Temporal Modalities for Concisely Capturing Timing Diagrams

Hana Chockler[1,2] and Kathi Fisler[1]

[1] Department of Computer Science, WPI,
100 Institute Road, Worcester, MA 01609, USA.
hanac@cs.wpi.edu, kfisler@cs.wpi.edu
[2] MIT CSAIL, 32 Vassar street,
Cambridge, MA 02139, USA.

Abstract. Timing diagrams are useful for capturing temporal specifications in which all mentioned events are required to occur. We first show that translating timing diagrams with both partial orders on events and don't-care regions to LTL potentially yields exponentially larger formulas containing several non-localized terms corresponding to the same event. This raises a more fundamental question: which modalities allow a textual temporal logic to capture such diagrams using a single term for each event? We define the shapes of partial orders that are captured concisely by a hierarchy of textual linear temporal logics containing future and past time operators, as well Laroussinie *et al*'s forgettable past operator and our own unforeseen future operator. Our results give insight into the temporal abstractions that underlie timing diagrams and suggest that the abstractions in LTL are significantly weaker than those captured by timing diagrams.

1 Introduction

Timing diagrams are a commonly used visual notation for temporal specifications. Although designers instinctively know when information can conveniently be expressed as a timing diagram, few researchers have explored the formal connections between timing diagrams and textual temporal logics. Understanding these formal connections would be useful for understanding what makes specifications designer-friendly, as well as for developing tools to visualize temporal logic specifications. Ideally, we would like to have constructive decision procedures for determining when a specification, given in a temporal logic or a specification language (such as LTL or PSL), can be rendered as a timing diagram. These could aid in both understanding and debugging specifications.

Identifying diagrammable LTL specifications appears to be very hard. Its complexity stems partly from the fact that a timing diagram contains several different visual elements (events, orderings and timings between events, event synchronization) which must be located within the more uniform syntax of a temporal logic formula. In addition, LTL formulas that capture timing diagrams appear to be at least one order of magnitude (and sometimes two) larger than the diagrams and use multiple logical terms for the same event. Before we can write an algorithm to recognize timing diagrams in temporal logic formulas, we need to understand how the patterns over visual elements that underlie timing diagrams would appear textually.

This paper explores this question by trying to identify textual temporal logic operators that capture timing diagrams concisely; the rendering problem would then reduce to recognizing uses of these operators in LTL. The core of a timing diagram is the partial order it imposes on events. We view a formula as capturing a partial order *concisely* if the formula characterizes instances of the partial order using *exactly one term* for each event in the partial order. We study a progression of linear temporal logics including LTL, PLTL, PLTL with forgettable past [10] and PLTL with forgettable past and unforeseeable future (which we have defined for this work). We identify the set of partial orders that each logic can capture concisely and show that some partial orders defy concise representation in even the richest of these logics. We do not address the rendering question in this paper, as our results indicate that additional theoretical work is required before pursuing that question.

Our results cover timing diagrams with both partial orders and don't-care regions (Section 2). To illustrate the subtleties in representing timing diagrams in LTL, Section 3 presents a translation from diagrams to LTL and argues that a small translation seems impossible. We provide a counterexample that shows that introducing don't-care regions explodes the size of the formula by forcing it to separately handle all possible total-order instances of the partial order. Section 4 presents our algorithm for efficient translation of a particular class of diagrams to formulas in LTL with the past-time and forgettable-past-and-future modalities; this section also identifies a class of diagrams that this logic cannot capture. Related work is mentioned throughout the paper.

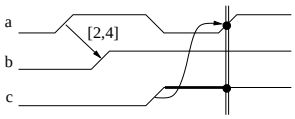
2 Timing diagrams

Timing diagrams depict changes in values on signals (*events*) over time. Figure 1 shows an example. Waveforms capture each signal (lower horizontal lines represent false and higher ones true), arrows order events, arrow annotations constrain the time within which the tail event must follow the head event, vertical lines synchronize behavior across signals, and bold lines indicate *care regions* in which the signal must hold the depicted value. Non-care regions between events are called *don't-care regions*; they allow the signal value to vary before the event at the end of the region occurs. The diagram in Figure 1 has signals a, b, and c. The rising transition on b must occur 2 to 4 cycles after the rising transition on a. The falling transition on a and the rising transitions on b and c may occur in any order (since no arrows order them). Once c rises, it must remain true until the second rising transition on a (due to the care region on c and the vertical lines that synchronize the transition on a and the value on c into a single event). The value of b may vary after its rise, since a don't-care region follows the rise.

The timing diagrams literature contains many variations on this core notation: diagrams may support events that contain no transitions, busses, bi-directional arrows, assumption events (to represent input from the environment) [3], or combinations of timing diagrams using regular expression operators [2]. This paper considers timing diagrams with don't-care regions and partial orders between events.

Definition 1 The syntax of timing diagrams is captured as follows:

– A *signal* is a proposition; p and $\neg p$ denote true and false values on signal p.



a	0	1	0	0	0	0	1	1	1	1
b	0	0	0	0	1	1	0	0	1	0
c	0	0	1	1	1	1	0	0	0	1
	0	1	2	3	4	5	6	7	8	9

Fig. 1. A timing diagram with a word that satisfies its semantics.

- A *transition* is a proposition annotated with a directional change: $p\downarrow$ and $p\uparrow$ denote falling and rising transitions, respectively.
- An *event* is a conjunction of values and at least one transition on signals.
- A *timing diagram* is a tuple $\langle E, C, M\rangle$ where E is a set of events, C (the care regions) is a set of tuples $\langle e_1, e_2, p, v\rangle$ where e_1 and e_2 are (uniquely identified[1]) events in E, p is a signal name and v is a boolean value, and M (the timing constraints) is a set of tuples $\langle e_1, e_2, l, u\rangle$ where e_1 and e_2 are events in E, l is a positive integer, and u is either an integer at least as large as l or the symbol ∞. For each signal, any region that is not within a care region is called a *don't-care region*.

The semantics of timing diagrams is defined in terms of languages over finite or infinite words in which characters are assignments of boolean values to signals. A word models a timing diagram if the earliest occurrence of each event that respects the partial order in the timing constraints respects the care regions and durations of timing constraints. The earliest occurrence requirement keeps the language unambiguous. Formally, we define what it means for an index into a word to satisfy an event, map events to indices in the word, and check that those mappings respect the diagram.

Definition 2 Let E be an event $v_1 \wedge \ldots \wedge v_k$ where each v_i is a proposition, its negation, or a rising or falling transition on a proposition. Let W be a word and i an index into W. Let $W_i(q)$ denote the value of proposition q at index i of W. Index i *satisfies* E if for every v_i, $W_i(p) = 0$ if $v_i = \neg p$, $W_i(p) = 1$ if $v_i = p$, $W_i(p) = 0$ and $W_{i+1}(p) = 1$ if $v_i = p\uparrow$, and $W_i(p) = 1$ and $W_{i+1}(p) = 0$ if $v_i = p\downarrow$.

Definition 3 Let $\langle E, C, M\rangle$ be a timing diagram, W be a word, and I a function from E to indices into W (I is called an *index assignment*). I is *valid* iff

- For every event $e \in E$, $I(e)$ satisfies e,
- For every care region $\langle e_1, e_2, p, v\rangle$, $W_i(p) = v$ for all $I(e_1) < i \leq I(e_2)$, and
- For every timing constraint $\langle e_1, e_2, l, u\rangle \in M$, $l \leq I(e_2) - I(e_1) \leq u$.

I is *minimal* iff for each event $e \in E$, $I(e)$ is the smallest index into W that satisfies e and occurs after all indices assigned to events that must precede E (by the partial order induced by M).

Definition 4 Let D be a timing diagram and let W be a word. $W \models D$ if there exists a minimal and valid index assignment I for D and W. The set of all such words forms the *language* of D (denoted $\mathcal{L}(D)$).

[1] A numbering scheme could distinguish syntactically similar events.

The semantics captures one occurrence of a timing diagram, rather than the multiple occurrences needed to treat a timing diagram as an invariant. The one-occurrence semantics provides a foundation for defining different multiple-occurrence semantics [5] and enables efficient complementation of timing diagrams [6].

3 Translating Timing Diagrams to LTL

Formulas of linear temporal logics describe computations on infinite paths where each state is labeled with a subset of atomic propositions AP that are true in that state. For a computation $\pi = w_0, w_1, \ldots$ and $i \geq 0$, let π^i be the computation π starting at the state w_i. In particular, $\pi^0 = \pi$. We use $\pi, i \models \varphi$ to indicate that a formula φ holds in a computation π with w_i taken as a start position. The relation $\models$ is inductively defined for each of the logics we define in this paper.

LTL Given a set AP of atomic propositions, the LTL formulas over AP are:

- **true**, **false**, p, or $\neg p$, for $p \in AP$,
- $\neg\psi$ or $\psi \vee \varphi$, where ψ and φ are LTL formulas, or
- $G\psi$, $X\psi$, or $\psi U \varphi$, where ψ and φ are LTL formulas.

The temporal operators G ("always"), X ("next") and U ("until") describe time-dependent events. F ("eventually") abbreviates $\mathbf{true}U$.

For LTL, $\pi, i \models \varphi$ is equivalent to $\pi^i, 0 \models \varphi$, since formulas in LTL are not concerned with past events. We use $\pi \models \varphi$ as a shorthand for $\pi, 0 \models \varphi$.

- For all paths π, $\pi \models$ **true** and $\pi \not\models$ **false**.
- For an atomic proposition $p \in AP$, $\pi \models p$ iff $p \in L(w_0)$.
- $\pi \models \neg\psi$ iff $\pi \not\models \psi$.
- $\pi \models \psi \vee \varphi$ iff $\pi \models \psi$ or $\pi \models \varphi$.
- $\pi \models \psi \wedge \varphi$ iff $\pi \models \psi$ and $\pi \models \varphi$.
- $\pi \models G\psi$ iff for all $i \geq 0$, $\pi^i \models \psi$.
- $\pi \models X\psi$ iff $\pi^1 \models \varphi$.
- $\pi \models \psi U \varphi$ iff there exists $i \geq 0$ such that $\pi^i \models \varphi$ and for all $j < i$, $\pi^j \models \psi$.

The rest of this section presents a translation from timing diagrams with partial orders on events and don't-care regions into LTL. Previous work [6] translated timing diagrams with don't-care regions and total orders on events to LTL. We could reuse the prior algorithm by enumerating all the total orders corresponding to the partial order and logically disjoining the result of converting each totally-ordered diagram to LTL. This approach has the obvious drawback of potentially requiring exponentially many disjuncts in the translated formula. We therefore wish to consider alternate approaches.

Amla *et al.* translate timing diagrams with partial orders but no don't-care regions to *universal finite automata (∀FA)* [1]. ∀FA differ from standard NFAs in accepting those words on which *all* possible runs (rather than some run) through the automaton end in a final state. Amla *et al*'s automata spawn one run for each event in the diagram, as well as one run for each waveform in the diagram.

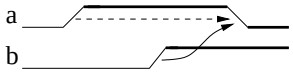


Fig. 2. Timing diagram motivating the need for an end marker.

Since proper handling of partial orders is new for translations to LTL, we first focus on this issue, returning afterwards to include don't cares. For partial orders, we use a similar idea to Amla *et al*'s: we construct a formula for each event and timing constraint in the diagram, and then conjoin these formulas into one LTL formula. The translation to LTL, however, is more difficult because LTL is inherently defined on infinite words. To see why this is a problem, consider the diagram in Figure 2 (the arrow from the rising to falling transition on a is dashed to indicate that it is implicit from the waveform). Clearly, the formula must locate the rise and fall of a and the rise of b and capture the ordering constraint (that the fall of a follows the rise of b).

We want a formula that is polynomial in the size of the diagram. Writing formulas to capture the individual waveform shapes is easy but capturing the ordering constraint is not. We cannot capture the waveforms together with the ordering constraint in one pass through the diagram due to the unspecified order between the fall of a and the rise of b. Writing one formula to capture that the fall of a follows the rise of a and another formula to capture that the rise of b follows the rise of a also doesn't work because both formulas must locate *the same fall of* a. Separate constraints would accept the word $w = (\overline{a}\overline{b}) \cdot (a\overline{b}) \cdot (\overline{a}\overline{b}) \cdot (a\overline{b}) \cdot (ab) \cdot (\overline{a}b)^{\omega}$ which does not satisfy the diagram (where $\overline{p}$ stands for $\neg p$). To align the searches for events, we will use LTL augmented with existential quantification over atomic propositions to introduce a special symbol called end into the word to mark the end of the diagram.[2] This problem does not exist in ∀FA, since all copies of the automaton must finish in an accepting state at the same time. Thus, in some sense, the end marker is implicitly present in ∀FA.

Returning to capturing timing constraints, assume we want to define an LTL formula $\varphi(a, i)$ that is true at the i^{th} transition on a, which happens to be a rise. Assume that n_a is the number of transitions on a, $finish(a)$ is the literal for the final value of a in the diagram, and that we have a proposition end identifying the end of the diagram. Then

$$\varphi(a, i) = \neg a \wedge X(aU(\neg aU(\ldots U(finish(a)Uend)\ldots))),$$

where the number of Uoperators is $n - i - 1$. Intuitively, the formula first describes whether the transition is a rise or a fall (a fall would begin with $a \wedge X\neg a\, U$...), then captures the rest of the waveform as the second argument to the first U.

Using such formulas, the formula for a whole timing constraint $\langle a_i, b_j, l, u\rangle$, where a is the i^{th} transition on a and b_j is the j^{th} transition on b, and the transition on b

[2] In general, adding quantification over atomic propositions increases the expressive power of temporal logics [12–14]. The restricted version that we use here does not add expressiveness, however, as all formulas that are created from timing diagrams can be translated to equivalent formulas in LTL with both past and future modalities using Gabbay *et al*'s work [7, 8].

happens within the $[l, u]$-interval after the one on a, is captured by $\xi(a_i, b_j, l, u)$, where

$$\xi(a_i, b_j, l, u) = F(\varphi(a, i) \wedge (aU\varphi(b, j)))$$

if a rises at i, and

$$\xi(a_i, b_j, l, u) = F(\varphi(a, i) \wedge (\neg aU\varphi(b, j)))$$

if a falls at i. If the timing constraint has time bounds (say $[l, u]$), then we replace $aU\varphi(b, j)$ with $\bigvee_{k=l}^{u} \psi \wedge X\psi \wedge \ldots \wedge X^{k-2}\psi \wedge X^{k-1}\varphi$, where X^m stands for m nested X operators, and l and u are natural numbers. Let $\Xi(D)$ be the set of all formulas $\xi(a_i, b_j, l, u)$, for all timing constraints in D.

For synchronization lines, the formulas that capture the fact that the i^{th} transition on a happens simultaneously with the j^{th} transition on b are

$$\gamma(a_i, b_j) = F(\varphi(a, i) \wedge \varphi(b, j)).$$

Let $\Gamma(D)$ be the set of all formulas $\gamma(a_i, b_j)$ for all synchronization events.

Timing diagrams contain implicit ordering arrows between each pair of consecutive events on a single waveform. Rather than encode these through ξ, we create a single formula that precisely captures the shape of its waveform. For a signal a, let $start(a)$ be the literal for the initial value of a in the diagram (either a or $\neg a$), and let $finish(a)$ be the literal that corresponds to the final value of a in the diagram. The formula ψ_a that describes the waveform of a is

$$\psi(a) = start(a)U(\neg start(a)U(start(a)U \ldots U(finish(a)Uend)\ldots) \qquad (1)$$

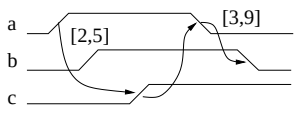
where the number of Uoperators equals the number of transitions on a in the diagram.

Finally, we combine all these formulas into a formula $\theta(D)$ that describes the language of the timing diagram D. The formula states that a word w belongs to the language of D iff there exists a position r in w such that when $w[r]$ is labeled with end, the word can be mapped to D. The formula $\theta(D)$ is as follows.

$$\theta(D) = \exists! end : \bigwedge_{a \in AP} \psi_a \wedge \bigwedge \Xi(D) \wedge \bigwedge \Gamma(D)), \qquad (2)$$

where $\exists! end$ means that exactly one position is labeled with end.

Example 1. As an illustration, consider a diagram D and its waveform formulas:



$$\psi(a) = \neg aU(aU(\neg aUend))$$
$$\psi(b) = \neg bU(bU(\neg bUend))$$
$$\psi(c) = \neg cU(cUend)$$

The arrows connect the rise of a with the rise of c, the rise of c with the fall of a, and the fall of a with the fall of b. The rise of a is characterized by the formula $\varphi(a, 1) = \neg a \wedge X(aU(\neg aUend))$, and similarly for other transitions. The timing constraints are

$$\xi(a_1, c_1, 2, 5) = \varphi(a, 1) U_{[2,5]} \varphi(c, 1)$$
$$\xi(c_1, a_2, 1, \infty) = \varphi(c, 1) U \varphi(a, 2)$$
$$\xi(a_2, b_2, 3, 9) = \varphi(a, 2) U_{[3,9]} \varphi(b, 2)$$

Finally, the formula $\theta(D)$ is

$$\exists! end : \psi(a) \wedge \psi(b) \wedge \psi(c) \wedge \xi(a_1, c_1, 2, 5) \wedge \xi(c_1, a_2, 1, \infty) \wedge \xi(a_2, b_2, 3, 9)).$$

Observation 1 (Complexity of $\theta(D)$) The formula $\theta(D)$ is polynomial in the size of the diagram D. Let D be a timing diagram of size n. The number of waveform formulas $\psi(a)$ is equal to the number of signals in D. The size of a waveform formula $\psi(a)$ is linear in the number of transitions, thus is $O(n)$. Since $\varphi(a)$ is a subformula of $\psi(a)$, we have that $|\varphi(a)| = O(n)$. The number of events in D is bounded by n. Therefore, the total size of $\theta(D)$ is bounded by $O(n^2)$.

Observation 2 (Adding Don't Cares) The ξ and γ formulas capture the diagram's constraints *under the assumption that the i^{th} transition as identified from the end of the diagram is the i^{th} transition from the beginning of the diagram*. This assumption may be false in the presence of don't-care regions; the end marker does not help because it isn't clear which occurrences of events should count. Handling both partial orders and don't cares seems to require enumerating the total orders for the partial order, which yields a formula of a (possibly) exponential complexity in the size of the diagram.

4 Cleanly Capturing Diagrams through Textual Logics

The previous section shows the complex structure of an LTL formula that captures a timing diagram with partial orders and don't cares. Some of this complexity arises from using separate subformulas for waveforms and timing constraints, which is needed to capture partial orders on events. The diagram in Figure 2 illustrates a core incompatibility between timing diagrams and LTL: LTL cannot cleanly capture separate paths converging on a future event while timing diagrams express this naturally.

This problem suggests that timing diagrams rely on a different set of temporal abstractions than those provided by the LTL operators. This raises an interesting question: how fundamental are these differences? Visually, timing diagrams define (potentially overlapping) windows that are bounded by events and contain other events and windows. In LTL, $[\phi U \psi]$ defines a window bounded on the left by the current position in a word and bounded on the right by positions satisfying ψ. Since the occurrence of ϕ can extend beyond that of ψ (if ψ were Fp, for example), LTL also supports some degree of overlapping windows. The future-time nature of LTL biases window locations towards future positions in a word, however, and leads to blowup when windows align on the right boundaries. Past-time temporal operators, however, could capture windows that align on right boundaries. Our question is *whether operators that fix window boundaries on one end of the word are rich enough to capture the window structure in timing diagrams while using only one term for each event in the diagram*.

The restriction to one term per event is important because it is a distinguishing feature of timing diagrams. We are trying to understand the differences between textual

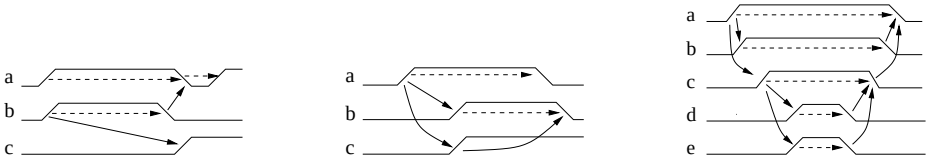


Fig. 3. Timing diagrams with various shapes of partial order.

temporal logics and timing diagrams from a logical perspective. In this work, we hold formulas to the one-term requirement and study the restrictions that this in turn places on the semantics of the operators. For the rest of this paper, we consider timing diagrams with ordering between events but no timing constraints on those orders. Such diagrams still blow up when translated to LTL but allow us to focus on the fundamental question of how well LTL-like logics capture the partial orders allowed in timing diagrams. We begin with several examples of timing diagrams with different shapes of partial orders and discuss which textual temporal logic operators capture them cleanly.

Tree-Shaped Partial Orders Two observations arise from the diagram in Figure 2. First, we could cleanly capture the diagram in LTL if the common event lay at the beginning of the partial order (i.e., if the orderings on events were reversed): we would locate the first event, then independently locate the remaining two events. Second (though related), this diagram appears easier to capture if we use past-time operators: we could locate the falling transition on a that is common to both chains of events, then look *backwards* to find the (now independent) rising transitions on a and b. These observations give rise to our first two claims about partial orders and clean temporal logic formulas: partial orders that form trees (or forests) can be cleanly translated into LTL, while those that form trees with their edges reversed can be cleanly translated into Past-LTL. Note that Past-LTL here means LTL with *only* past-time temporal operators. We will use PLTL to mean the temporal logic with both future time and past-time operators.

Partial Orders with Multiple Minimal Events The leftmost diagram in Figure 3 has multiple minimal events. A formula capturing this diagram cannot start simultaneous (i.e. conjoined) searches from the rising transitions on a and b because those searches converge on the falling transition on a. Using both past and future-time operators, however, a formula could search for the first rising transition on a followed by the falling transition on a; at that point, the search could split into two independent searches: one forward for the second rise on a, and another backward for the transitions on b followed by a forward search for the transition on c. All of the edges in the partial order are edges in the tree specifying this search, but some of those edges are reversed in the tree. Intuitively, this criterion characterizes when a search can be captured in PLTL.

Diamond-Shaped DAGs The events in the middle diagram in Figure 3 form a diamond (between the rising transition on a and the falling transition on b). If a formula searches for the rising transition on a first, it cannot then spawn independent searches for the rising transitions on b and c (the "bulge" of the diamond) because they must meet up

again at the falling transition on b. Searching first for the falling transition in b causes similar problems. We could conduct this search cleanly if we had a way to "remember" the location of the rising transition on a, then search forwards for the falling transition on b, then backwards for the rising transitions on b and c, but with these last two searches bounded by the remembered position of the search on a.

Laroussinie, Mackey, and Schoebelen's linear temporal logic with forgettable past (NLTL) does exactly this. It adds an operator N to PLTL that restricts the path to the suffix starting from the position at which the N is encountered. Using N, we could capture the diamond pattern roughly as $FN(a\uparrow \wedge F(b\uparrow \wedge F(b\downarrow \wedge P(c\uparrow))))$. The N prevents the backwards search for $c\uparrow$ from going beyond the location of $a\uparrow$.

The rightmost diagram in Figure 3 contains one diamond pattern nested inside another. This diagram is hard to capture cleanly using just NLTL because both prefixes and suffixes must be truncated during the search. We therefore introduce an analogous operator to N, called $\tilde{N}$, that limits the scope of a search to a prefix of the path.

The following subsections formalize our observations about the temporal operators needed to cleanly capture various shapes of partial orders. We define the logics PLTL and NÑLTL and present an algorithm for translating a subset of partial orders into formulas in NÑLTL. We prove that the translation is correct and show a richer partial order that NÑLTL cannot capture. Characterizations of the partial orders captured by LTL, PLTL, and Past-LTL follow from the correctness of the translation algorithm.

4.1 The Logics

PLTL The logic PLTL (LTL+Past) is the logic LTL extended with past time modalities: Y ("yesterday") is the opposite of X, that is, it denotes an event that happened in the previous step; P ("past") is the opposite of F, that is, it denotes an event that happened somewhere in the past; and S ("since") is the opposite of U. We refer to Y, P, and S as *past modalities*, and to X, U, F, and G as *future modalities*. The semantics for the past modalities is as follows.

- $\pi, i \models Y\psi$ iff $i > 0$ and $\pi, i-1 \models \psi$.
- $\pi, i \models \psi S \varphi$ iff $\pi, j \models \varphi$ for some $0 \leq j \leq i$ such that $\pi, k \models \psi$ for all $j < k \leq i$.

We use P as a shortcut for $\mathbf{true}S$.

The N and $\tilde{N}$ modalities The logic NLTL (*LTL with forgettable past*) is defined by extending the logic PLTL with the unary modality N [10]. The semantics of N is defined as follows: $N\varphi$ is satisfied in the i^{th} position of a path π iff φ is satisfied in the path $\rho = \pi^i$. In other words, N ignores everything that happened in π prior to the position i. Formally, $\pi, i \models N\varphi$ iff $\pi^i, o \models \varphi$.

NÑLTL includes N and a similar modality for the *unforeseeable future*. The unforeseeable future modality ("up to now") is denoted by $\tilde{N}$. Semantically, $\tilde{N}\varphi$ is satisfied in the i^{th} position of a path π iff φ is satisfied in the last position of the finite path ρ obtained from π by cutting off its suffix π^{i+1}. That is, $\rho = \pi[0..i]$.

Gabbay [7, 8] proved that any linear-time temporal property expressed using past-time modalities can be translated into an equivalent (when evaluated at the beginning

Gen-Formula(P)
if P contains multiple connected components $P_1, \ldots P_k$ **return** $\bigwedge_{i=1}^{i=k}$ Gen-Formula(P_i)
elseif P has a valid schedule tree T and no dividing events **return** Gen-Tree(P, T, root(T))
else let $e_1, \ldots, e_n$ be a sequence of dividing events for P
return $\varphi_{e_1} \wedge \tilde{N}$(Gen-Formula($R_0$))
$\wedge XNF(\varphi_{e_2} \wedge \tilde{N}$ (Gen-Formula(R_1))
$\wedge XNF(\varphi_{e_3} \wedge \ldots \wedge XNF(\varphi_{e_n} \wedge \tilde{N}$ (Gen-Formula(R_{n-1}))
$\wedge XNF$(Gen-Formula(R_n))$\ldots$)

Gen-Tree(P,T,e)
if e has no successors **return** φ_e
else let $en_1, \ldots, en_k$ be successors of e in T in same direction as in P
let $ep_1, \ldots, ep_j$ be successors of e in T in opposite direction as in P
return $\varphi_e \wedge \bigwedge_{i=1}^{k}(XF$Gen-Tree$(P, T, en)) \wedge \bigwedge_{i=1}^{j}(P$Gen-Tree$(P, T, ep))$

Fig. 4. The formula generation algorithm. φ_e denotes the formula that captures an event e: $\neg a \wedge X(a)$ captures $a\uparrow$ and $a \wedge X(\neg a)$ captures $a\downarrow$.

of the path) pure future formula. In other words, PLTL is not more expressive than LTL. Gabbay's proof can be extended to NLTL as well [10]. Since the modality $\tilde{N}$ is symmetrical to N; the same proof applies to N$\tilde{\text{N}}$LTL. Gabbay's translation yields a formula whose size is assumed to be non-elementary in the size of the initial formula. It was recently proved that PLTL is at least exponentially more succinct than LTL, and that NLTL is at least exponentially more succinct than PLTL [9]. It is easy to see that the proof of exponential gap in succinctness can be used almost without change for $\tilde{N}$. That is, introducing either N or $\tilde{N}$ is enough for the exponential gap. Observe that in general, chopping off the prefix is not equivalent to chopping off the suffix, since the former leaves us with an infinite path, while the latter preserves only a finite portion of the path. However, Laroussinie *et al.*'s proof uses only propositional formulas. The same proof therefore works if we reverse the direction of the input, switch past and future modalities in formulas and use $\tilde{N}$ instead of N. While using both N and $\tilde{N}$ modalities proves to be helpful in translating timing diagrams, it seems that having both of them does not introduce an additional gap in succinctness as opposed to having only one. That is, the logic N$\tilde{\text{N}}$LTL seems to be no more succinct than NLTL.

4.2 Compiling Partial Orders to N$\tilde{\text{N}}$LTL

The diagrams in Figure 3 illustrate how the events in timing diagrams bound windows in which other events must occur. Our discussion illustrates how the N and $\tilde{N}$ operators are useful for enforcing these boundaries in temporal logic. Our translation strategy is to use N and $\tilde{N}$ to (recursively) scope the window boundaries at the outermost levels and to use PLTL to capture the events that fall within these windows. This approach works when the contents of windows form forests (or can be further decomposed into

subwindows). We limit the algorithm to these cases, then show a richer structure that NÑLTL is not capable of capturing.

The algorithm appears in Figure 4. It finds window boundaries by locating *dividing events* in the partial order.

Definition 5 An event e of a diagram D is a *dividing event* iff there exists a partition of the events of D minus e into sets E_1 and E_2 such that $e \succ e_1$ for all $e_1 \in E_1$ and $e \prec e_2$ for all $e_2 \in E_2$. Given a sequence of dividing events, $e_1 \prec e_2 \prec \ldots \prec e_n$, the *region* R_i is the set of events e such that $e_i \prec e \prec e_{i+1}$ (with R_0 containing those events that precede e_1 and R_n containing those events that follow e_n).

By definition, the dividing events are totally ordered and thus can be captured by a sequence of nested LTL F operators. All remaining events fall into regions encapsulated by the dividing events. Our translation algorithm bounds these regions in the formula by inserting N and $\tilde{N}$ at the beginning and the end of regions.

If a partial order contains no dividing events, then each connected component within the partial order is compiled to a formula in PLTL. This translation relies on a *schedule tree* that specifies the order in which to search for each event in the component.

Definition 6 Given a partial order P, a *schedule tree* T of P is a tree with directed edges such that the set of nodes in T is the set of nodes in P. We call a schedule tree *valid* if for each edge $e_1 \rightarrow e_2$ in T, P contains either an edge from e_1 to e_2, or an edge from e_2 to e_1. In other words, all edges in T must be justified by edges in P, but the edges in T may occur in the reversed direction from those in P.

Note that a single partial order could have several schedule trees. As a simple example, the partial order $a \prec b$, $b \prec c$, and $a \prec c$ could schedule b or c in either order.

Definition 7 Given a partial order P, a *schedule forest* F of P is a set of trees with directed edges such that the set of nodes in F is the set of nodes in P and each tree in F is a schedule tree for its subset of nodes.

The following theorem establishes that the result of Gen-Formula(P) is a formula that recognizes all valid instances of P.

Theorem 3. *Let D be a diagram with partial order P. Let $\mathcal{R}$ be the set of regions (sub-orders) between each pair of subsequent dividing events of P. Then, if the partial order for each region $R \in \mathcal{R}$ can be translated to a schedule forest, Gen-Formula(P) is of size $O(|P|)$ and defines the same language as D.*

Proof. We argue the size and correctness claims from the theorem separately, starting with size. Formulas describing individual events are constant in size. Each dividing event is included in the formula one time (in the final **return** of Gen-Formula). The formula returned from Gen-Tree has linear size in the number of events in the tree by construction. Each non-dividing event appears in at most one schedule tree, so the formula resulting from Gen-Formula(P) has linear size in the number of events in P.

For the correctness claim, we will prove that Gen-Formula(P) requires exactly those event orderings contained in P. Let φ be the result of Gen-Formula(P). We first show that the formula enforces every ordering edge in P. Let $e \rightarrow e'$ be an edge in P. One of the following cases must hold:

– e is a dividing event in some portion of P. Then φ contains the subformula $\varphi_e \wedge \phi \wedge XN\psi$ where the term for e' occurs in ψ. The use of N ensures that e occurs no later than e' and the use of X ensures that e occurs strictly later than e'.
– e' is a dividing event in some portion of P. Then e must lie in the region preceding e' (unless e was a dividing event, which the previous case covered). φ contains the subformula $\varphi_{e'} \wedge \tilde{N}(R)$, where R is the region containing e. The $\tilde{N}$ ensures that e occurs before e' (no X is needed here because $\tilde{N}$ cuts off the future between the two halves of e' while two positions are needed to capture both halves of e).
– Neither e nor e' is a dividing event, which means the subformula containing their terms was generated by a call to Gen-Tree on some schedule tree T over a portion of P. By the definition of valid schedule trees, T must contain an edge between e and e' (in one direction or the other). If $e \rightarrow e'$ was an edge in T, then the XF in the output of Gen-Tree ensures that e occurs before e'. If $e' \rightarrow e$ was an edge in T, then the P in the output of Gen-Tree ensures that e occurs before e'.

We now argue that φ does not require any event orders beyond those in P. Let e_1 and e_2 be unordered events (directly or transitively) in P. Since e_1 and e_2 are unordered, no dividing event can occur between them, so they must lie in the same region R of the diagram. There are two possible cases:

– e_1 and e_2 are in different connected components of R. Gen-Formula connects separate components only through $\wedge$, which induces no temporal ordering, so φ respects the lack of order between e_1 and e_2.
– e_1 and e_2 are in the same connected component of R. In this case, both events will be in the schedule tree T for their enclosing component. If e_1 and e_2 are in different branches within T, φ relates them only by $\wedge$ and thus respects their lack of ordering. Assume that e_1 and e_2 are on a common branch, and assume without loss of generality that e_1 is closer to the root of T than is e_2. The path from e_1 to e_2 must contain at least one edge that occurs in the same direction as in P and one that occurs in the opposite direction as in P (otherwise, P would order e_1 and e_2). This means that both an F and a P operator will separate e_1 and e_2 in φ. This allows e_2 to occur on either side of e_1 (no N or $\tilde{N}$ operator can intervene because those are dropped only at dividing events), so the theorem holds. □

Lemma 1. *If the directed graph induced by partial order P forms a tree, then there exists a formula in LTL that recognizes P.*

Proof. In this case, P is a valid schedule tree of itself. Gen-Tree uses only future time operators when the edges in the schedule tree match the edge direction in P.

Lemma 2. *If the directed graph constructed by reversing every edge in a partial order P forms a tree, then there exists a formula in Past-LTL that recognizes P.*

Proof. Follows by similar argument as for Lemma 1. The expressions for events that use the X operator can be rewritten to use Y.

Lemma 3. *If the graph induced by partial order P is a tree with multiple minimal events, then there exists a formula in PLTL that captures P.*

Proof. True by the definition of Gen-Tree since such a P has a valid schedule tree.

Note that don't cares are handled implicitly by not being present as events in P. The care regions are present in P as separate events for each care position, and the complexity in this case would also depend on the unary representation of the length of the care region.

4.3 Limitations of NÑLTL

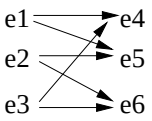
The Gen-Formula algorithm uses the F and P operators only to search for events. These uses all take the form $F/P(e \wedge \psi)$, where e is an event and ψ is an NÑLTL formula. We call these forms of F and P *search operators*. NÑLTL restricted to search operators cannot capture all timing diagrams using only one term per event. Consider the following diagram D_{bad}, which has no dividing events and as many arrows as it does events (which precludes a schedule tree). The partial order over events in this diagram appears on the right. The rise and subsequent fall of a correspond to the events e_1 and e_2, and the rise and subsequent fall of b correspond to the events e_4 and e_3, respectively.

D_{bad} is expressible in NÑLTL as $F(F(e_2) \wedge F(e_3) \wedge P(e_1) \wedge P(e_4))$. The outermost F locates an index lying within the windows on a and b from which to start the search; it effectively performs existential quantification on the starting position. This is a rather different use of F from the search operators. The following lemma argues that D_{bad}'s partial order cannot be expressed concisely in NÑLTL using only search operators.

Lemma 4. *No NÑLTL formula restricted to search operators captures D_{bad} with exactly one term per event.*

Proof (sketch). Any NÑLTL formula that captures D_{bad} using only search operators must search for the events in some tree-shaped order (corresponding to the order in the formula's parse tree). The restriction that each event appear once in the formula allows the proof to exhaustively consider each order of the four events. Intuitively, N and $\tilde{N}$ cannot be used since there is no common minimal or maximal event: dropping an N marker at e_1 or e_4 before the other has been located would induce an unwanted ordering between these events; a similar problem with $\tilde{N}$ governs e_2 and e_3. Therefore, no event can serve as a valid starting point for the search embodied in the parse tree. None of the remaining operators help encode constraints in which one node is incident on two others. Any attempt to construct a formula that captures all orderings therefore either omits an arrow or imposes an order that does not exist in the diagram. □

D_{bad} has the simplest partial order with n events and n non-transitive edges forming an undirected cycle. Larger orders of this form cannot be expressed in NÑLTL even by using F to search for a starting position. The analogous formula for the order on the right, for example, would induce unwanted constraints on unrelated events (such as e_2 and e_4).[a] We defer the proof to a full paper.



[a] Thanks to Shriram Krishnamurthi for suggesting this example.

Note that while the partial order in D_{bad} is diamond shaped, the direction of the arrows is different from the expressible diamond shape in the middle diagram from Figure 3. The diamond in the Figure 3 diagram has one minimal and one maximal event, while D_{bad} has two minimal and two maximal events. Multiple extremal events (and their common incident events) are at the heart of the argument for Lemma 4 because they preclude dividing events that decompose the search.

5 Conclusions and Future Work

LTL is often viewed as the canonical linear temporal logic for formal specification. Although newer logics such as PSL and OVA challenge this view in practical settings, LTL is still a benchmark logic for theoretical verification research. Timing diagrams are often viewed as just a pretty interface for LTL. This paper questions that view by illustrating the distance between the LTL operators and the temporal abstractions that timing diagrams express so naturally. Our translation from timing diagrams to LTL—the first to handle both partial event orders and don't-care regions—illustrates the gap, while our results on concisely capturing various shapes of partial orders in temporal logics put the mismatch in a formal context.

Perspective Our results relating partial orders to temporal logics serve two purposes. First, they suggest temporal abstractions that we would need to be able to recognize in textual formulas (in any logic) in order to render specifications as timing diagrams. Translating textual specifications to diagrams is attractive as an aid for understanding and debugging complex specifications. One interpretation of our work is that we have partly reduced the problem of recognizing that a formula can be drawn as a diagram to one of recognizing when an LTL formula captures a (more compact) NÑLTL formula. Our work suggests that recognizing diagrams in formulas might not make sense for LTL, as we do not expect designers would ever write LTL formulas as complicated as our translations of timing diagrams. Rendering diagrams that *approximate* temporal specifications may be a more realistic approach that would still benefit from our observations.

Second, our results suggest than a good textual analog to timing diagrams needs a different semantic philosophy than LTL. LTL and its extensions break paths into windows in which subformulas are satisfied, but these windows are strictly bounded at one end. This characteristic captures nested windows and windows that are ordered on one side, but not windows that overlap one another with constraints on both ends. Timing diagrams capture these richer spatial operators between windows. The inexpressible diagram in Section 4.3 provides an example of complex constraints between windows that do not fit within the styles of operators traditional in LTL extensions.

Future Work Characterizing the class of diagrams for which there are no equivalent NÑLTL formulas of the same size remains an open problem. Section 4.3 presents initial steps in this direction. Given such a characterization, we must confirm that our algorithm handles all expressible partial orders other than D_{bad}. We have not yet considered the impact of timing constraints on our conciseness results. Logics such as PSL in which

windows in words are an explicit part of the semantics may provide a better textual analog for timing diagrams. We intend to perform a similar analysis to the one in this paper for PSL. This exercise should give a different perspective on the temporal operators that are fundamental to timing diagrams yet natural to capture textually. If existing windows-based logics also prove insufficient for cleanly capturing timing-diagram-like specifications, developing native verification techniques for timing diagrams may well prove beneficial. Similar comparisons to interval-based temporal logics would also be instructive [4, 11]. Finally, it would be useful to understand the shapes of partial orders that designers frequently express in timing diagrams in practice. While we have seen examples that require rich partial orders, we lack more detailed data about the frequency of each of these shapes in practice.

Acknowledgments: Research funded by NSF grants CCR-0132659 and CCR-0305834.

References

1. N. Amla, E. A. Emerson, and K. S. Namjoshi. Efficient decompositional model checking for regular timing diagrams. In *IFIP Conference on Correct Hardware Design and Verification Methods*, 1999.
2. N. Amla, E. A. Emerson, K. S. Namjoshi, and R. J. Trefler. Visual specifications for modular reasoning about asynchronous systems. In *International Conference on Formal Techniques for Networked and Distributed Systems*, pages 226–242, 2002.
3. E. Cerny, B. Berkane, P. Girodias, and K. Khordoc. *Hierarchical Annotated Action Diagrams*. Kluwer Academic Publishers, 1998.
4. L. Dillon, G. Kutty, L. Moser, P. M. Melliar-Smith, and Y. S. Ramakrishna. A graphical interval logic for specifying concurrent systems. *ACM Transactions on Software Engineering and Methodology*, 3(2):131–165, Apr. 1994.
5. K. Fisler. Timing diagrams: Formalization and algorithmic verification. *Journal of Logic, Language, and Information*, 8:323–361, 1999.
6. K. Fisler. On tableau constructions for timing diagrams. In *NASA Langley Formal Methods Workshop*, 2000.
7. D. Gabbay. The declarative past and imperative future. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 407–448. Springer-Verlag, 1987.
8. D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. 7th ACM Symp. on Principles of Programming Languages*, pages 163–173, January 1980.
9. F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *Proc. 17th IEEE Symp. Logic in Computer Science (LICS'2002)*, pages 383–392, 2002.
10. F. Laroussinie and P. Schnoebelen. A hierarchy of temporal logics with past. In *Proc. 11th Symp. on Theoretical Aspects of Computer Science*, Caen, February 1994.
11. B. Moszkowski. A temporal logic for multi-level reasoning about hardware. *IEEE Computer*, pages 10–19, February 1985.
12. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, Austin, January 1989.
13. A. Sistla. *Theoretical issues in the design of distributed and concurrent systems*. PhD thesis, Harvard University, Cambridge, MA, 1983.
14. A. Sistla, M. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.