Technical Report CS-TR-2012-11: Monitoring Dense-Time, Continuous-Semantics, Metric Temporal Logic

Forward This document constitutes the 'full version' of the paper [1] that appeared in Runtime Verification 2012. It differs by the addition of appendices that provide proofs of the correctness and complexity.

There are also a number of corrections to the published version – most significantly a correction to the definition of the *until* and *since* operators. The published version of the paper implemented a definition of *since* that required that $\phi \mathcal{S} \psi \rightarrow \phi$ and neglected the requirement that $\psi \rightarrow \phi \mathcal{S} \psi$. Correspondingly, the transducer tables for *until* implemented $\phi \mathcal{U} \psi \rightarrow \phi$ and failed to implement $\psi \rightarrow \phi \mathcal{U} \psi$. The previously-implemented definitions may be valuable for the continuous semantics of metric temporal logic on dense time; The author's preference for the continuous semantics rests on an assertion that they are more intuitive for dense time and a similar argument might be made for the previous definitions of *until* and *since*. However, operators implementing those definitions would be nothing more than a syntactic convenience since they are identical to $\phi \wedge \phi \mathcal{U} \psi$ and $\phi \wedge \phi \mathcal{S} \psi$, respectively, so their use would only serve to confuse the reader.

The treatment of the *eventually* operator from the original paper employed an initial-transition table that was only sensible in the case of the so-called instantaneous version (for which the maximum extent of its interval was zero). This is unneeded given that the ENQUEUE procedure ensures that only valid boolean signals result – regardless of the direction of the time shift of the output events – and so it has been removed. Transducer algorithms omitted from the published paper in the interest of brevity are reintroduced and the order of presentation of the transducer algorithms and tables has changed in the interest of clarity.

Monitoring Dense-Time, Continuous-Semantics, Metric Temporal Logic

Kevin Baldor[1,2] and Jianwei Niu[1]

[1] University of Texas at San Antonio, USA
{kbaldor,niu}@cs.utsa.edu
[2] Southwest Research Institute, San Antonio, USA

Abstract. The continuous semantics and dense time model most closely model the intuitive meaning of properties specified in metric temporal logic (MTL). To date, monitoring algorithms for MTL with dense time and continuous semantics lacked the simplicity the standard algorithms for discrete time and pointwise semantics. In this paper, we present a novel, transition-based, representation of dense-time boolean signals that lends itself to the construction of efficient monitors for safety properties defined in metric temporal logic with continuous semantics. Using this representation, we present a simple lookup-table-based algorithm for monitoring formulas consisting of arbitrarily nested MTL operators. We examine computational and space complexity of this monitoring algorithm for the past-only, restricted-future, and unrestricted-future temporal operators.

1 Introduction

Program monitoring has attracted interest as an alternative to model checking or theorem proving as these become impractical due to the size of the state space of a full program. A dynamic analysis, monitoring is limited to the detection of property violations that are actually observed in a program's execution, and more fundamentally, to so called *safety properties*, those that can be falsified given only a finite number of program events.

Temporal logics such as linear temporal logic (LTL) [7] and computation tree logic (CTL) [3] provide an effective formal description for desired or undesired program behavior and are commonly used in monitoring applications. Each of these logics specifies constraints on the order of occurrence of events. For example, they can state that "after event p, event q must take place at some point in the future". This is not an enforceable safety property but would become one if modified to state that q must occur within a certain period of time. To do so, these logics must be augmented with an explicit notion of time.

One such augmentation is metric temporal logic (MTL) [5]. It introduces limits on the periods of time over which a logical connective operates. For example, the property described in the preceding paragraph may be specified as $p \rightarrow \Diamond_{[0,5]}\ q$. The subscript on the *eventually* operator ($\Diamond$), is an interval – relative to the current time – in which q must hold in order for the statement to be true at the current time. Some notations support a number of subscript forms, but without loss of generality, we restrict our presentation to the use of intervals that may be closed or open on either end. Additionally, we admit intervals of the form $[a, a]$, though their use incurs a potential space penalty.

The runtime-verification community employs two time models for MTL: discrete and dense. Within the dense-time model, there are two semantics: point-based and continuous [8] [2]. We

concentrate on the latter, as in [2], Basin et al. assert that "Real-time logics based on a dense, interval-based time model are more natural and general than their counterparts based on a discrete or point-based model". But in it, they present a monitoring algorithm that they describe as "conceptually simpler" for the point-based semantics than for the interval-based (continuous) semantics.

Our contribution with this paper is the introduction of a transition-based – rather than interval-based [2] – representation for the dense-time boolean signals that are a feature of the continuous semantics. With this representation, the output of all MTL connectives can be expressed as a simple lookup-table indexed by the input. Using this representation, we present a conceptually simple MTL monitoring algorithm modeled on the transducer-approach of [6] that reduces to something like the LTL-monitoring algorithm of [4] for past-only operators. We then observe the increase in space complexity of its extension to future MTL expressions.

2 Background

2.1 LTL and MTL

Logical expressions use the connectives for disjunction ($\vee$), *logical or*; conjunction ($\wedge$), *logical and*; and negation ($\neg$) to describe the relationship between logical statements at the current time. Linear Temporal Logic (LTL) augments them with a number of logical connectives that describe the relationship between logical expressions over time.

The past-only operator *historically* ($\boxminus\phi$) indicates that the expression ϕ has been true since time zero, *once* ($\diamondminus\phi$) indicates that ϕ must have been true at some point in time since time zero, and *since* ($\phi\ \mathcal{S}\ \psi$) indicates that at some point in the past ψ must have been true and that ϕ must at least have been true at every point after that until the current time;

The future-only operator *henceforth* ($\Box\phi$) indicates that the expression ϕ is true now and will be true at all points in the future, *eventually* ($\Diamond\phi$) indicates that ϕ must be true at the current time or at some point in the future, and *until* ($\phi\ \mathcal{U}\ \psi$) indicates that at the current time or at some point in the future ψ must be true and that that ϕ must at least have been true at every point between the current time and that point.

The semantics of LTL operate on a trace, a countably infinite sequence of truth values of atomic elements. LTL expressions are only interpreted as having a truth value at the instants of time corresponding to the elements of the trace. Beyond the order of events, the actual time at which the events plays no role in determining the truth of the LTL expressions.

The pointwise semantics of MTL are a natural extension of the semantics of LTL in that, while they augment the order constraints of LTL with true time constraints, the truth of an expression is only defined at discrete points in time. When used for monitoring, an MTL expression might be evaluated only when an input event arrives. This is more efficient than periodically re-evaluating expressions, but can lead to counter-intuitive results. In [2], Basin et al. discuss a number of such results. Perhaps most striking is that under pointwise semantics $\diamondminus_{[0,1]}\diamondminus_{[0,1]}\phi$ is not logically equivalent to $\diamondminus_{[0,2]}\phi$. This is illustrated in the case that ϕ is true at time $\tau = 0$, and the next observation of the system takes place at time $\tau = 2$, $\diamondminus_{[0,2]}\phi$ is *true* at time $\tau = 2$, but $\diamondminus_{[0,1]}\diamondminus_{[0,1]}\phi$ is *false* since the observations lack the 'bridge' at time $\tau = 1$ (for which $\diamondminus_{[0,1]}\phi$ would evaluate to *true*) needed to declare $\diamondminus_{[0,1]}\diamondminus_{[0,1]}\phi$ to be *true* at time $\tau = 2$.

As observed by the authors of [2], adding additional sample points can restore the equivalence of these expressions at the cost of additional computation by the monitor. In a discrete-valued-time

system, this can be taken to the extreme of evaluating all expressions with each 'clock tick'. Beyond the computational cost, this cannot be extended to the dense-time representation that best models external events for which there is no shared clock.

2.2 Continuous Semantics and Boolean Signals

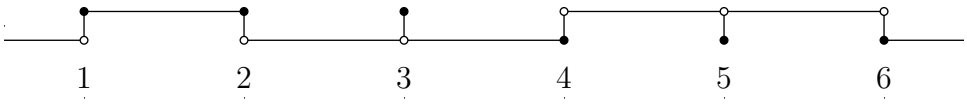
Under continuous semantics, we avoid the ambiguity introduced by the selection of sample points. Loosely, the continuous semantics (MTL) assign a truth value to any expression for any point of time greater than zero. A more formal definition is presented in [2], but we will present enough here for the purpose of discussion. Essentially, the notion of a trace used in LTL and the pointwise semantics of MTL is replaced by a mapping from time $\tau \in \mathbb{R}_{\geq 0}$ to $\{true, false\}$ that the authors term a boolean signal. In their formulation, the boolean signal for the expression ϕ, denoted γ_ϕ, is the set of all points in time for which ϕ evaluates to true. In Figure 1 – expanded from the definitions given in [2] to include $\mathcal{U}_I$, the set of all signals in a model is written $\hat{\gamma}$, $\tau \in \mathbb{R}_{\geq 0}$ denotes the time for which the statement applies, and the subscript I is an interval on $\mathbb{R}$ for which the operator applies.

$$
\begin{array}{lll}
\hat{\gamma}, \tau \models p & \text{iff} & \tau \in \gamma_p \\
\hat{\gamma}, \tau \models \neg\phi & \text{iff} & \hat{\gamma}, \tau \not\models \phi \\
\hat{\gamma}, \tau \models \phi \wedge \psi & \text{iff} & \hat{\gamma}, \tau \models \phi \text{ and } \hat{\gamma}, \tau \models \psi \\
\hat{\gamma}, \tau \models \phi \, \mathcal{S}_I \, \psi & \text{iff} & \exists\, \tau' \in [0, \tau] \text{ such that } \tau - \tau' \in I,\ \hat{\gamma}, \tau' \models \psi, \text{ and } \hat{\gamma}, \kappa \models \phi \ \forall \kappa \in (\tau', \tau] \text{ or}^3 \\
 & & \quad \exists\, \tau'' \in [0, \tau') \text{ such that } \tau - \tau'' \in I,\ \hat{\gamma}, \kappa \models \psi \ \forall \kappa \in [\tau'', \tau') \text{ and } \hat{\gamma}, \kappa \models \phi \ \forall \kappa \in [\tau', \tau] \\
\hat{\gamma}, \tau \models \phi \, \mathcal{U}_I{}^4 \, \psi & \text{iff} & \exists\, \tau' \geq \tau \text{ such that } \tau' - \tau \in I,\ \hat{\gamma}, \tau' \models \psi, \text{ and } \hat{\gamma}, \kappa \models \phi \ \forall \kappa \in [\tau, \tau') \text{ or} \\
 & & \quad \exists\, \tau'' > \tau' \text{ such that } \tau'' - \tau \in I,\ \hat{\gamma}, \kappa \models \psi \ \forall \kappa \in (\tau', \tau''] \text{ and } \hat{\gamma}, \kappa \models \phi \ \forall \kappa \in [\tau, \tau']
\end{array}
$$

Fig. 1. Continuous Semantics of MTL

3 Modeling Dense-Time Boolean Signals as Event Sequences

The dense model of time precludes a representation consisting of every value for which a boolean signal is true as there may be uncountably many such points. However, boolean signals are defined as satisfying the finite-variability condition that on any bounded interval there exists a finite number of non-overlapping intervals over which the signal is *true*. This lends itself to the representation used in [2] an at-most-countably-infinite set of non-overlapping intervals. Our representation differs in that we model boolean signals not as a series of intervals, but as a sequence of timed events denoting a transition from one truth value to another. For example, a signal described with the intervals $\{[1, 2], [3, 3], (4, 5), (5, 6)\}$ might be illustrated as



[3] This clause was added to capture another boundary condition introduced by the dense time model

[4] We introduce the *until* operator using the presentation of [2] to relate the definition of future operators in the standard definitions of the runtime-verification community.

where the higher line indicates *true* and the lower, *false*. The dots at the transition indicate whether the signal is considered *true* at the transition point. We represent this signal as the series of transitions

$$\{(\text{⤒}, 1), (\text{⤓}, 2)\,(\text{⤒}, 3)\,(\text{⤓}, 4), (\text{⤒}, 5)\,(\text{⤓}, 6)\}$$

This example exhausts all of the transition types required to describe boolean signals. In the following sections the additional events $(\_, \tau)$ and $(\overline{\ }, \tau)$ are used to indicate the lack of transition on one of the inputs of a binary operator. They are not strictly required to unambiguously describe a boolean signal, but are convenient for the implementation of the monitor.

Definition 1. *A boolean signal is described by an infinite sequence of timed transitions* (δ_i, τ_i) *for which* $\delta_0 \in \{\_, \overline{\ }, \text{⤒}, \text{⤓}\}$, $\delta_i \in \{\text{⤒}, \text{⤒}, \text{⤒}, \text{⤓}, \text{⤓}, \text{⤓}\} \forall\ i > 0$, *and* $\tau_i \in \mathbb{R}_{\geq 0}$, $\tau_0 = 0$, *and* $\tau_{i+1} > \tau_i \forall i \geq 0$. *The timed transitions are subject to the further constraint types of adjacent transitions must agree in the sense that the incoming value of each transition must match the outgoing value of the previous transition. More formally,*

$$\delta_i \in \{\overline{\ }, \text{⤒}, \text{⤒}, \text{⤒}\} \rightarrow \delta_{i+1} \in \{\text{⤓}, \text{⤓}, \text{⤓}\} \text{ and}$$
$$\delta_i \in \{\_, \text{⤓}, \text{⤓}, \text{⤓}\} \rightarrow \delta_{i+1} \in \{\text{⤒}, \text{⤒}, \text{⤓}\}.$$

Definition 2. *The truth of a signal* γ *at time* τ *is given by*

$$\tau \in \gamma \doteq \begin{cases} \delta_k \in \{\overline{\ }, \text{⤒}, \text{⤓}, \text{⤓}\} & \exists k : \tau_k = \tau \\ \delta_k \in \{\overline{\ }, \text{⤒}, \text{⤒}, \text{⤓}\} & \exists k : \tau_k < \tau \wedge\ (\tau_{k+1} > \tau \vee k = |\gamma|) \end{cases}$$

4 Monitor Construction

4.1 Supported Temporal Operators

Although the timed *until* and *since* are sufficient to capture MTL semantics, the treatment of their transitions is sufficiently complicated that we follow the approach of [6] and introduce timed *eventually* to enable the treatment of only the non-metric *until* and *since*. This is accomplished by exploiting the fact that timed *since* and *until* are redundant given timed *historically* ($\boxminus_I$) , *once* ($\diamondminus_I$), *henceforth* ($\Box_I$), and *future* ($\Diamond_I$) since

$$\phi\ \mathcal{S}_{[a,b]}\psi \leftrightarrow \boxminus_{[0,a]}(\phi\ \mathcal{S}\ \psi) \wedge\ \diamondminus_{[a,b]}\psi \text{ and}$$
$$\phi\ \mathcal{U}_{[a,b]}\psi \leftrightarrow \Box_{[0,a]}(\phi\ \mathcal{U}\ \psi) \wedge\ \Diamond_{[a,b]}\psi$$

and that $\boxminus_I$ and $\Box_I$ are redundant since

$$\boxminus_I \phi \leftrightarrow \neg\diamondminus_I \neg\phi$$
$$\Box_I \phi \leftrightarrow \neg\Diamond_I \neg\phi$$

Further, we observe that for the same input the output of $\diamondsuit_{[a,a+\Delta]}$ and $\diamondsuit_{[b,b+\Delta]}$ are both simply a time-shifted version of the output of $\diamondsuit_{[0,\Delta]}$. By generalizing the intervals to support negative indices, we obtain

$$\diamondsuit_{[a,b]}\phi \leftrightarrow \diamondsuit_{[-b,-a]}\phi$$

As a result, we can monitor both future and past MTL using only the transducers for the operators $\neg$, $\wedge$, $\mathcal{S}$, $\mathcal{U}$, and $\diamondsuit_I$, the formal definitions for which are given in Figure 2.

$$\begin{array}{lll}
\hat{\gamma}, \tau \models \neg\phi & \text{iff} & \hat{\gamma}, \tau \not\models \phi \\
\hat{\gamma}, \tau \models \phi \wedge \psi & \text{iff} & \hat{\gamma}, \tau \models \phi \text{ and } \hat{\gamma}, \tau \models \psi \\
\hat{\gamma}, \tau \models \diamondsuit_I\ \phi & \text{iff} & \exists\ \tau' \text{such that } \tau' - \tau \in\ I,\ \hat{\gamma}, \tau' \models\ \phi \\
\hat{\gamma}, \tau \models \phi\ \mathcal{S}\ \psi & \text{iff} & \exists\ \tau' \in\ [0, \tau] \text{ such that } \hat{\gamma}, \tau' \models \psi \text{ and } \hat{\gamma}, \kappa \models \phi\ \forall \kappa \in (\tau', \tau] \text{ or} \\
 & & \quad \exists\ \tau'' \in\ [0, \tau') \text{ such that } \hat{\gamma}, \kappa \models \psi\ \forall \kappa \in [\tau'', \tau') \text{ and } \hat{\gamma}, \kappa \models \phi\ \forall \kappa \in [\tau', \tau] \\
\hat{\gamma}, \tau \models \phi\ \mathcal{U}\ \psi & \text{iff} & \exists\ \tau' \geq \tau \text{ such that } \hat{\gamma}, \tau' \models \psi \text{ and} \hat{\gamma}, \kappa \models \phi\ \forall \kappa \in (\tau', \tau] \text{ or} \\
 & & \quad \exists\ \tau'' > \tau' \text{ such that } \hat{\gamma}, \kappa \models \psi\ \forall \kappa \in (\tau', \tau''] \text{ and } \hat{\gamma}, \kappa \models \phi\ \forall \kappa \in [\tau, \tau']
\end{array}$$

Fig. 2. Semantics of Monitored MTL Connectives

4.2 Monitoring Algorithm

To monitor a formula ϕ, we begin by converting it into a parse tree. From this, we construct an array Φ consisting of one transducer for each node of the parse tree in reverse-topological-sort order. That is, for any node in the parse tree, its children appear before it in Φ. The transducers maintain some operation-specific fields, but each contains at least $\langle op, inputs, Q, I_{\text{valid}} \rangle$ where *op* identifies the operation, *inputs* contains a pointer to the elements of Φ upon which it depends, Q is a queue containing the output of the transducer, and I_{valid} indicates the time interval over which the output of the transducer is valid.

The valid interval allows a transducer to 'stop time' while its state is undetermined. For example, the transducer for metric *eventually* uses this to apply a constant offset to all output transitions, whereas the *until* transducer may delay its output for an indeterminate period. Even the simple transducers such as negation and conjunction must be able to specify a valid interval since their input might do so.

Some transducers define additional state variables in addition to those mentioned above. The *since* transducer maintains a state indicating whether or not the latest transition left its output in the UP state; The *eventually* transducer maintains a timer that is used by the monitoring algorithm to call UPDATE again at some point in the future.

The monitoring procedure consists of gathering an ensemble of simultaneous transition events for the external inputs to the monitor and storing them in a container, Δ, that maps input variable names to transition types. If a timer expires, UPDATE may be called with no input transitions. The UPDATE function returns the next timer expiration time so that the monitor may call it when it

has expired. The following pseudocode describes the most general version of the update operation and the UPDATE procedure. Subsequent sections will describe the simplifications that are possible when supporting subsets of MTL.

```
1  function UPDATE(Φ, Δ, τ)
2      for φ ∈ Φ do
3          if φ.op ∈ input variables then
4              if φ.op.id ∈ Δ then
5                  ENQUEUE(φ_Q, Δ[φ.op.id], τ)
6                  φ.I_valid ← [0, τ]
7          else
8              while (δ*, τ) ← SYNC(φ.inputs, φ.τ_timer) do
9                  UPDATE_φ.op(φ, δ*, τ)
10         UPDATEVALIDINTERVAL_φ.op(φ)
11     return min({φ.τ_timer for φ ∈ Φ})
```

The return value indicates the time at which the earliest timer will expire. The use of this is dependent upon the resources available to the monitor. For a real-time monitor, it will generally be used to schedule an operating-system or hardware timer. For an offline monitor, there may be no need to make use of the timer until the end of the input sequence at which point either each timer can be expired individually or a final UPDATE call can be made at a timestamp later than the latest timer.

The update procedure depends on five sub-procedures: The ENQUEUE, DEQUEUE. SYNC, and two operation-specific procedures, $\text{UPDATE}_{\varphi.\text{op}}$ and $\text{UPDATEVALIDINTERVAL}_{\varphi.\text{op}}(\varphi)$. The first and second are related to delivering input to the transducers that conforms to the rules of Definition 1 and that binary operators receive their input events in the correct order. The second two – actually a family of procedures, one for each transducer type – are used to produce the output of each transducer.

The ENQUEUE procedure supports the 'time-shifting' property of the *eventually* operator by ensuring that its output obeys the initial-transition properties of Definition 1. Lines 4 through 8 handle the case for which the time-shift produces a transition with timestamp less than zero, its potential initial transition is set to the constant value ($\overline{\ }$ or _) corresponding to the value of the signal immediately after the transition. In this case, the valid interval on the output will end at a timestamp less than zero, so this process can be repeated until the time-shift of the transducer is no longer able to produce transitions with timestamp less than zero. Similarly, if input arrives with a timestamp of zero, lines 10 through 18 ensure that the initial transition of the output is of a legal initial-transition type for which the truth value agrees with that of the input type for $\tau = 0$ and immediately afterward – that is, before the next transition event. Finally, if the time shift produces a first transition with timestamp greater than zero, lines 20 through 27 guarantee that there will be a constant-valued transition (_ or $\overline{\ }$) matched to the current transition type being enqueued with timestamp zero added before adding the current transition. Note that if the first transition type is either _ or $\overline{\ }$, only the zero-timestamped transition will be added because those transition types are only legal for the initial transition.

```
1  function ENQUEUE(Q, δ, τ)
2      if δ = ∅ then return
3      case τ < 0
4          Clear(Q)
5          if δ ∈ {↧, ↓, ↥} then
6              APPEND(Q,(‾,0))
7          else
8              APPEND(Q,(_,0))
9      case τ = 0
10         Clear(Q)
11         case δ ∈ {_, ‾}
12             APPEND(Q,(δ, 0))
13         case δ = ↧
14             APPEND(Q,(‾,0))
15         case δ ∈ {↥, ↓}
16             APPEND(Q,(↓,0))
17         case δ ∈ {↧, ↥}
18             APPEND(Q,(↥,0))
19         case δ = ↥
20             APPEND(Q,(_,0))
21     case τ > 0
22         if EMPTY(Q) ∧ Q.value = ∅ then
23             if δ ∈ {_, ‾} then
24                 APPEND(Q,(δ, 0))
25             else if δ ∈ {↧, ↓, ↥} then
26                 APPEND(Q,(_, 0))
27             else
28                 APPEND(Q,(‾, 0))
29         APPEND(Q,(δ, τ))
```

Whereas the ENQUEUE procedure ensures that the initial conditions of the output of each stage of the transducer are correct, the DEQUEUE ensures that the output boolean signal will return the correct constant value if queried at a time for which there is no transition event even if all transitions have been consumed from its queue. This is simply a matter of storing the constant *true* or *false* transition depending on the truth value that will immediately follow the current transition. The simplicity of the DEQUEUE procedure depends on the SYNC procedure only calling it at the time of the next transition in its queue.

```
1  function DEQUEUE(Q)
2      (δ, τ) ← REMOVEFIRST(Q)
3      case δ ∈ {‾, ↧, ↓, ↥}
4          Q.value ← ‾
5      case δ ∈ {_, ↥, ↧, ↧}
6          Q.value ← _
7      return (δ, τ)
```

The SYNC procedure handles the synchronization of multiple inputs of a single transducer stage. This allows the transducer UPDATE procedures to depend upon always receiving one transition on each of its inputs – even if those transitions are actually the constant values ‾ or _. As presented here, it allows the UPDATE procedure to treat unary and binary transducers identically. The *eventually* transducer is the only one for which the timer must be considered, so it is handled in the unary-tranducer section. Timer expiration is effectively treated like a special case of a binary operator. If the time expires before the next input transition, the SYNC procedure returns the constant-value transition and the timestamp of the timer expiration. Otherwise, it returns the type and timestamp of the first input transition that is within the valid interval of the input signal.

```
1  function SYNC(inputs, τ_timer)
2      φ, ψ ← inputs
3      if ψ = ∅ then                                          ▷ one input
4          if EMPTY(φ.Q) ∨ HEAD(φ.Q).τ > τ_timer then
5              if τ_timer ∈ φ.I_valid then
6                  return (φ.Q.value, τ_timer)
7              return ∅
8          return DEQUEUE(φ.Q)
9      else                                                   ▷ two inputs
10         I_valid ← φ.I_valid ∩ ψ.I_valid
11         e_φ ← HEAD(φ.Q) if HEAD(φ.Q).τ ∈ I_valid
12         e_ψ ← HEAD(ψ.Q) if HEAD(ψ.Q).τ ∈ I_valid
13         if e_φ = ∅ ∧ e_ψ = ∅ then
14             return ∅
15         if e_φ ≠ ∅ ∧ e_ψ ≠ ∅ then
16             if e_φ.τ = e_ψ.τ then
17                 DEQUEUE(φ.Q)
18                 DEQUEUE(ψ.Q)
19                 return ((e_φ.δ, e_ψ.δ), e_φ.τ)
20             if e_φ.τ < e_ψ.τ then
21                 e_ψ ← ∅
22             else
23                 e_φ ← ∅
24         if e_φ ≠ ∅ then
25             DEQUEUE(φ.Q)
26             return ((e_φ.δ, ψ.Q.value), e_φ.τ)
27         else
28             DEQUEUE(ψ.Q)
29             return ((φ.Q.value, e_ψ.δ), e_φ.τ)
```

For binary transducers, the possible scenarios are: neither input has a transition; one input has a transition, but the other does not; both inputs have transitions, but they have different timestamps; and both inputs have transitions with the same timestamp. The SYNC procedure handles these cases in lines 10 through 12 by storing the contents of the head of each of the input queues if their respective timestamp are within the valid interval of both of the inputs.

Lines 13 and 14 handle the case in which neither has a transition. Since no transducer will produce an output transition when the truth value of neither of its inputs changes, there is no need to process further. Lines 15 through 19 handle the case for which both inputs exhibit a simultaneous transition by consuming the first transition on each input and returning the pair of transitions types. Lines 20 through 23 convert the case for which both input have a valid transition, but at different timestamps, into effectively the case for which only the input with the earliest transition contained a transition. Lines 24 through 29 handle this and the case in which only one input exhibited a transition within the valid interval. It does this by consuming one whichever of the inputs has a transition and sending the appropriate constant-value transition type (_or ¯) for the other input.

5 Signal Transducer Tables

5.1 Negation and Conjunction

Figure 3 contains the transition tables for negation and conjunction. The INIT versions are used for the first transition and the regular version for all subsequent transitions. The non-transition states of the inputs are illustrated, but table entries for which there is no transition are omitted in the interest of readability. Note that for the INIT versions of the tables, the non-transition outputs ($\overline{\ }$ and _) are shown because they are actually produced for the initial entry of a boolean signal.

```
1 function UPDATE¬(φ, δ, τ)
2     if τ = 0 then
3         ENQUEUE(φ.Q, NOT_INIT[δ], τ)
4     else
5         ENQUEUE(φ.Q, NOT[δ], τ)
6     φ.I_valid ← [0, τ]
7 function UPDATEVALIDINTERVAL¬(φ)
8     φ.I_valid ← φ.inputs.I_valid
```

```
1 function UPDATE∧(φ, δ_φ, δ_ψ, τ)
2     if τ = 0 then
3         ENQUEUE(φ.Q, AND_INIT[δ_φ, δ_ψ], τ)
4     else
5         ENQUEUE(φ.Q, AND[δ_φ, δ_ψ], τ)
6     φ.I_valid ← [0, τ]
7 function UPDATEVALIDINTERVAL∧(φ)
8     φ, ψ ← φ.inputs
9     I_valid ← φ.I_valid ∩ ψ.I_valid
```

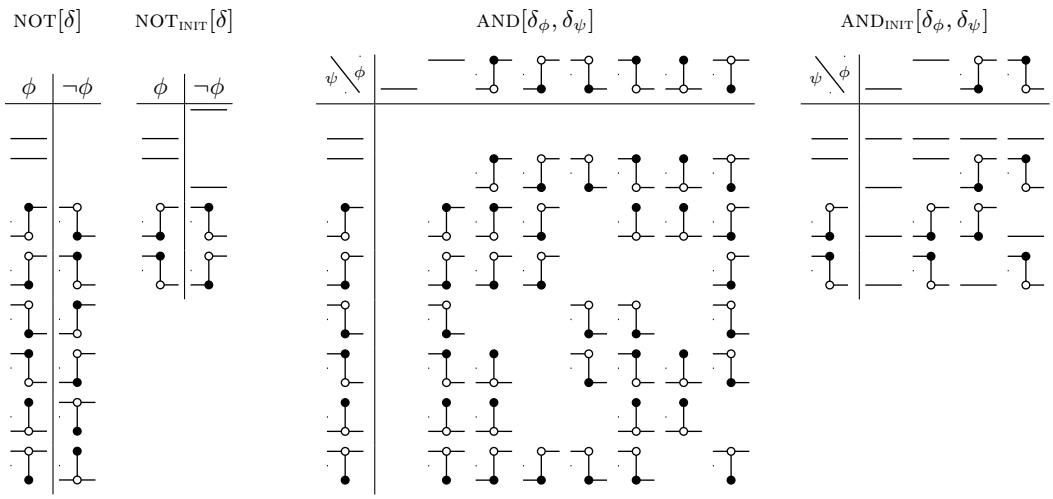


Fig. 3. Transition Tables for the Negation and Conjunction Operators

5.2 Since

The transducer for $\phi \mathcal{S} \psi$ is somewhat more complicated. The output transition for a given set of input transitions is influenced by whether or not the since operator is currently in the down or the up state. As in the previous section, omitted entries indicate states for which there is no transition. Some of the omitted entries indicate states that cannot be reached, but these are not specially indicated – nor is any special handling required to deal with unreachable states.

The transducer begins by applying the table $\text{SINCE}_{\text{INIT}}[\delta_\phi, \delta_\psi]$ for the initial transition, afterward the tables $\text{SINCE}_{\text{UP}}[\delta_\phi, \delta_\psi]$ and $\text{SINCE}_{\text{DOWN}}[\delta_\phi, \delta_\psi]$ are used. Which of the tables is to be used is determined by the type of transition last emitted. If it is in $\{\overline{}, \uparrow, \downarrow, \updownarrow\}$, then the UP table is used, otherwise, it is the DOWN table.

```
1  function UpdateS(φ, δφ, δψ, τ)
2      if τ = 0 then
3          δ' ← SinceInit[δφ, δψ]
4      else if φ.state = up then
5          δ' ← SinceUp[δφ, δψ]
6      else
7          δ' ← SinceDown[δφ, δψ]
8      if δ' ≠ ∅ then
9          Enqueue(φ.Q, δ', τ)
10         φ.state ← { up    δ' ∈ {—, ↑, ↓, ↕}
                     { down  else
11     φ.I_valid ← [0, τ]
12 function UpdateValidIntervalS(φ)
13     φ, ψ ← φ.inputs
14     I_valid ← φ.I_valid ∩ ψ.I_valid
```

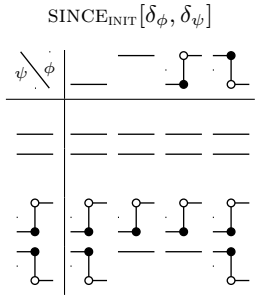


Fig. 4. Initialization Table for the *Since* Operator

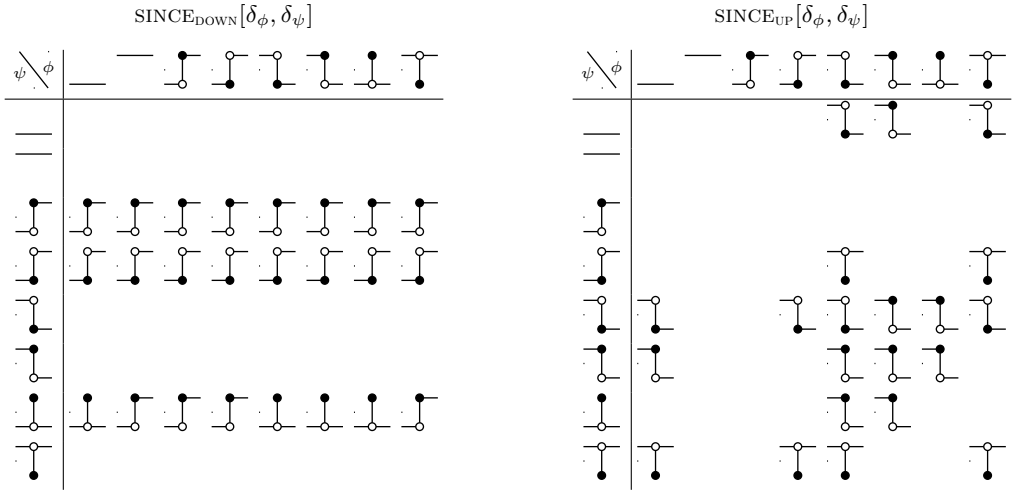


Fig. 5. Transition Tables for the *Since* Operator

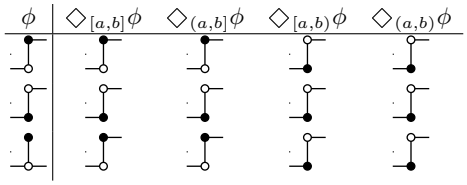
5.3 Eventually

The metric eventually operator uses the tables from Figure 6. Its transducer is distinguished from those introduced thus far by the addition of a timer used to generate a down event $b-a$ time units after the input transitions to the down state. All events emitted in response to an event at time τ are emitted at time $\tau-b$. Also, it can enter an indeterminate state in response to a down transition. When a down transition occurs, the transducer stores the potential down transition – the type of which is determined by the interval type – in the state variable $\delta_{\downarrow}$. The actual transition emitted can be affected if an up transition occurs before or simultaneous with the timer expiration.

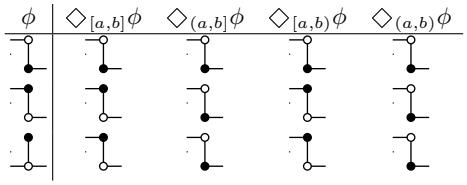
```
1  function UPDATE_{◇_I}(φ, δ, τ)
2      τ' ← τ − b                                  ▷ output time offset
3      if τ = 0 ∧ δ ∈ {‾, _} then
4          ENQUEUE(φ.Q, δ, τ')
5      else if φ.τ_timer = ∅ then
6          if δ ∈ EVENTUALLY_a then
7              ENQUEUE(φ.Q, EVENTUALLY_a[δ], τ')
8      else
9          if τ = φ.τ_timer then
10             if δ = ∅ then
11                 ENQUEUE(φ.Q, φ.δ_↓, τ')
12             else if φ.δ_↓ = ↴ then
13                 ENQUEUE(φ.Q, EVENTUALLY_c[δ], τ')
14         φ.δ_↓ ← ∅
15         φ.τ_timer ← ∅
16     if δ ∈ EVENTUALLY_b then                    ▷ down transition
17         φ.δ_↓ ← EVENTUALLY_b[δ]
18         φ.τ_timer ← τ + b − a
```

```
1  function UPDATEVALIDINTERVAL_{◇_I}(φ)
2      φ ← φ.inputs
3      switch φ.I_valid
4          case [0, i]
5              φ.I_valid ← [0, i − b]
6          case [0, i)
7              φ.I_valid ← [0, i − b)
```

EVENTUALLY$_a[\delta_\phi]$: Up Transition



EVENTUALLY$_b[\delta_\phi]$: Potential Down Transition



EVENTUALLY$_c[\delta_\phi]$: Up Transition and Potential ↴

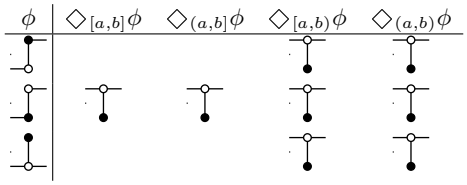


Fig. 6. Transition Tables for the *Eventually* Operator

5.4 Until

The transducer for the *until* operator can enter a more complicated indeterminate state than that of the non-metric *eventually* operator; It can maintain a different potential transition depending on whether it is ultimately found to be *true* or *false* at the point in time at which its output became uncertain. The simplest example occurs when monitoring $\phi \,\mathcal{U}\, \psi$ and ϕ becomes *true* with transition type υ at a time τ when ψ is *false*. If ψ becomes *true* before ϕ becomes *false*, then the transducer should emit the transition event (υ, τ); If ϕ becomes *true* before ψ becomes *false*, this potential transition is abandoned and the output remains *false* up to and including the current time. It is possible that the transducer will maintain two such potential transitions, $\delta_\downarrow$ or $\delta_\uparrow$. We introduce the notations *non-deterministic up* υ and *non-deterministic down* $\downsilon$ in $\textsc{until}_\text{a}$ to denote which of the potential transitions is to be emitted upon the arrival of input events.

```
1  function UPDATE_U(φ, δ_φ, δ_ψ, τ)
2      if τ = 0 then
3          ENQUEUE(φ.Q, UNTIL_INITa[δ_φ, δ_ψ], τ)
4          φ.δ↑ ← UNTIL_INITb[δ_φ, δ_ψ]
5          φ.δ↓ ← UNTIL_INITc[δ_φ, δ_ψ]
6          φ.τ_pending ← τ
7      else
8          switch UNTIL_a[δ_φ, δ_ψ]
9              τ′ ← φ.τ_pending
10             case υ
11                 ENQUEUE(φ.Q, φ.δ↑, τ′)
12             case ↓
13                 ENQUEUE(φ.Q, φ.δ↓, τ′)
14         ENQUEUE(φ.Q, UNTIL_b[δ_φ, δ_ψ], τ)
15         φ.δ↑ ← UNTIL_c[δ_φ, δ_ψ]
16         φ.δ↓ ← UNTIL_INITd[δ_φ, δ_ψ]
17         φ.τ_pending ← τ
18     if ¬(φ.δ↑ = φ.δ↓ = ∅) then
19         φ.I_valid ← [0, τ)
```

```
1  function UPDATEVALIDINTERVAL_U(φ)
2      φ, ψ ← φ.inputs
3      if φ.δ↑ = φ.δ↓ = ∅ then
4          φ.I_valid ← φ.I_valid ∩ ψ.I_valid
```

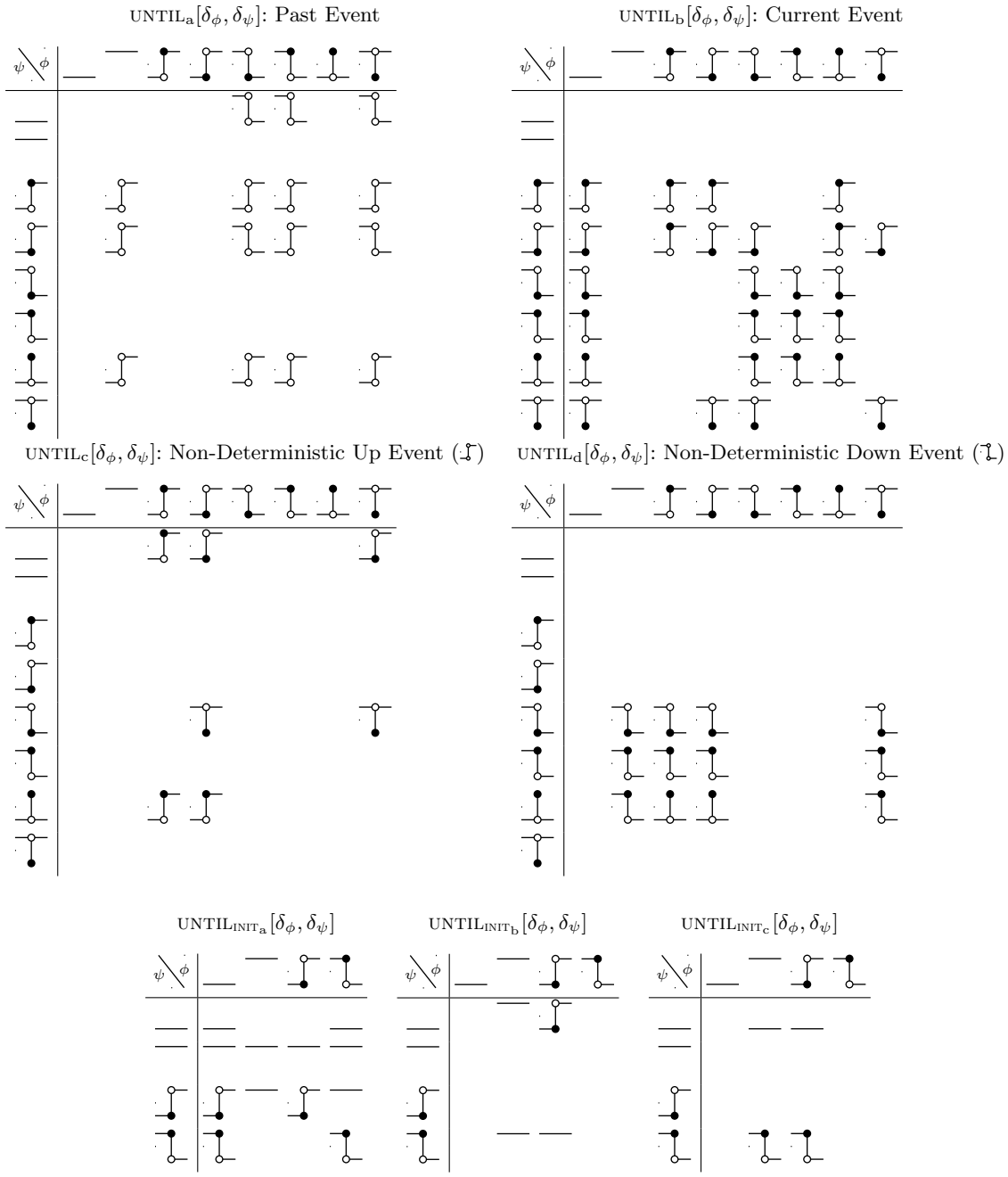


Fig. 7. Transition Tables for the *Until* Operator, $\phi \, \mathcal{U} \, \psi$

6 Correctness

Theorem 1. *The* UPDATE *procedure applied to the above transducer tables correctly models the semantics of Figure 1.*

Proof. See appendix A.

7 Monitoring-Algorithm Complexity

7.1 Instantaneous Transducers

The instantaneous transducers are defined as those for which all emitted transitions take place at the current time, that is, with the same τ as that of the event that caused them. They comprise $\neg$, $\wedge$, $\mathcal{S}$, and $\diamondsuit_{[-a,0]}$.

The UPDATE procedure can be simplified in that there is no need to keep track of the *valid* intervals. Without the potential for delayed output, the queues will never grow larger than one element and can be replaced with single values.

Theorem 2. *The runtime to monitor expression ϕ that consists of only instantaneous operators on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs.*

Proof. The proof is provided in appendix B, but is reasonably clear from the pseudocode if it is given that ENQUEUE, DEQUEUE, and SYNC run in constant time.

Theorem 3. *The space required to monitor expression ϕ that consists of only instantaneous operators on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|)$.*

Proof. The proof is provided in appendix B and centers on a proof that the queues for the instantaneous transducers do not grow larger than one element.

7.2 Strictly Past Transducer

The transducer for the operator $\diamondsuit_{[a,b]}$ for $a <= b < 0$ operates identically to that of $\diamondsuit_{[b-a,0]}$ except that the timestamp of the output that results from an event at time τ is $\tau - b$. The corresponding statement is true for other intervals with open bounds as well. This introduces the need to maintain valid ranges and queues to store the output of the intermediate stages to support operators with multiple inputs.

Theorem 4. *The runtime to monitor expression ϕ that consists of only past-time operators on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

Theorem 5. *The space required to monitor expression ϕ that consists of only past-time operators with $\diamondsuit_{[a,b]}$ where $a < b < 0$ on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi| \left\lfloor \frac{|b|}{b-a} \right\rfloor)$.*

Theorem 6. *The space required to monitor expression ϕ that consists of only past-time operators with $\diamondsuit_{[a,a]}$ where $a < 0$ on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

7.3 Restricted Future

The next increment in monitor complexity introduces the metric eventually operator $\Diamond_{[a,b]}$ with $b > 0$. From the UPDATE procedure, we see that it introduces a delay in its output events relative to the input events that produces them. This adds no computational complexity, but reduces the guarantees that can be made about space complexity even when $a \neq b$.

Theorem 7. *The runtime to monitor expression ϕ that consists of past-time and restricted-future operators on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

Theorem 8. *The space required to monitor expression ϕ that consists of only past-time operators with $\Diamond_{[a,b]}$ where $a <= b$ and $b > 0$ on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

7.4 Unrestricted Future

The introduction of the *until* operator presents two challenges related to the fact that it may remain in an indeterminate state for an arbitrary length of time. The unchanged space complexity belies the fact that whereas a bound may be placed on the growth of the size of the monitor for restricted-future operators if a limit can be placed on the number of transitions within any time interval, no such limit can be placed on the size of the monitor for unrestricted-future operators. Also, it is possible to construct liveness properties that can not be falsified by a monitoring procedure.

Theorem 9. *The runtime to monitor expression ϕ that consists of past-time and restricted-future operators on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

Theorem 10. *The space required to monitor expression ϕ that consists of only past-time operators with $\Diamond_{[a,b]}$ where $a <= b$ and $b > 0$ on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

8 Conclusion

We have presented straightforward procedures for monitoring dense-time continuous-semantics MTL formulae as well as the tradeoffs in runtime and space complexity incurred as the expressiveness of the supported formulae increases.

We have included the unrestricted-future operators to demonstrate support for full MTL but also because policy writers may find them to be the most natural way of representing the policy that they wish to enforce. That said, they must be used with care as they introduce the ability to describe pure liveness properties for which no truth value will ever be determined, such as $\neg\Diamond\neg\Diamond\phi$.

Future work may include mechanisms for trimming the boolean signals of subexpressions that cannot affect the truth of the full monitored expression as well as the augmentation to the unrestricted-future monitoring algorithm to support the extension of the valid-interval for binary operators in cases for which its output value can be determined based on only the one of its inputs for which the valid-interval extends further in time. For example, a conjunction for which one of its input is unknown beyond τ), but the other is known to be *false* over the entire interval $[\tau, \textit{current time})$.

More immediately, we intend to pursue a VHDL implementation of the subset of MTL for which size restrictions can be guaranteed.

9 Acknowlegements

Jianwei Niu is supported in part by NSF award CNS-0964710 and the UTSA research award TRAC-2008.

References

1. Kevin Baldor and Jianwei Niu. Monitoring dense-time, continuous-semantics, metric temporal logic. In Shaz Qadeer and Serdar Tasiran, editors, *Runtime Verification*, volume 7687 of *Lecture Notes in Computer Science*, pages 245–259. Springer Berlin Heidelberg, 2013.
2. David Basin, Felix Klaedtke, and Eugen Zălinescu. Algorithms for monitoring real-time properties. In *Proceedings of the 2nd International Conference on Runtime Verification (RV)*, volume 7186 of *Lecture Notes in Computer Science*, pages 260–275. Springer-Verlag, 2012.
3. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, April 1986.
4. Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In *In Tools and Algorithms for Construction and Analysis of Systems (TACAS02*, pages 342–356. Springer, 2002.
5. Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2:255–299, 1990. 10.1007/BF01995674.
6. Oded Maler, Dejan Nickovic, and Amir Pnueli. From mitl to timed automata. In Eugene Asarin and Patricia Bouyer, editors, *Formal Modeling and Analysis of Timed Systems*, volume 4202 of *Lecture Notes in Computer Science*, pages 274–289. Springer Berlin / Heidelberg, 2006. 10.1007/11867340_20.
7. Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.
8. Pavithra Prabhakar and Deepak D'Souza. On the expressiveness of mtl with past operators. In *Proceedings of the 4th international conference on Formal Modeling and Analysis of Timed Systems*, FORMATS'06, pages 322–336, Berlin, Heidelberg, 2006. Springer-Verlag.

A Correctness

This section demonstrates that the update procedure applied to the transducer tables models the continuous semantics of MTL. The transducer approach allows this proof to be broken into a proof of correctness for each transducer alone. That is, once we have established that the update procedure presents the output of each stage to the input of subsequent stages in a timely manner, the correctness of the output follows from the correctness of each stage.

$$\hat{\gamma}, \tau \models \neg\phi \quad \text{iff} \quad \hat{\gamma}, \tau \not\models \phi \tag{1}$$

$$\hat{\gamma}, \tau \models \phi \wedge \psi \quad \text{iff} \quad \hat{\gamma}, \tau \models \phi \text{ and } \hat{\gamma}, \tau \models \psi \tag{2}$$

$$\hat{\gamma}, \tau \models \Diamond_I \ \phi \quad \text{iff} \quad \exists\ \tau' \text{such that } \tau' - \tau \in\ I,\ \hat{\gamma}, \tau' \models\ \phi \tag{3}$$

$$\hat{\gamma}, \tau \models \phi\ \mathcal{S}\ \psi \quad \text{iff} \quad \exists\ \tau' \in\ [0, \tau] \text{ such that } \hat{\gamma}, \tau' \models \psi \text{ and } \hat{\gamma}, \kappa \models \phi\ \forall \kappa \in (\tau', \tau] \text{ or} \tag{4}$$

$$\exists\ \tau'' \in\ [0, \tau') \text{ such that } \hat{\gamma}, \kappa \models \psi\ \forall \kappa \in [\tau'', \tau') \text{ and } \hat{\gamma}, \kappa \models \phi\ \forall \kappa \in [\tau', \tau] \tag{5}$$

$$\hat{\gamma}, \tau \models \phi\ \mathcal{U}\ \psi \quad \text{iff} \quad \exists\ \tau' \geq \tau \text{ such that } \hat{\gamma}, \tau' \models \psi \text{ and} \hat{\gamma}, \kappa \models \phi\ \forall \kappa \in (\tau', \tau] \text{ or} \tag{6}$$

$$\exists\ \tau'' > \tau' \text{ such that } \hat{\gamma}, \kappa \models \psi\ \forall \kappa \in (\tau', \tau''] \text{ and } \hat{\gamma}, \kappa \models \phi\ \forall \kappa \in [\tau, \tau'] \tag{7}$$

In the following proofs, we will use the notation γ_ϕ to represent the transition-sequence model of the signal that describes the truth evaluation of the signal ϕ. To paraphrase definition 2, we have the truth evaluation of ϕ at time τ represented as $\tau \in \gamma_\phi$ when γ_ϕ contains a transition at time τ that is in $\{\overline{\ }, \mathfrak{f}, \mathfrak{t}\}$ or γ_ϕ does not contain a transition at time τ, but the transition immediately preceding time τ is in $\{\overline{\ }, \mathfrak{f}, \mathfrak{f}, \mathfrak{t}\}$.

We also will speak of transitions taking place at a specific time τ and will discuss the truth value immediately preceding, τ^-, and immediately following, τ^+, this time. By immediately preceding, we mean any time in (τ', τ) where τ' is the time of the transition with the maximum timestamp that is less than τ. Similarly, immediately following indicates any timestamp in (τ, τ'') where τ'' is the time of the transition with the minimum timestamp that is greater than τ if such a transition exists or in (τ, ∞) if it does not. When discussing initial transitions for which $\tau = 0$, we will use 0^+ to represent the time immediately after $\tau = 0$.

Lemma 1. *The application of the tables* NOT_{INIT} *and* NOT *in the update procedure satisfies the negation operation (1).*

Proof. Taking γ_ϕ as the input, the output, $\gamma_{\neg\phi}$, of the transducer must be *true* only at those time point for which γ_ϕ is *false*.

We begin with $\tau = 0$, when γ_ϕ must contain a transition in $\{\overline{\ }, \_, \mathfrak{f}, \mathfrak{t}\}$. Because this transition takes place at time $\tau = 0$, the table NOT_{INIT} will be applied to produce the first output transition. From Figure 3, we see that the corresponding output for each of the inputs is $\{\_, \overline{\ }, \mathfrak{t}, \mathfrak{f}\}$. Definition 1 gives that $\gamma_{\neg\phi}, 0 \not\models\ \phi$. That is, at time $\tau = 0$, if the input is *true*, the output if *false*, and vice versa. We also see that any input transition that leaves the input *true* at time 0^+ leaves the output *false* until the next transition and vice versa.. Thus, we know that the transducer produces correct output for time $\tau \in [0, \tau')$, where τ' is the time of the second transition of the input.

Next, we consider subsequent transitions beginning with the first transition after $\tau = 0$. The previous analysis established that immediately prior to the next transition either γ_ϕ or $\gamma_{\neg\phi}$ is *true*,

but not both. Let us first consider the case where γ_ϕ is *false* at τ^-. From Definition 1, we have that the next transition must be in $\{\uparrow, \uparrow, \downarrow\}$. The table NOT in Figure 3 gives the corresponding outputs as $\{\uparrow, \downarrow, \uparrow\}$. Observe that in each case, the value at τ^- is *false* for γ_ϕ and *true* for $\gamma_{\neg\phi}$ and that the truth evaluation of each is the opposite of the other at the time of the transition and at times τ and τ^+. Similarly, when γ_ϕ is *true* at τ^-, its next transition must be in $\{\uparrow, \downarrow, \uparrow\}$and the table NOT gives the resulting output transitions as $\{\downarrow, \downarrow, \downarrow\}$. As before, these satisfy the negation property. In either case, we are left at τ^+ in a state such that the value of $\gamma_{\neg\phi}$ is the opposite of that of γ_ϕ, just as we had for time 0^+.

Having shown that an initial transition in $\{\overline{\ }, \_, \uparrow, \downarrow\}$ will be produced at time $\tau = 0$ that satisfies the negation property w.r.t. the input initial transition and that from any state for which the negation property holds a legal input transition will result in a legal output transition that satisfies the negation property, we can state that the update procedure applied to the NOT_{INIT} and NOT tables models the negation operation.

Lemma 2. *The application of the tables* AND_{INIT} *and* AND *in the update procedure satisfies the conjunction operation (2).*

Proof. Taking γ_ϕ and γ_ψ as the inputs, the output, $\gamma_{\phi\wedge\psi}$, of the transducer must be *true* only at those time point for which both γ_ϕ and γ_ψ are *true*.

We first consider the initial state. We begin by noting that the first row and first column of AND_{INIT} – those that correspond to either γ_ϕ or γ_ψ having the initial state _ – consist entirely of output transitions of type _This is correct because if either input is *false* at all three τ^-, τ, and τ^+ then the output must also be *false* at τ^-, τ, and τ^+. Next, we consider the second row and second column – those that correspond to either γ_ϕ or γ_ψ having the initial state $\overline{\ }$. In this row or column, note that the output transition type is identical to that of the input that is not constant on that row. This is correct because if either input is *true* at all three τ^-, τ, and τ^+ then the truth – and thus transition type – is completely determined by the truth of the other input. Next note the diagonal for which γ_ϕ and γ_ψ have the same initial transition type. The output transition type in these cases is identical to that of each of the inputs. Finally, consider the case where the initial transition types of γ_ϕ and γ_ψ are $(\uparrow, \downarrow)$ or vice versa. The output transition type for each is _, which is correct because in both of these cases, the value of each input is the opposite of that of the other at each time τ^-, τ, and τ^+.

Changes in the state of either input are handled by selecting the output transition type from AND. First note that it is symmetrical about the diagonal defined by each input having the same transition type. In the following analysis, we will consider only the upper triangle, but all of the arguments will hold for the lower triangle. Also, as this is evaluated only at time $\tau > 0$, we needn't concern ourselves with the states for which both of the inputs are in $\{\_, \overline{\ }\}$ as the update procedure will not be called if neither input experiences a transition that results in a change to its truth value at τ or τ^+.

To avoid tedious analysis, we assert that all entries in the table for which an output transition is provided can be seen to be correct by observing that the output transition type is determined by the truth of the input transitions at times τ^-, τ, and τ^+. That is, the output transition will only be *true* at τ^- if both input transitions are *true* at τ^- and so on. The entries for which there is no output transition given are those for which the the transition type would have been _, but in either case no output transition will be produced because the transition model for boolean signals forbids the redundant production of _ or $\overline{\ }$ for any $\tau > 0$.

Lemma 3. *The application of the tables* EVENTUALLY$_a[\phi]$, EVENTUALLY$_b[\phi]$, *and* EVENTUALLY$_c[\phi]$ *in the update procedure satisfies the* eventually *operation (3).*

Proof. Since the metric *eventually* operator can introduce a time shift, there is less of a clear distinction between the initial transition and the subsequent transitions. The legality of the output of the transducer is guaranteed by the ENQUEUE operation on its output. However, since the initial transition can exhibit the constant-valued transition types $\_$ and $\bar{\ }$, lines 3 and 4 are required to handle it. Quite simply, if the input is known to be true at time zero, then the output must also be true at an offset of b time units, and vice versa. This is correct because for an output at timestamp $-b$ the interval to check for input is $[b-a, 0]$ and the input will either be *true* at all point in that interval or *false* at all points in that interval. Thus, it doesn't matter whether the interval on the metric operator is closed at the minimum or maximum extent.

Subsequent transitions are handled as follows.

The algorithm uses the fact that the timer is set or not set to determine the state of the transducer. The timer is only set when a 'down' transition, one for which the input is *false* at time τ or τ^+, occurs. Thus, if it is set, the output must be in the *true* state, but will become *false* when the timer expires unless an 'up' transition arrives on its input before or simultaneous with the timer expiration.

We will begin with the handling of an 'up' transition on the input, one for which the input is *true*at time τ or τ^+, when the input has been *false* for longer than the last $b-a$ time units. As the output of the transducer must be *false* at such a time, this must yield an output transition that is *true* at time $(\tau - b)^+$. The output at time $(\tau - b)$ is determined by whether the interval on the operator is closed at the maximal value and whether the input is *true* at time τ. Inspection of table EVENTUALLY$_\text{a}$ bears this out and lines 5 and 6 ensure that it is applied only in this case.

If the input exhibits a 'down' transition it becomes possible that the output will also exhibit a 'down' transition, but if an up transition occurs within $b-a$ time units this 'down' transition will not take place. This is handled by storing the type of down transition that will take place $b-a$ (the width of the interval) time units later if no such 'up' events arrive and saving the timestamp $\tau+b-a$. Similar to the case for the 'up' transitions, the potential down transitions are characterized by being *true* at time $(\tau+b-a)^-$ and *false* at time $(\tau+b-a)^+$, but the truth at time $(\tau+b-a)$ is determined by the truth of the input at time τ and by whether the interval is closed at its minimal value. Inspection of EVENTUALLY$_\text{b}$ bears this out.

If there is a timer set that it has not expired, the arrival of an 'up' transition will simply cancel the timer as the output will remain true for all future time unless a 'down' event arrives. This is accomplished by line 9, which guarantees that no output transition is produced, and lines 14 and 15, which cancel the timer.

Should the timer expire before the arrival of any input transitions, the stored 'down' transition is emitted with the timer-expiration timestamp shifted – as always –by b time units. This is handled by lines 9 through 11 with lines 14 and 15 canceling the timer as above.

If an 'up' input transition arrives precisely at the time of a timer expiration then we know that the output must be *true* at time $(\tau - b)^+$. If the potential down transition was of type $\overline{\downarrow}$, then this would result in the non-transition $\bar{\ }$, so line 14 ensures that the update procedure emits no transition in this case. If, however, the potential down transition was of type $\underline{\downarrow}$ it is possible that the output will be *false* at time $(\tau - b)$ producing an output transition of type $\downarrow$. This is determined just as it was for simple 'up' transitions and table EVENTUALLY$_\text{c}$ illustrates the input transitions and interval types for which this is the case.

Note that ↥ is both an 'up' transition and a 'down' transition as we have defined them for this proof. Thus conditional on line 16 is checked even for algorithmic states for which only an 'up' transition is possible.

Lemma 4. *The application of the tables* $\text{SINCE}_{\text{INIT}}$, SINCE_{UP}, *and* $\text{SINCE}_{\text{DOWN}}$ *in the update procedure satisfies the* since *operation (4).*

Proof. From the initial state, we begin by observing the degenerate behavior of $\phi \mathcal{S} \psi$ – it is true at any time τ for which ψ is *true* regardless of the value of ϕ at time τ or at any point $\tau' > \tau$. Otherwise, $\phi \mathcal{S} \psi$ can only be *true* at time τ if ψ has been *true* at some time $\tau' < \tau$. As a result, the initial transition type of $\gamma_{\phi \mathcal{S} \psi}$ is equal to the initial transition of γ_ψ for all but two cases, those for which ψ is ↧ and $\phi \in \{\text{¯}, ↥\}$. In each of these cases, the output transition is of type ¯. Its truth at time τ^- and τ is given by the truth of the initial transition of ψ at those times, but the truth of the output at time τ^+ is given by the truth of the initial transition of ϕ at τ^+.

At time $\tau = 0^+$, it is possible to be in any of the reachable states of the *since* operator: $\gamma_{\phi \mathcal{S} \psi}$ = _ ,where (γ_ϕ, γ_ψ) may be either (_, _) or (¯, _), and $\gamma_{\phi \mathcal{S} \psi}$ = ¯, where (γ_ϕ, γ_ψ) will be in {(_, ¯), (¯, ¯), (¯, _)}.

For any time τ for which one or both of the inputs exhibits a transition, if $\gamma_{\phi \mathcal{S} \psi} = \textit{false}$ at τ^-, the table $\text{SINCE}_{\text{DOWN}}$ is used to determine the output transition. Conversely, if $\gamma_{\phi \mathcal{S} \psi} = \textit{true}$ at τ^-, the table $\text{SINCE}_{\text{DOWN}}$ is used to determine the output transition.

For the *false* case, since $\gamma_{\phi \mathcal{S} \psi}$ will be *true* whenever γ_ψ is *true*, it is impossible for γ_ψ to exhibit any of the transitions in {¯, ↧, ↥, ↾}. As such, the table $\text{SINCE}_{\text{DOWN}}$ has no entries for these transitions on ψ. Also, if γ_ψ = _, $\gamma_{\phi \mathcal{S} \psi}$ must also be _ regardless of the transition type of γ_ϕ because if γ_ψ is *false* the only way for $\gamma_{\phi \mathcal{S} \psi}$ to be *true* is for γ_ϕ to be *true* and have been *true* since some time in the past at which γ_ψ was *true*. No such time can exists at time τ or τ^+ when γ_ψ is _ at time τ. The transition type _ is not legal at time $\tau > 0$, so $\text{SINCE}_{\text{DOWN}}$ has no entries for the row for which the γ_ψ is _. Since $\gamma_{\phi \mathcal{S} \psi}$ must be *true* whenever γ_ψ is *true* but can not become *true* at any point that γ_ψ is *false*, if γ_ψ is ↥ or ↾ $\gamma_{\phi \mathcal{S} \psi}$ will exhibit the same transition type. Similarly, when γ_ψ is ↧ at time τ, $\gamma_{\phi \mathcal{S} \psi}$ is guaranteed to be *false* at time τ^- and *true* at time τ. However, its truth at time τ^+ will depend upon the truth of γ_ϕ at τ^+because this satisfies the condition that ϕ is *true* and has been *true* since a point in time when ψ was *true*.

The case for which $\gamma_{\phi \mathcal{S} \psi}$ is *true* at τ^- uses SINCE_{UP} and is complicated enough that we will consider each transition type on γ_ψ separately:

For γ_ψ = _ we first have that γ_ϕ mustn't be in {_,¯} as one of the inputs must exhibit a transition. We also know that γ_ϕ must be *true* at τ^-, so the transitions in {↥,↾, ↧} are also impossible. Of those that remain, the output remains *true* until the first instant that γ_ϕ becomes *false*, and then remains *false* thereafter; that is, ↧ yields ↧, ↧ yields ↧, and ↾ yields ↧.

For γ_ψ = ¯ no transition on γ_ϕ can make $\gamma_{\phi \mathcal{S} \psi}$ false, so SINCE_{UP} lacks transitions for all transitions of γ_ϕ in this case.

For γ_ψ = ↥ the truth of γ_ψ at τ or τ^+ guarantees that $\gamma_{\phi \mathcal{S} \psi}$ will be *true* at those times. As we are considering cases for which $\gamma_{\phi \mathcal{S} \psi}$ is *true* at τ^-, SINCE_{UP} lacks transitions for all transitions of γ_ϕ in this case as well.

For γ_ψ = ↥ The truth of γ_ψ at τ^+ guarantees that $\gamma_{\phi \mathcal{S} \psi}$ will be *true* at τ^+ and the fact that $\gamma_{\phi \mathcal{S} \psi}$ is *true* at τ^- means that γ_ϕ must be *true* at τ^-. Of the transitions on γ_ϕ for which this is the case, any for which γ_ϕ is *true* at time τ, will yield no transition on the output. The only remaining transitions on γ_ϕ are ↧ and ↾, both of which yield ↾ because of the guarantee that $\gamma_{\phi \mathcal{S} \psi}$ be *true* at τ^+.

For $\gamma_\psi = \uparrow$ $\gamma_{\phi \mathcal{S} \psi}$ will also be $\uparrow$ unless the transition of γ_ϕ is *true*at time τ. If γ_ϕ is *true* at both times τ and τ^+, as is the case of $\overline{\ }$ and $\downarrow$, then the output will remain *true* at both times as well and thus will produce no output transition. If γ_ϕ is true at time τ but not at time τ^+, the output transition will be $\uparrow$.

By a similar argument, $\gamma_\psi = \uparrow$ will result in $\gamma_{\phi \mathcal{S} \psi}$ exhibiting the transition $\uparrow$, except for those transitions on γ_ϕ for which it is *true* for τ^+that result in no transition.

For $\gamma_\psi = \downarrow$, γ_ϕ must be *true* at a time τ^- so we can disregard all transitions for which this is not the case. The fact that γ_ψ is *true* at time τ guarantees that the output will also be *true* at time τ. Whenever γ_ϕ is *true* at time τ^+, $\gamma_{\phi \mathcal{S} \psi}$ will be satisfied at time τ^+. For those transitions on γ_ϕ for which this is not the case, $\uparrow$ and $\downarrow$, the output will be *false* at time τ^+, resulting in an output transition of $\downarrow$ in both cases.

For $\gamma_\psi = \updownarrow$, the output must be *true* at both time τ^- and τ^+. For any transition on γ_ϕ for which ϕ is *true* at time τ, the output will also be *true* at time τ resulting in no transition on the output. For all other transitions on γ_ϕ the output will be *false* at time τ, yielding an output transition of type $\updownarrow$.

Lemma 5. *The application of the tables* $\text{UNTIL}_{\text{INIT}_{a\text{-}c}}$ *and* $\text{UNTIL}_{a\text{-}d}$ *in the update procedure satisfies the* until *operation (5).*

Proof. The *until* transducer will enter an indeterminate state whenever $\gamma_\phi$is *true* but $\gamma_\psi$is *false*. The transition type, if any, emitted at the time at which this first becomes true is determined by whether or not γ_ψ becomes *true* before γ_ϕ becomes *false*. Thus, the monitoring algorithm must wait before emitting any transition until it is clear which is the case. The update procedure for the transducer uses two state variables to keep track of the potential outcomes of entering this state: $\delta_\uparrow$, for when γ_ψ becomes *true* before γ_ϕ becomes *false*, and $\delta_\downarrow$, for when it does not. Either of these transitions could be $\varnothing$ even when in the indeterminate state, but at least one of them will hold a potential transition – otherwise there would be no uncertainty about the transition type.

The rules for determining the transition types are as follows:

1. If γ_ψ is *true* at time τ^-, $\gamma_{\phi \mathcal{U} \psi}$ is *true* at time τ^-.
2. if γ_ψ is *true* at time τ, $\gamma_{\phi \mathcal{U} \psi}$ is *true* at time τ.
3. If γ_ψ is *true* at time τ^+, $\gamma_{\phi \mathcal{U} \psi}$ is *true* at time τ^+.
4. If γ_ϕ is *true* at time τ^-, $\gamma_{\phi \mathcal{U} \psi}$ is *true* at time τ^- if it is found to be true at time τ.
5. If γ_ϕ is *true* at time τ, $\gamma_{\phi \mathcal{U} \psi}$ is *true* at time τ if it is found to be true at time τ^+.
6. If γ_ψ is *false* at time τ^+ and γ_ϕ is *true* at time τ^+, $\gamma_{\phi \mathcal{U} \psi}$ is *true* at time τ^+ only if it is found to be *true* at time τ^-_{next} (i.e. immediately before the time of the next transition).
7. In all other cases, $\gamma_{\phi \mathcal{U} \psi}$ is *false*.

Base Case (Initial Transition)

For the initial transition, there is no need to consider the possibility of being in an indeterminate state before the first transition, so there are only three tables used. $\text{UNTIL}_{\text{INIT}_\text{a}}$ describes the transition type to be emitted immediately, but since it is possible that the initial transitions can result in an indeterminate state, $\text{UNTIL}_{\text{INIT}_\text{b}}$ and $\text{UNTIL}_{\text{INIT}_\text{c}}$ are used to produce $\delta_\uparrow$ and $\delta_\downarrow$, respectively.

By rule 6, when δ_ψ is _, $\delta_{\phi \mathcal{U} \psi}$ will be indeterminate for $\delta_\phi \in \{\overline{\ }, \downarrow\}$
By rule 7, when δ_ψ is _, $\delta_{\phi \mathcal{U} \psi}$ will be _ for $\delta_\phi \in \{\_, \uparrow\}$
By rules 1-3, when δ_ψ is $\overline{\ }$, $\delta_{\phi \mathcal{U} \psi}$ will be $\overline{\ }$.
By rules 3 and 7, when δ_ψ is $\downarrow$, $\delta_{\phi \mathcal{U} \psi}$ will be $\downarrow$ for $\delta_\phi \in \{\_, \downarrow\}$.

By rules 3 and 5, when δ_ψ is $\uparrow$, $\delta_{\phi\mathcal{U}\psi}$ will be $\overline{\ }$ for $\delta_\phi \in \{\overline{\ }, \downarrow\}$.
By rule 6, when δ_ψ is $\downarrow$, $\delta_{\phi\mathcal{U}\psi}$ will be indeterminate for $\delta_\phi \in \{\overline{\ }, \uparrow\}$
By rules 1,2, and 7, when δ_ψ is $\downarrow$, $\delta_{\phi\mathcal{U}\psi}$ will be $\downarrow$ for $\delta_\phi \in \{\_, \downarrow\}$

Initial indeterminate states

By rules 4 through 6, when δ_ψ is _ and δ_ϕ is $\overline{\ }$, $\delta_{\phi\mathcal{U}\psi}$ will be $\overline{\ }$ if $\gamma_{\phi\mathcal{U}\psi}$ is found to be *true* at time τ^-_{next}.
By rule 7 , when δ_ψ is _ and δ_ϕ is $\overline{\ }$, $\delta_{\phi\mathcal{U}\psi}$ will be _ if $\gamma_{\phi\mathcal{U}\psi}$ is found to be *false* at time τ^-_{next}.
By rules 7 and 6, when δ_ψ is _and δ_ϕ is $\uparrow$, $\delta_{\phi\mathcal{U}\psi}$ will be $\uparrow$ if $\gamma_{\phi\mathcal{U}\psi}$ is found to be *true* at time τ^-_{next}.
By rule 7, when δ_ψ is _and δ_ϕ is $\uparrow$, $\delta_{\phi\mathcal{U}\psi}$ will be $\downarrow$ if $\gamma_{\phi\mathcal{U}\psi}$ is found to be *false* at time τ^-_{next}.
By rules 1,2, and 6, when δ_ψ is $\downarrow$and δ_ϕ is in $\{\overline{\ }, \uparrow\}$, $\delta_{\phi\mathcal{U}\psi}$ will be $\overline{\ }$ if $\gamma_{\phi\mathcal{U}\psi}$ is found to be *true* at time τ^-_{next}.
By rules 1,2, and 6, when δ_ψ is $\downarrow$and δ_ϕ is in $\{\overline{\ }, \uparrow\}$, $\delta_{\phi\mathcal{U}\psi}$ will be $\downarrow$ if $\gamma_{\phi\mathcal{U}\psi}$ is found to be *false* at time τ^-_{next}.

Inductive Step (Next Transition)

Resolution of indeterminate transition

Table UNTIL$_{\text{a}}$ is used to determine which, if either, of the nondeterministic transitions was actually correct. By rule 6, the state of the prior transition can only be indeterminate if, for previous transition time τ_{prev}, γ_ϕ is *true* at τ^+_{prev} and γ_ψ is *false* at τ^+_{prev}. As such, the only possible transitions that could resolve the indeterminate state are those for which γ_ϕ is *true* at τ^+_{prev} and γ_ψ is *false* at τ^-.

When δ_ψ is _, none of the conditions in rules 1 or 4 apply for any of the three of the applicable transitions on γ_ψ so all result in the nondeterministic-down transition being selected.

When δ_ψ is $\uparrow$, rule 4 is satisfied for all applicable transitions on γ_ψ so all result in the nondeterministic-up transition being selected.

When δ_ψ is $\uparrow$, rule 5 – and therefore 4 – is satisfied for the transitions $\overline{\ }$ and $\downarrow$ on γ_ψ so both result in the nondeterministic-up transition being selected, but for transitions $\downarrow$ and $\updownarrow$, neither of the rules 1, 4 applies so both result in the nondeterministic-down transition being selected.

When δ_ψ is $\updownarrow$, rule 4 is satisfied for all applicable transitions on γ_ψ so all result in the nondeterministic-up transition being selected.

Production of determinate current transition

When δ_ψ is _, no rule provides a deterministic transition.

When δ_ψ is $\overline{\ }$, rules 1 through 3 guarantee that $\gamma_{\phi\mathcal{U}\psi}$ is true at times τ^-, τ, and τ^+. As $\overline{\ }$ is only legal for an initial transition, no transition is produced.

When δ_ψ is $\uparrow$, rules 2 and 3 guarantee that $\gamma_{\phi\mathcal{U}\psi}$ will be *true* at time τ and τ^+. For transitions in which δ_ϕ is *false* at time τ^-, rule 10 applies for time τ^-, yielding a transition of type $\uparrow$. For the other cases, the non-deterministic up will have been produced at time τ_{prev} and the current time will not produce any transition as the $\overline{\ }$ transition is illegal for any but the initial transition.

When δ_ψ is $\uparrow$, rule 3 guarantees that $\gamma_{\phi\mathcal{U}\psi}$ will be *true* at time τ^+. If δ_ϕ is *false*at time τ, rule 10 applies at times τ^- and τ, yielding a transition of type $\uparrow$. For those cases in which δ_ϕ is *false* at τ^-, but *true* at τ, rule 5 results in a transition of type $\uparrow$. For the remaining cases, δ_ϕ is *true* at

both times τ^- and τ, these will both have selected a non-deterministic up transition at time τ_{prev}, resulting in the illegal transition $\overline{}$.

When δ_ψ is $\uparrow$, a determinate current transition can only be produced if δ_ϕ is *false* at τ^+. In these cases, rules 4 through 6 can not apply, so rules 1 and 10 give that all transitions are of type $\uparrow$.

When δ_ψ is $\downarrow$, a determinate current transition can only be produced if δ_ϕ is *false* at τ^+. In these cases, rules 1, 2, and 10 give that all transitions are of type $\downarrow$.

When δ_ψ is $\updownarrow$, a determinate current transition can only be produced if δ_ϕ is *false* at τ^+. If δ_ϕ is *true* at τ^-, rules 2, 4, and 10 yield a transition of type $\downarrow$. Otherwise, rules 2 and 10 yield a transition of type $\updownarrow$.

Production of indeterminate transitions

Tables UNTIL_c and UNTIL_d yield the nondeterministic up and down transitions, respectively. By rule 6, the state of the current transition can only be indeterminate if γ_ϕ is *true* at τ^+ and γ_ψ is *false* at τ^+.

When δ_ψ is $\_$, δ_ϕ must be a real transition, that is not $\overline{}$. Of those transition types, previous arguments show that $\gamma_{\phi \mathcal{U} \psi}$ will have been found to have been *false* at τ^+_{prev} and will thus $\delta_{\phi \mathcal{U} \psi}$ be *false* at τ^-, so only nondeterministic up transitions can be produced. By definition these will be *true* at time τ^+. Rules 5 gives the the truth of $\delta_{\phi \mathcal{U} \psi}$ as equal to that of γ_ϕ at time τ.

When δ_ψ is $\uparrow$, rule 1 gives that $\delta_{\phi \mathcal{U} \psi}$ must be *true* at time τ^-. Since a transition of $\overline{}$ is illegal, the only possible nondeterministic up transition is $\uparrow$. It will be produced whenever δ_ϕ is *false* at time τ. All nondeterministic down transitions will be of type $\downarrow$ because rule 5 can not be satisfied for a down transition.

When δ_ψ is $\downarrow$, rules 1 and 2 give that $\delta_{\phi \mathcal{U} \psi}$ must be *true* at times τ^- and τ. As such, no legal up transition can be produced and the only possible down transition is of type $\downarrow$.

When δ_ψ is $\updownarrow$, rules 2 gives that $\delta_{\phi \mathcal{U} \psi}$ must be *true* at time τ. If δ_ϕ were *true* at τ^-, rule 4 would give that $\gamma_{\phi \mathcal{U} \psi}$ is *true* at τ^- and yield the illegal $\overline{}$ transition. For the other indeterminate cases, a nondeterministic up transition of type $\updownarrow$ will be produced. The nondeterministic down transitions are similarly constrained. If δ_ϕ is *true* at time τ^-, the transition type will be $\downarrow$, otherwise it will be $\updownarrow$.

B Complexity

This section proves the runtime and space complexity of the monitoring algorithm as applied to significant subsets of MTL: instantaneous transducers, limited-future transducers, and unlimited-future transducers (full MTL). The following proofs make use of the following notations. When monitoring a property for which the formula is ϕ, $|\phi|$ indicates the number of operators in the formula. $|\gamma|$ is the number of transitions on all input signals.

We begin with a few lemmata that are valid for all subsets and are used in the proofs for the specific subsets of MTL.

Lemma 6. *The* ENQUEUE *procedure for runs in constant time for each input transition.*

Proof. The ENQUEUE procedure contains no loops and simply performs a number of comparison and queue operations. Given that appending to, emptying, and checking the emptiness of a queue can be implemented to execute in constant time, the ENQUEUE operation itself executes in constant time.

Lemma 7. *The* DEQUEUE *procedure for runs in constant time for each output transition.*

Proof. The DEQUEUE procedure contains no loops and simply extracts the first transition from the queue and stores the constant value associated with the extracted transitions truth at time τ^+. Given that extracting the first element of a queue can be implemented to execute in constant time, the DEQUEUE operation itself executes in constant time.

Lemma 8. *The* SYNC *procedure for runs in constant time for each output transition.*

Proof. The SYNC procedure contains no loops. Lines 3 through 8 handle the single-input case. As the *eventually* operator has only one input, it ensures that the next transition event is before or simultaneous with the current timer. To do so, it uses a new operation HEAD which simply returns the content of the head of the queue without actually removing it. It is taken as given that this can be implemented to run in constant time. Lines 10 though 29 handle the two input case. It introduces no new operators and simply compares valid intervals to determine which, if any, input transitions should be dequeued. In each case, all operations execute in constant time, so the SYNC operation does so as well.

Lemma 9. *The update procedure for negation runs in constant time for each input transition.*

Proof. For each input transition, exactly one output transition is produced either on line 3 or 5 depending on the value of the timestamp τ. The type of each output transition is determined by a table lookup that is accomplished in constant time and supplied as an argument to an ENQUEUE operation that Lemma 6 gives as executing in constant time.

The UPDATEVALIDINTERVAL for negation simply copies the valid interval – guaranteed to be contiguous – of its input and thus executes in constant time.

Lemma 10. *The length, measured in number of transitions, of the negation of a signal is equal to that of the signal itself.* $|\gamma_{\neg p}| = |\gamma_p|$

Proof. As described in the previous proof, for each input transition, exactly one output transition is produced either on line 3 or 5 depending on the value of the timestamp τ.

Lemma 11. *The update procedure for conjunction runs in constant time for each pair of input transitions.*

Proof. For each pair of input transitions, exactly one output transition is produced either on line 3 or 5 depending on the value of the timestamp τ. The type of each output transition is determined by a table lookup that is accomplished in constant time and supplied as an argument to an ENQUEUE operation that Lemma 6 gives as executing in constant time.

The UPDATEVALIDINTERVAL for conjunction simply computes the intersection of the valid intervals of its inputs. As they are guaranteed to be contiguous, this requires only the comparison of the lower and upper limits of each and thus executes in constant time.

Lemma 12. *The length of the conjunction of two signals is less than or equal to the sum of the lengths of the input signals.* $|\gamma_{p \wedge q}| \leq |\gamma_p| + |\gamma_q|$

Proof. The longest series of transitions occurs when no two input transitions are simultaneous. In this case, each input transition results in one output transition. If any two input transitions are simultaneous, the pair will result in only one output transition.

Lemma 13. *The update procedure for the* since *operator runs in constant time for each pair of input transitions.*

Proof. For each pair of input transitions, exactly one output transition – or non-transition – is produced either on line 3, 5, or 7 depending on the value of the timestamp τ and the current state of the *since* transducer. The type of each output transition is determined by a table lookup that is accomplished in constant time and – if it is not null – supplied as an argument to an ENQUEUE operation that Lemma 6 gives as executing in constant time and used to determine the next state of the transducer based on the transition type.

The UPDATEVALIDINTERVAL for *since* simply computes the intersection of the valid intervals of its inputs. As they are guaranteed to be contiguous, this requires only the comparison of the lower and upper limits of each and thus executes in constant time.

Lemma 14. *the length of the output of the* since *operator is less than or equal to the length of the input signal for the second argument.* $|\gamma_{p \mathcal{S} q}| \leq |\gamma_q|$

Proof. This is the result of the dominance of the second argument on the *since* operation. The formula $p \mathcal{S} q$ is *true* whenever q is *true*. Thus, $|\gamma_{p \mathcal{S} q}|$ can be equal to $|\gamma_q|$. If p is *true* at all times, $\gamma_{p \mathcal{S} q}$ will also be *true* at all times and then $|\gamma_{p \mathcal{S} q}| = 1$ regardless of $|\gamma_q|$. $\gamma_{p \mathcal{S} q}$ can only become *true* at a point in time that γ_q is *true*. The truth of γ_p can only keep $\gamma_{p \mathcal{S} q}$ *true* after γ_q has become *false*, and it will only do so until γ_p becomes *false* for the first time after a down transition on γ_q. Thus, γ_p can only delay a down transition that would have resulted from a down transition on γ_q or remove a pair of transitions (down then up) on γ_q if γ_p remains *true* between them.

Lemma 15. *The update procedure for the* eventually *operator runs in constant time for input or timer expiration.*

Proof. For each input transition or timer expiration, At most one call to the ENQUEUE operation is made on a transition type obtained from a table lookup. In some cases, the $\delta_{\downarrow}$ and τ_{timer} may be updated as many as two times. As all of these operations execute in constant time, the entire update procedure does so as well.

The UpdateValidInterval for *eventually* simply updates its interval based on that of its input with an offset to the maximum extent. The switch simply ensures that the inclusion of the maximum extent matches that of the input interval.

Lemma 16. *the length of the output of the* eventually *operator is no more than twice the length of its input.* $|\gamma_{\Diamond_I p}| \leq 2|\gamma_p|$

Proof. The introduction of the timer allows that an event can occur at a time for which there is no input. However, this can only take place after a ↿, ⇂, or ↥ transition for which no ↾, ⇃, or ↥ occurs within $|I|$ time units. If such an input transition occurs within $|I|$ time units, neither transition will produce an output transition. No ↿ input transition ever produces an output transition. Thus, the maximum number of output events produced by an up event (↾ or ⇃) followed by a down event (↿ or ⇂) is two – no net gain. Only the ↥ transition can produce more than one output transition. When it is not followed by an up transition within $|I|$ time units, it produces exactly one up transition and one down transition. As such, the worst case behavior is achieved when all input events are ↥ transitions separated by greater than $|I|$ time units and it will be precisely twice the length of the input.

Lemma 17. *The length of the output of cascaded* eventually *operators is also not more than twice the length of the input.*

Proof. As described above, only the ↥ transition can produce more than one output transition. The types of the two output transitions that it produces are exactly one transition in {↾, ⇃} and one transition in {↿,⇂}. Neither of these transitions will result in more than one transition in subsequent *eventually* transducers. As such the length of the output of a sequence of *eventually* operators is less than or equal to twice the length of the input to the sequence of *eventually* operators.

Lemma 18. *The runtime cost of the introduction of clocks is asymptotically linear with product of the size of the formula,* $|\phi|$*, and the size of the input,* $|\gamma|$*.*

Proof. The number of clocks is limited by the number of *eventually* operators in the formula, $|\phi_{\Diamond_I}|$, as each can only maintain one clock at any given time. Each *eventually* operator can produce at most one clock per input event. Determining the next clock event is done as part of the update procedure pass over all transducers. Its runtime cost is constant with each transducer, so the cost is in $O(|\phi|)$. Thus the additional overhead associated with clocks is limited to $O(|\gamma| \cdot |\phi| + |\gamma| \cdot |\phi_{\Diamond_I}|)$ which is in $O(|\gamma| \cdot |\phi|)$.

Lemma 19. *The update procedure for the* until *operator runs in constant time for each pair of input transitions.*

Proof. For each input transition or timer expiration, at most two calls to the Enqueue operation are made on a transition type obtained from a table lookup, both $\delta_\uparrow$ and $\delta_\downarrow$ are set as is the τ_{pending}. As all of these operations execute in constant time, the entire update procedure does so as well.

In the worst case, the UpdateValidInterval for *until* simply computes the intersection of the valid intervals of its inputs. As they are guaranteed to be contiguous, this requires only the comparison of the lower and upper limits of each and thus executes in constant time. If either $\delta_\uparrow$ or $\delta_\downarrow$ is set, even this action is not taken.

Lemma 20. *the length of the output of the* until *operator is less than or equal to the length of the input signal for the second argument.* $|\gamma_{p\mathcal{U}q}| \leq |\gamma_q|$

Proof. This follows the same logic as the proof of Lemma 14. The output must be *true* at all points for which γ_q is *true* but the truth of γ_p can only serve to reduce the number of transitions on the output if it remains *true* between two intervals during which γ_q is *true*.

B.1 Instantaneous Transducers

The instantaneous operators are: negation, conjunction, since, and the metric eventually, $\Diamond_I$ for which the upper bound of the interval I is zero. They all have the property that every output transition that happens in response to an input transition at time τ will take place at time τ. This simplifies the ENQUEUE procedure because it is never necessary to store more than one transition on the output of any transducers, the one for the current time. Similarly, DEQUEUE will operate on a single value simply storing the appropriate ‾ or _ transition in the single storage cell for the transducer's output after removing the stored transition. Each of these operations takes constant time.

Theorem 2. *The runtime to monitor expression ϕ that consists of only instantaneous operators on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

Proof. The naïve update procedure presented in this paper consists of visiting each transducer and running its update procedure to determine whether it can produce output based on the input that has arrived for each input transition. Assuming the worst case input – that for which no input transitions are simultaneous – this establishes a floor of $|\phi|n$ update operations. The timers of the *eventually* operator can cause the update procedure to be called at points for which there are no input transitions. In the worse case, all of the operators in ϕ are *eventually* operators and the input and timer expirations are such that no two events are simultaneous, this gives a maximum number of transducer update procedure calls of $2|\phi|n$. As lemmas 9, 11, 13, and 15 give that the update for each transducer runs in constant time, this yields an asymptotic runtime of $\mathrm{O}(|\phi|n)$.

Theorem 3. *The space required to monitor expression ϕ that consists of only instantaneous operators on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|)$.*

Proof. The monitor for any formula consists of a tree formed by transducers. The leaves of this tree represent the input signals. Assuming that we maintain a separate queue for each transducer input that comes from an external signal we have a maximum of $|\phi| + 1$ input queues – produced when all operators are binary. Each transducer output also requires a queue for a total of maximum of $2|\phi|+1$. At any point in time τ the valid interval for all input signals will be $[0, \tau]$. For instantaneous transducers, this results in valid intervals of the transducer outputs that are also always of the form $[0, \tau]$. Inspection of the SYNC procedure shows that if all inputs and transducers have valid intervals that include the current time, then no transition on any queue with timestamp less than or equal to τ will be left on the queue pending some future event on the other input. That is, any external input transitions, with timestamp τ = now, will be immediately consumed and the output of each transducer will also have a timestamp of τ and will thus be immediately consumed by subsequent transducers. Consequently, no queue will ever contain more than one transition. Thus the size of all queues must be in $\mathrm{O}(|\phi|)$. Some transducers also maintain state information, but it is of a constant size; The *eventually* operator maintains a timer and *since* maintains a boolean state. The size of all such storage will be in $\mathrm{O}(|\phi|)$, so the entire update procedure requires storage that is in $\mathrm{O}(|\phi|)$.

B.2 Strictly Past Transducer

The transducer for the operator $\Diamond_{[a,b]}$ for $a <= b < 0$ operates identically to that of $\Diamond_{[b-a,0]}$ except that the timestamp of the output that results from an event at time τ is $\tau - b$. The corresponding statement is true for other intervals with open bounds as well. Since b is defines as being less than zero, this output will take place at a time that is in the future. This introduces the need to maintain valid ranges and queues to store the output of the intermediate stages to support operators with multiple inputs as we may need to wait $-b$ time units .

Theorem 4. *The runtime to monitor expression ϕ that consists of only past-time operators on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the number of transitions on all inputs with timestamp less than or equal to τ.*

Proof. This follows from the proof of the runtime for instantaneous transducers.

Lemma 21. *For the output of the metric eventually operator of the form $\Diamond_{[a,b]}$, the duration of any interval – other than those beginning at time zero – during which it is* true *must be $b - a$ time units. More formally, $\forall i : (\delta_i, \tau_i) \in \gamma_{\Diamond_{[a,b]}}, \delta_i \in \{\downarrow, \updownarrow, \uparrow\} \rightarrow \ \tau_{i+1} - \tau_i \geq \ b - a$.*

Proof. First note that all up transitions, produced on lines 4, 7, or 13 of UPDATE$_{MTLFI}$, are the result of an input up transition at time τ and are produced at time $\tau - b$. The only way to produce a down transition is to have set a timer in response to an input transition on lines 17 and 18. The shortest-duration interval between an up transition and its subsequent down transition will occur when the input both causes an up transition and results in the setting of a timer to produce a down transition. For such an input transition that takes place at time τ, Its resulting up transition will have a timestamp of $\tau' = \tau - b$ and a timer will be set for time $\tau + b - a$. When that timer expires, and the update procedure produces a down transition, it will do so with a timestamp $\tau'' = (\tau + b - a) - b = \tau - a$ The difference between the timestamps of these events will be $\tau'' - \tau' = (\tau - a) - (\tau - b) = b - a$.

The reason that an interval beginning at time zero might be of shorter duration than $b - a$ is that for if $b > 0$, the timestamps produced by the *eventually* operator might be negative and lines 3 through 8 of the ENQUEUE procedure will suppress their production – effectively only emitting the portion of the *true* interval for which time timestamps are non-negative.

Lemma 22. *The output of cascaded* eventually *operators of the form $\Diamond_{[a_1,b_1]}$ followed by $\Diamond_{[a_2,b_2]}$ is equivalent to that of $\Diamond_{[a_1+a_2,b_1+b_2]}$.*

Proof. From Figure 2, we have

$$\hat{\gamma}, \tau \models \Diamond_I \ \phi \qquad \text{iff} \quad \exists\ \tau' \text{such that } \tau' - \tau \in \ I,\ \hat{\gamma}, \tau' \models \ \phi$$

which, for cascaded *eventually* operators gives

$$\hat{\gamma}, \tau \models \Diamond_{[a_1,b_1]}\ \Diamond_{[a_2,b_2]}\ \phi \quad \text{iff} \quad \exists\ \tau', \tau'' \text{such that } \tau' - \tau \in [a_2, b_2] \text{ and } \tau'' - \tau' \in [a_1, b_1],\ \hat{\gamma}, \tau'' \models \ \phi$$

The constraints can be rewritten as

$$\begin{array}{lll} a_1 \leq \tau'' - & \tau' & \leq b_1 \\ a_2 \leq & \tau' - \tau & \leq b_2. \end{array}$$

Summing these lines yields

$$a_1 + a_2 \leq \tau'' - \tau \ \leq b_1 + b_2,$$

which gives

$$\begin{array}{lll} \hat{\gamma}, \tau \models \Diamond_{[a_1,b_1]} \Diamond_{[a_2,b_2]} \phi & \text{iff} & \exists\ \tau'' \text{such that } \tau'' - \tau \in [a_1 + a_2, b_1 + b_2],\ \hat{\gamma}, \tau'' \models \phi \\ \hat{\gamma}, \tau \models \Diamond_{[a_1+a_2,b_1+b_2]} \phi & \text{iff} & \exists\ \tau'' \text{such that } \tau'' - \tau \in [a_1 + a_2, b_1 + b_2],\ \hat{\gamma}, \tau'' \models \phi \end{array}$$

Theorem 5. *The space required to monitor expression ϕ that consists of only past-time operators with $\Diamond_{[a,b]}$ where $a < b < 0$ on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi| \left\lfloor \frac{|b|}{b-a} \right\rfloor)$.*

Proof. The increased space requirement comes from the fact that the timestamp of the output produced by the *eventually* transducer in response to an event (transition or timer expiration) at time τ will have timestamp $\tau - b$. Since b is known to be negative in this case, these events take place at a future time compared to other inputs to the monitor. Subsequent binary transducers, such as conjunction, must wait until the valid intervals of both of its inputs contain the timestamp of any input transition before it can be acted upon. For a concrete example, consider

$$p \wedge \Diamond_{[a,b]} q.$$

In this case, the output queue of the *eventually* operator will have to store all transitions until the SYNC operation finds that the valid interval of the p input contains their timestamp. As the p input's valid interval will always be $[0, \tau_{\text{current}}]$. That is, it will potentially store as many transitions as the *eventually* operator can produce within $|b|$ time units. Lemma 21 gives a minimum width of $b - a$ for any interval for which the output of the *eventually* operator is *true*. The caveat about intervals beginning at time zero does not apply to the past-time version, so we know that there can be only $\left\lfloor \frac{|b|}{b-a} \right\rfloor$ such intervals. The maximum number of transitions that this could entail would be $2 \left\lfloor \frac{|b|}{b-a} \right\rfloor$, but this is not important to the asymptotic performance. The worst case storage is achieved when the number of mismatched input valid intervals is maximized which arises in the formula

$$\bigwedge_{i=0}^{k} p_i \wedge \Diamond_{[a,b]} q_i.$$

This produces $2k$ conjunction operators and k eventually operators, thus the number of eventually operators is in $\Theta(|\phi|)$ giving a worst-case space complexity in $\Theta(|\phi| \left\lfloor \frac{|b|}{b-a} \right\rfloor)$.

Cascading *eventually* operators of the form $\Diamond_{[a,b]}$ does not increase this asymptotic behavior because Lemma 22 gives that this is equivalent to a single *eventually* operator $\Diamond_{[2a,2b]}$. The space requirement for that case would be in $\mathrm{O}(|\phi| \left\lfloor \frac{2|b|}{2(b-a)} \right\rfloor) = \mathrm{O}(|\phi| \left\lfloor \frac{|b|}{b-a} \right\rfloor)$.

For formulas containing *eventually* operators with different intervals, it suffices to consider only the *eventually* operator for which the ratio of $|b|$ to $b-a$ is maximized. This is because the ratio for the cascaded *eventually* operators must be less than that of one of its inputs. This can be shown by contradiction. To simplify the presentation, we will introduce $n_1 = |b_1|$ and $d_1 = b_1 - a_1$ as the numerator and denominator of the ratio for the first of the *eventually* operators and n_2 and d_2 for the second. To show the contradiction, we assume that the cascaded *eventually* operators do entail a higher space complexity than either of the individual *eventually* operators alone.

$$\begin{aligned} \frac{n_1}{d_1} &< \frac{n_1+n_2}{d_1+d_2} \\ n_1(d_1+d_2) &< d_1(n_1+n_2) \\ n_1d_1+n_1d_2 &< d_1n_1+d_1n_2 \\ n_1d_2 &< d_1n_2 \\ \frac{n_1}{d_1} &< \frac{n_2}{d_2} \end{aligned} \qquad \begin{aligned} \frac{n_2}{d_2} &< \frac{n_1+n_2}{d_1+d_2} \\ n_2(d_1+d_2) &< d_2(n_1+n_2) \\ n_2d_1+n_2d_2 &< d_2n_1+d_2n_2 \\ n_2d_1 &< d_2n_1 \\ \frac{n_2}{d_2} &< \frac{n_1}{d_1} \end{aligned}$$

which establishes the contradiction.

Theorem 6. *The space required to monitor expression ϕ that consists of only past-time operators with $\Diamond_{[a,a]}$ where $a < 0$ on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

Proof. The transducer for this case is not implemented according to the algorithm described for the general case. For each input transition (δ, τ), the corresponding output is $(\delta, \tau - a)$. A time-shifted version of the input. Again, as $a < 0$, this new input will be in the future. However, unlike the previous case, there is nothing to limit the number of transitions that can occur within those $|a|$ time units. In the worst case, a formula such as

$$\bigwedge_{i=0}^{k} p \wedge \Diamond_{[a,a]} p$$

for which all n input takes place within $|a|$ time units. The queues will have to contain $k \cdot n$ unprocessed transitions at some point. As k grows, $|\phi|$ grows linearly, so the worst case has a space complexity of $\mathrm{O}(|\phi|n)$.

B.3 Restricted Future

The next increment in monitor complexity introduces the metric eventually operator $\Diamond_{[a,b]}$ with $b > 0$. From the UPDATE procedure, we see that it introduces a delay in its output events relative to the input events that produces them. This adds no computational complexity, but reduces the guarantees that can be made about space complexity even when $a \neq b$.

Theorem 7. *The runtime to monitor expression ϕ that consists of past-time and restricted-future operators on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

Proof. This follows from the proof of the runtime for instantaneous transducers.

Theorem 8. *The space required to monitor expression ϕ that consists of only past-time operators with $\diamondsuit_{[a,b]}$ where $a <= b$ and $b > 0$ on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

Proof. With b greater than zero, the valid interval for the $\diamondsuit_{[a,b}$ operator will be b time units earlier than that of its input. In the expression $p \wedge \diamondsuit_{[a,b]} q$, the queue containing the transitions of the p input will have to store all transitions that have taken place within the last b time units because of the behavior of the SYNC procedure on the inputs to the conjunction operator. There is no restriction on the number of transitions that can take place on the input p within b time units, so the queue can potentially contain all n transitions. As with the previous proof, formulas of the form $\bigwedge_{i=0}^{k} p \wedge \diamondsuit_{[a,a]} p$ can produce a worse-case space complexity of $\mathrm{O}(|\phi|n)$.

B.4 Unrestricted Future

Theorem 9. *The runtime to monitor expression ϕ that consists of past-time and restricted-future operators on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

Proof. The the introduction of the unrestricted-future expands the monitoring capability to full MTL with the introduction of the *until* operator. Lemma 20 gives that the addition of the *until* operator doesn't entail any increase in the number of output transitions over those of the input. Lemma 19 gives that each call to the update procedure for the *until* operator executes in constant time. With the addition of these, the proof follows the form of that for the instantaneous transducers.

Theorem 10. *The space required to monitor expression ϕ that consists of only past-time operators with $\diamondsuit_{[a,b]}$ where $a <= b$ and $b > 0$ on input signals $\hat{\gamma}$ from time zero until time τ is in $O(|\phi|n)$, for n = the sum of the number of transitions on all inputs with timestamp less than or equal to τ.*

Proof. This proof follows from that of Theorem 8, but using an expression of the form $p \wedge r\mathcal{U}\, q$. The worst-case runtime is the same, but the inability to limit the space complexity is somewhat stronger as the valid interval of the *until* operator can have an upper limit that is an arbitrary amount of time difference from the current time. Even if there were some limit that could be placed on the number of input transitions within a fixed period, there would still be no limit on the worst-case space complexity for full MTL.