must extend. This leaves us with the desired situation that the sole responsibility of the clock operator will be to "light up" the states that are relevant for the current clock context, which is the intuitive notion of a clock.

4.1 Syntax

The syntax of LTL$^{@}$ is defined below, where we use the term *boolean expression* to refer to any application of the standard boolean operators to atomic propositions.

Definition 1 (Formulas of LTL$^{@}$).

- *If p is an atomic proposition, then p and $p!$ are* LTL$^{@}$ *formulas.*
- *If clk is a boolean expression and f, f_1, and f_2 are* LTL$^{@}$ *formulas, then the following are* LTL$^{@}$ *formulas: $\neg f$, $f_1 \wedge f_2$, $\mathsf{X}!\ f$, $[f_1\ \mathsf{U}\ f_2]$, $f\,@clk$.*

Additional operators are derived from the basic operators defined above:[2]

- $f_1 \vee f_2 \stackrel{\text{def}}{=} \neg(\neg f_1 \wedge \neg f_2)$
- $f_1 \rightarrow f_2 \stackrel{\text{def}}{=} \neg f_1 \vee f_2$
- $\mathsf{F}\ f \stackrel{\text{def}}{=} [\mathrm{T}\ \mathsf{U}\ f]$
- $\mathsf{X}\ f \stackrel{\text{def}}{=} \neg\mathsf{X}!\ \neg f$
- $\mathsf{G}\ f \stackrel{\text{def}}{=} \neg\mathsf{F}\ \neg f$
- $[f_1\ \mathsf{W}\ f_2] \stackrel{\text{def}}{=} [f_1\ \mathsf{U}\ f_2] \vee \mathsf{G}\ f_1$

LTL is the subset of LTL$^{@}$ consisting of the formulas that have no clock operator and no sub-formulas of the form $p!$, for some atomic proposition p.

4.2 Semantics

We define the semantics of LTL$^{@}$ formulas over words[3] from the alphabet 2^P. A letter is a subset of the set of atomic propositions P such that T belongs to the subset and F does not. We will denote a letter from 2^P by ℓ and an empty, finite, or infinite word from 2^P by w. We denote the length of word w as $|w|$. An empty word $w = \epsilon$ has length 0, a finite word $w = (\ell_0\ell_1\ell_2\cdots\ell_n)$ has length $n+1$, and an infinite word has length ∞. We denote the i^{th} letter of w by w^i. We denote by $w^{i\cdot\cdot}$ the suffix of w starting at w^i. That

[2] Where T is an atomic proposition that holds on every state. In the sequel, we also use F, which is an atomic proposition that does not hold for any state.

[3] Relating the semantics over words to semantics over models is done in the standard way. Due to lack of space, we omit the details.

is, $w^{i\cdot\cdot} = (w^i w^{i+1} \cdots w^n)$ or $w^{i\cdot\cdot} = (w^i w^{i+1} \cdots)$. We denote by $w^{i\cdot\cdot j}$ the finite sequence of letters starting from w^i and ending in w^j. That is, $w^{i\cdot\cdot j} = (w^i w^{i+1} \cdots w^j)$.

We first present the semantics of LTL$^{@}$ minus the clock operator over infinite, finite, and empty words (*unclocked semantics*). We then present the semantics of LTL$^{@}$ over infinite, finite, and empty words (*clocked semantics*). Later, we relate the two.

Unclocked semantics We now present a semantics for LTL$^{@}$ minus the clock operator. The semantics is defined with respect to an infinite, finite, or empty word. The notation $w \models f$ means that formula f holds along the word w. The semantics is defined as follows, where p denotes an atomic proposition, f, f_1, and f_2 denote formulas, and j and k denote natural numbers (i.e., non-negative integers).

- $w \models p \iff |w| = 0$ or $p \in w^0$
- $w \models p! \iff |w| > 0$ and $p \in w^0$
- $w \models \neg f \iff w \not\models f$
- $w \models f_1 \wedge f_2 \iff w \models f_1$ and $w \models f_2$
- $w \models \mathsf{X}!\ f \iff |w| > 1$ and $w^{1\cdot\cdot} \models f$
- $w \models [f_1\ \mathsf{U}\ f_2] \iff$ there exists $k < |w|$ such that $w^{k\cdot\cdot} \models f_2$, and for every $j < k$ $w^{j\cdot\cdot} \models f_1$

Clocked semantics We define the semantics of an LTL$^{@}$ formula with respect to an infinite, finite, or empty word w and a context c, where c is a boolean expression over P. For word w and boolean expression b, we say that $w^i \models b$ iff $w^{i\cdot\cdot i} \models b$.

Definition 2 (is a clock tick of). *We say that finite word* w is a clock tick of c *iff* $|w| > 0$ *and* $w^{|w|-1} \models c$ *and for every natural number* $i < |w| - 1$, $w^i \not\models c$.

The notation $w \models^c f$ means that formula f holds along the word w in the context of clock c. The semantics of an LTL$^{@}$ formula is defined as follows, where p denotes an atomic proposition, c, and c_1 denote boolean expressions, f, f_1, and f_2 denote LTL$^{@}$ formulas, and j and k denote natural numbers.

- $w \models^c p \iff$ for all $j < |w|$ such that $w^{0\cdot\cdot j}$ is a clock tick of c, $p \in w^j$
- $w \models^c p! \iff$ there exists $j < |w|$ such that $w^{0\cdot\cdot j}$ is a clock tick of c and $p \in w^j$
- $w \models^c \neg f \iff w \not\models^c f$
- $w \models^c f_1 \wedge f_2 \iff w \models^c f_1$ and $w \models^c f_2$
- $w \models^c \mathsf{X}!\ f \iff$ there exist $j < k < |w|$ such that $w^{0\cdot\cdot j}$ is a clock tick of c and $w^{j+1\cdot\cdot k}$ is a clock tick of c and $w^{k\cdot\cdot} \models^c f$
- $w \models^c [f_1\ \mathsf{U}\ f_2] \iff$ there exists $k < |w|$ such that $w^k \models c$ and $w^{k\cdot\cdot} \models^c f_2$ and for every $j < k$ such that $w^j \models c$, $w^{j\cdot\cdot} \models^c f_1$
- $w \models^c f @ c_1 \iff w \models^{c_1} f$

In LTL$^{@}$, every formula is evaluated in the context of a clock. The projection view requires that propositions are evaluated not at the first state of a path, but at the first state where the context clock ticks (if there is such a state). To be consistent with this, if the clock does not tick in the first state of a path, a formula $\mathsf{X}f$ or $\mathsf{X}!f$ must be evaluated in terms of the value of f at the second tick of the clock after the initial state.

5 Meeting the goals

In this section, we analyze the logic LTL$^{@}$ with respect to the goals of Section 3. Due to lack of space all proofs are omitted; they can be found in the full version of the paper. The following definitions are needed for the sequel.

Definition 3 (Projection). *The projection of word w onto clock c, denoted $w|_c$, is the word obtained from w after leaving only the letters which satisfy c.*

Definition 4 (Unclocked equivalent). *Two* LTL$^{@}$ *formulas f and g with no clock operator are unclocked equivalent ($f \equiv g$) if for all words w, $w \models f$ if and only if $w \models g$.*

Definition 5 (Clocked equivalent). *Two* LTL$^{@}$ *formulas f and g are clocked equivalent ($f \overset{@}{\equiv} g$) if for all words w and all contexts c, $w \overset{c}{\models} f$ if and only if $w \overset{c}{\models} g$.*

Goal 1 The following theorem states that when a single clock is applied to a formula, the projection view is obtained.

Theorem 1 *Let f be an* LTL$^{@}$ *formula with no clock operator, c a boolean expression and w an infinite, finite, or empty word.*

$$w \overset{c}{\models} f \quad \text{if and only if} \quad w|_c \models f$$

Corollary 1 *Let f be an* LTL$^{@}$ *formula with no clock operator, and w a word.*

$$w \overset{\mathrm{T}}{\models} f \quad \text{if and only if} \quad w \models f$$

Goal 2 Looking at the semantics for $f@c_1$ in context c it is easy to see that $f@c_1@c_2 \overset{@}{\equiv} f@c_1$, and therefore clocks do not accumulate.

Goal 3 The following claim states that this goal is met.

Claim. $(\neg f)@b \overset{@}{\equiv} \neg(f@b)$

Goal 4 The clocked version of $(\mathsf{F}\ p) \wedge (\mathsf{G}\ q)$ is $((\mathsf{F}\ p) \wedge (\mathsf{G}\ q))@c$, and holds if p holds for some state and q holds for all states on the projected path.

Goal 5 The following claim states that Goal 5 is met.

Claim. Let b, clk and c be boolean expressions, w a finite word, and w' an infinite or finite word.

$$w \overset{c}{\models} (\mathsf{F}\ b)@clk \implies ww' \overset{c}{\models} (\mathsf{F}\ b)@clk$$

Goal 6 The following claim states that Goal 6 is met.

Claim. Let f and g be LTL$^{@}$ formulas with no clock operators, and let b be a boolean expression.

$$f \equiv g \quad \Longrightarrow \quad f@b \stackrel{@}{\equiv} g@b$$

Note that if f and g are unclocked formulas then for some boolean expression c it may be that $f@c \stackrel{@}{\equiv} g@c$, even though $f \not\equiv g$. For example, let $f = (\neg c) \rightarrow \mathrm{T}$ and let $g = (\neg c) \rightarrow \mathrm{F}$. Then $f@c \stackrel{@}{\equiv} g@c$, but $f \not\equiv g$.

Goal 7 We use the notation $\varphi[\psi \leftarrow \psi']$ to denote the formula obtained from φ by replacing sub-formula ψ with ψ'. The following claim states that this goal is met.

Claim. Let g be a sub-formula of f, and let $g' \stackrel{@}{\equiv} g$. Then $f \stackrel{@}{\equiv} f[g \leftarrow g']$.

Goal 8 The following claim states that this goal is met.

Claim. For every word w,

$$w \stackrel{\mathrm{T}}{\models} (\mathsf{G}(p \rightarrow \mathsf{X}\ q))@clka \Longleftrightarrow w \models \mathsf{G}\ ((clka \wedge p) \rightarrow \mathsf{X}[\neg clka\ \mathsf{W}\ (clka \wedge q)])$$

6 Discussion

Looking backwards In LTL, the evaluation of formula $\mathsf{G}(p \rightarrow f)$, where p is a boolean expression, depends only on the evaluations of f starting at those points where p holds. In particular, satisfaction of $\mathsf{G}(p \rightarrow f)$ on w is independent of the initial segment of w before the first occurrence of p. We might hope that satisfaction of $\mathsf{G}(p@clkp \rightarrow f@clkf)$ on w will be independent of the initial segment of w before the first occurrence of p at a tick of $clkp$. This is not the case. For instance, consider the following formula:

$$\mathsf{G}(p@clkp \rightarrow q@clkq) \tag{4}$$

which is a clocked version of the simple invariant $\mathsf{G}(p \rightarrow q)$, where both p and q are boolean expressions. Formula 4 can be rewritten as

$$\mathsf{G}([\neg clkp\ \mathsf{W}\ (clkp \wedge p)] \rightarrow [\neg clkq\ \mathsf{W}\ (clkq \wedge q)]) \tag{5}$$

by the rewrite rules in Theorem 2 below. The result is a dimension of temporality not present in the original, unclocked formula. For instance, for the behavior of p shown in Figure 1, Formula 5 requires that q hold at time 4 (because $[\neg clkp\ \mathsf{W}\ (clkp \wedge p)]$ holds at time 3, and in order for $[\neg clkq\ \mathsf{W}\ (clkq \wedge q)]$ to hold at time 3, we need q to hold at time 4). Not only does Formula 5 require that q hold at time 4 for the behavior of p shown in Figure 1, it also requires that q hold at time 2 (because $[\neg clkp\ \mathsf{W}\ (clkp \wedge p)]$ holds at time 2, and in order for $[\neg clkq\ \mathsf{W}\ (clkq \wedge q)]$ to hold at time 2, we need q to hold at time 2). Thus, the direction of the additional dimension of temporality may be backwards as well as forwards.

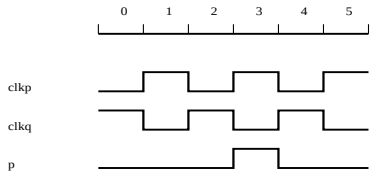

Fig. 1. Behavior of p illustrating a problem with Formula 5

To avoid the "looking backward" phenomenon the semantics of a boolean expression under clock operators should be non-temporal. For instance, we could define $p@clk = p$ and $p!@clk = p!$, or alternatively, $p@clk = clk \rightarrow p$ and $p!@clk = clk \wedge p!$. The disadvantage of these definitions is that the projection view is not preserved (because on a path such that p holds at the first clock but does not hold at the first state, $p@clk$ and/or $p!@clk$ do/does not hold).

We note that Formula 4 has the same backwards-looking feature in other semantics with strong and weak clocks [3, 1], so the phenomenon does not arise purely from the design decisions we have taken here. Furthermore, if the multi-clocked version is taken as $(\mathsf{G}(p \rightarrow (q@clkq)))@clkp$, then the phenomenon does not arise. Many properties of practical interest are of this form, for example a property asserting that the data is not corrupted between input and output interfaces clocked on different clocks:

$$(\mathsf{G}((receive \wedge (data\_in = d)) \rightarrow (\neg send \,\mathsf{U}\, (send \wedge (data\_out = d)))@clk\_out))@clk\_in \quad (6)$$

$[f \;\mathsf{U}\; g]$ as a fixed point In standard LTL, $[f \;\mathsf{U}\; g]$ can be defined as a least solution of the equation $S = g \vee (f \wedge \mathsf{X}!\; S)$. In LTL$^@$, there is a fixed point characterization if f and g are themselves unclocked, because $[f \;\mathsf{U}\; g] \equiv (\mathrm{T}! \wedge g) \vee (f \wedge \mathsf{X}![f \;\mathsf{U}\; g])$ (the conjunction with T! is required in order to ensure equivalence on empty paths as well as the non-empty paths on which standard LTL formulas are interpreted). Thus by the claim of Goal 6 $[f \;\mathsf{U}\; g]@c \stackrel{@}{\equiv} ((\mathrm{T}! \wedge g) \vee (f \wedge \mathsf{X}![f \;\mathsf{U}\; g]))@c$ for any clock c and any formulas f and g containing no clock operators, and hence by the semantics, the truth value of $[f \;\mathsf{U}\; g]$ under context c is the same as the truth value of $(\mathrm{T}! \wedge g) \vee (f \wedge \mathsf{X}![f \;\mathsf{U}\; g])$ under context c, for any context c. If f and g contain clock operators, this equivalence no longer holds. Let p, q and d be atomic propositions, and let $f = q@d$. Consider a word w such that $w^0 \models d \wedge q$ and for all $i > 0$, $w^i \not\models d \wedge q$, and $w^0 \not\models c$. Then $w \stackrel{c}{\models} f$ hence $w \stackrel{c}{\models} (\mathrm{T}! \wedge f) \vee (p \wedge \mathsf{X}![p \;\mathsf{U}\; f])$. However, since $w^0 \not\models c$, and there is no state other than w^0 where $d \wedge q$ holds, $w \stackrel{G}{\not\models} [p \;\mathsf{U}\; f]$. Note that while of theoretical interest, the lack of a fixed point characterization of $[f \;\mathsf{U}\; g]$ is not an obstacle to model checking, since any LTL$^@$ formula can be translated to an equivalent LTL formula by the rewrite rules presented below.

$\mathsf{X}f$ and $\mathsf{X}!f$ on states where the clock does not hold As mentioned earlier, another property of our definition is that on states where the clock does not hold, the next operators take us two clock cycles into the future, instead of the one clock cycle

that we might expect. Further consideration shows that this is a direct result of the projection view: since $p@clk$ must mean that p holds at the next clock, it is clear that an application of a next operator (as in $(\mathsf{X}p)@clk$ or $(\mathsf{X}!p)@clk$) must mean that p holds at the one after that. This behavior of a clocked next operator is a consideration only in multi-clocked formulas, since in a singly-clocked formula, we are never "at" a state where the clock does not hold (except perhaps at the initial state).

Expressive power The clock operator provides a concise way to express what would otherwise be cumbersome, but it does not add expressive power. Theorem 2 below states that the truth value of any LTL$^@$ formula under context clk is the same as that of the LTL formula $f' = \mathcal{T}^{clk}(f)$, where $\mathcal{T}^{clk}(f)$ is defined as follows:

- $\mathcal{T}^{clk}(p) = [\neg clk\ \mathsf{W}\ (clk \wedge p)]$
- $\mathcal{T}^{clk}(p!) = [\neg clk\ \mathsf{U}\ (clk \wedge p)]$
- $\mathcal{T}^{clk}(\neg f) = \neg\mathcal{T}^{clk}(f)$
- $\mathcal{T}^{clk}(f_1 \wedge f_2) = \mathcal{T}^{clk}(f_1) \wedge \mathcal{T}^{clk}(f_2)$
- $\mathcal{T}^{clk}(\mathsf{X}!\ f) = [\neg clk\ \mathsf{U}\ (clk \wedge \mathsf{X}![\neg clk\ \mathsf{U}\ (clk \wedge \mathcal{T}^{clk}(f))])]$
- $\mathcal{T}^{clk}([f_1\ \mathsf{U}\ f_2]) = [(clk \rightarrow \mathcal{T}^{clk}(f_1))\ \mathsf{U}\ (clk \wedge \mathcal{T}^{clk}(f_2))]$
- $\mathcal{T}^{clk}(f@clk_1) = \mathcal{T}^{clk_1}(f)$

Theorem 2 *Let f be any* LTL$^@$ *formula, c a boolean expression, and w a word.*

$$w \stackrel{c}{\models} f \quad \textit{if and only if} \quad w \models \mathcal{T}^c(f)$$

Clearly $\mathcal{T}^{clk}()$ defines a recursive procedure whose application starting with $clk = \mathrm{T}$ results in an LTL formula with the same truth value in context T. Note that while we can rewrite a formula f into an LTL formula f' with the same truth value, we cannot use formulas f and f' interchangeably. For example, $p!@clk1$ translates to $[\neg clk1\ \mathsf{U}\ (clk1 \wedge p)]$, but these two are not clocked equivalent (because clocking each of them with $clk2$ will give different results).

7 Conclusion and future work

We have given a relatively simple definition of multiple clocking for LTL augmented with a clock operator that we believe captures the intuition behind hardware clocks, and have presented a set of rewrite rules that can be used as an implementation of the clock operator. In our definition, the only role of the clock operator is to define a projection of the path, and it is its own dual.

Our semantics, based on strong and weak propositions, achieves goals not achieved by semantics based on strong and weak clocks. In particular, it gives the projection view for singly-clocked formulas and a uniform treatment of empty and non-empty paths, including the interpretation of the operators G and F. It does not provide an easy solution to the question of how to define U as a fixed point operator for multi-clocked formulas. Future work should seek a way to resolve these issues without losing the advantages.

It may be noted that in the strong/weak clock semantics, alignment is always applied immediately after the setting of a clock context; while in the strong/weak proposition semantics, it is always applied immediately before an atomic proposition. Allowing more flexibility in where alignment (and strength) is applied may be a useful avenue for investigation.

Acknowledgements

We would like to thank Sharon Barner, Shoham Ben-David, Alan Hartman and Emmanuel Zarpas for their help with the formal definition of multiple clocks. We would also like to thank Mike Gordon, whose work on studying the formal semantics of Sugar2.0 with HOL [4] greatly contributed to our understanding of the problems discussed in this paper. Finally, thank you to Shoham Ben-David, Avigail Orni and Sitvanit Ruah for careful review and important comments.

References

1. R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification language. In J.-P. Katoen and P. Stevens, editors, *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2280 of *Lecture Notes in Computer Science*. Springer, 2002.
2. E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logics of Programs*, LNCS 131, pages 52–71. Springer-Verlag, 1981.
3. C. Eisner and D. Fisman. Sugar 2.0 proposal presented to the Accellera Formal Verification Technical Committee, March 2002. At http://www.haifa.il.ibm.com/projects/verification/sugar/Sugar_2.0_Accellera.ps.
4. M. J. C. Gordon. Using HOL to study Sugar 2.0 semantics. In *Proc. 15th International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, NASA Conference Proceedings CP-2002-211736, 2002.
5. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.
6. J. Havlicek, N. Levi, H. Miller, and K. Shultz. Extended CBV statement semantics, partial proposal presented to the Accellera Formal Verification Technical Committee, April 2002. At http://www.eda.org/ vfv/hm/att-0772/01-ecbv_statement_semantics.ps.gz.
7. C. Liu and M. Orgun. Executing specifications of distributed computations with Chronologic(MC). In *Proceedings of the 1996 ACM Symposium on Applied Computing (SAC), February 17-19, 1996, Philadelphia, PA, USA*. ACM, 1996.
8. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*, pages 272–273. Springer-Verlag, New York, 1995.
9. M. Morley. Semantics of temporal e. In T. F. Melham and F. G. Moller, editors, *Proc. Banff'99 Higher Order Workshop (Formal Methods in Computation)*, 1999. University of Glasgow, Dept. of Computing Science Technical Report.
10. A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
11. F. Wang, A. K. Mok, and E. A. Emerson. Distributed real-time system specification and verification in APTL. *ACM Transactions on Software Engineering and Methodology*, 2(4):346–378, Oct. 1993.