Computer Science at Kent

From LIDL$^-$ to Timed Automata

Rodolfo Gómez

Technical Report No. 02-09
April 2009

Published by the Computing Laboratory,
University of Kent, Canterbury, Kent, CT2 7NF, UK

From LIDL$^-$ to Timed Automata*

Rodolfo Gómez

Computing Laboratory, University of Kent, United Kingdom
rsg5@kent.ac.uk

July 10, 2009†

Abstract

LIDL$^-$ is a decidable fragment of Interval Duration Logic with Located Constraints, an expressive subset of dense-time Duration Calculus. It has been claimed that, for any LIDL$^-$ formula D, a timed automaton can be constructed which accepts the models of D. However, the proposed construction is incomplete and has not been proved effective. In this paper, we prove the effective construction of equivalent timed automata from LIDL$^-$ formulae.

1 Introduction

LIDL$^-$ [21] is a decidable fragment of Interval Duration Logic with Located Constraints, an expressive propositional subset of dense-time Duration Calculus (DC) [32, 31]. LIDL$^-$ is interpreted on finite timed state sequences [14, 4]. The decidability of LIDL$^-$ is obtained by disallowing the DC duration operator ($\int$) and restricting timing constraints to a particular form, called *located constraints*. A located constraint is a formula of the form $P \leadsto \ell \sim c$, where P is the *anchor* proposition, ℓ denotes the elapsed time in a sequence, $\sim$ is a relational operator and $c \in \mathbb{N}$ is a constant. The LIDL$^-$ formula $P \leadsto \ell \sim c$ holds in a sequence where the time elapsed between the last time a (previous) state satisfying P was entered, and the time the current state was entered, satisfies the constraint ($\sim c$). LIDL$^-$inherits all other DC operators, such as the *chop* operator ($\frown$) [19, 26] and quantification over propositional variables. This allows LIDL$^-$ to express a large and practically relevant class of timing constraints (for instance, most of the well-known requirements patterns of [25] can be naturally expressed in LIDL$^-$). In addition, LIDL$^-$ includes a past operator, $\overleftarrow{\Diamond}$ (*"sometime in the past"*), which extends the satisfiability of LIDL$^-$ formulae to previously observed states in the sequence.

A translation from LIDL$^-$ to deterministic event-recording timed automaton (ERA) [2] is outlined in [21]. (Moreover, LIDL$^-$ and ERA are claimed to be equally expressive.) Given an LIDL$^-$ formula D, each located constraint in D is replaced by an "untimed" formula that preserves the state-based semantics but omit timing constraints. This formula includes a fresh propositional variable that witnesses the last state of the constrained sequence, and will help to recover the timing constraint in the final automaton. It is claimed that, this reduction yields a formula $\mathcal{D}$ in Quantified Discrete Duration Calculus (QDDC) [22], for which it is known that an equivalent finite state automaton (FSA), $\mathcal{A}(\mathcal{D})$, can be effectively constructed (the alphabet of $\mathcal{A}(\mathcal{D})$ is the set of all different valuations for propositional variables in $\mathcal{D}$, and the language accepted by $\mathcal{A}(\mathcal{D})$ is the set of state sequences satisfying $\mathcal{D}$).

*This research has been supported by the UK Engineering and Physical Sciences Research Council under grant EP/D067197/1.

†This is a revised version of the original paper (April, 2009).

Finally, [21] proposes to convert the $\mathcal{A}(\mathcal{D})$ into an ERA $A(D)$, by recovering the timing constraints which were lost during the untimed reduction to QDDC. This is achieved by adding one clock x_i for each located constraint $P_i \rightsquigarrow \ell \sim_i c_i$ in D $(i : 1..n)$, by resetting x_i whenever P_i holds, and by testing $x_i \sim_i c_i$ whenever the last state of a constrained (sub)sequence is found in the input word (the witnesses added during the untimed reduction help to identify the transitions where clocks must be tested).

The existence of $A(D)$ implies the decidability of LIDL$^-$, because the reachability problem is decidable for ERA [3, 2]. Thus, [21] lays the foundation for automatic verification of LIDL$^-$ properties via observers [20], and contributes to a line of research that has identified decidable subsets of DC that are suitable for model-checking [6, 15, 29, 17]. However, the construction of $A(D)$, as described in [21], is incomplete and has not been proved effective. In particular, no procedure was given to identify the transitions where clocks must be reset. We note that, this cannot be trivially inferred from the structure of the $A(D)$, because the anchor propositions (P_i, $i : 1..n$) may refer to previous states in the input sequence. In addition, the past operator $\overleftarrow{\Diamond}$ is not defined in QDDC. Thus, the untimed formula, which is obtained after substitution of located constraints, *is not* a QDDC formula. Moreover, no trivial characterization of $\overleftarrow{\Diamond}$ seems to exist in terms of QDDC operators and (currently) there are no available tools that can automatically generate the FSA $\mathcal{A}(\mathcal{D})$. (As a final remark, by definition, $A(D)$ is not an ERA. For each symbol a in the alphabet of an event-recording automaton, a clock x_a is given which measures the time since the last occurrence of a in the input word [3]. Instead, clocks in $A(D)$ are associated with located constraints, s.t. x_i measures the time since the last position in the input word where P_i was true. Each P_i is actually a LTL [23, 16] formula on the automaton's alphabet, and thus $A(D)$ should be considered a *formula-recording* timed automaton [2, page 8]. Nonetheless, the class of formula-recording timed automata is a generalization of ERA, which preserves closure under Boolean operations (hence, both classes are determinizable), and preserves the decidability and complexity of the reachability problem.)

Contribution. We prove the effective construction of equivalent timed automata from LIDL$^-$ formulae. We observe that, the extension of QDDC with $\overleftarrow{\Diamond}$, QDDC$_{\overleftarrow{\Diamond}}$, can be easily encoded in Weak Second Order Theory of 1 Successor (WS1S) [7, 8, 30]. Hence, we know that equivalent FSA can be effectively constructed (future versions of the DCVALID tool [22], which currently translates pure QDDC formulae to automata, could support this extension). Following the approach in [21], we define an untimed reduction from LIDL$^-$ to QDDC$_{\overleftarrow{\Diamond}}$. However, in contrast to [21], we define a QDDC$_{\overleftarrow{\Diamond}}$ formula that includes witnesses for both, the states where anchors are true and the final states of constrained sequences. In this way, we are able to construct an equivalent FSA that allows a simple and effective translation to deterministic timed automata (in particular, in the class of formula-recording automata that we mentioned before).

We also introduce the logic LIDL$^-_\Delta$, a variant of LIDL$^-$ where the use of $\overleftarrow{\Diamond}$ is disallowed and timing constraints take the form of *future* located constraints. Future located constraints are formulae of the form $P\Delta_{\sim c}$, which define sequences where P holds in the first state but nowhere else except possibly in the last state, and where the time elapsed in the sequence satisfies the constraint $(\sim c)$. With trivial modifications, the translation from LIDL$^-$ to timed automata applies also to LIDL$^-_\Delta$. Thus, we prove that LIDL$^-_\Delta$ is decidable. However, in contrast to LIDL$^-$, the absence of $\overleftarrow{\Diamond}$ and past located constraints allows LIDL$^-_\Delta$ to be reduced directly to pure QDDC, and thus, the DCVALID tool [22] can be used to generate the equivalent FSA. Moreover, and at least for the class of timing constraints that are commonly found in practice, LIDL$^-_\Delta$ seems as expressive as LIDL$^-$.

Outline. Section 2 recalls the syntax and semantics of LIDL$^-$. Section 3 proves the decidability of LIDL$^-$. Section 4 introduces the logic LIDL$^-_\Delta$. We conclude this paper in Section 5, with a comparative discussion on the expressive power of LIDL$^-$, LIDL$^-_\Delta$, and other related logics.

2 The logic LIDL$^-$

This section introduces the syntax and semantics of LIDL$^-$, as originally defined in [21].

Preliminaries. Let $\mathbb{P}$ be a set of propositional variables. A state over $\mathbb{P}$ is an element of $2^{\mathbb{P}}$. A finite timed state sequence over $\mathbb{P}$ of length $n > 0$ ($n \in \mathbb{N}$) is a sequence of pairs $\theta = (s_0, t_0) \ldots (s_{n-1}, t_{n-1})$, where (1) $\sigma(\theta) = s_0 s_1 \ldots s_{n-1}$ is a state sequence, s.t. $s_i \in 2^{\mathbb{P}}$ for all $i : 0..n-1$; (2) $\tau(\theta) = t_0 t_1 \ldots t_{n-1}$ is a sequence of time instants, s.t. $t_i \in \mathbb{R}^{+0}$ for all $i : 0..n-1$, $t_0 = 0$ and $t_i \leq t_{i+1}$ for all $i : 0..n-2$; and (3) a unique state $\sigma(t)$ is identified with every instant $t \in \mathbb{R}^{+0}$, s.t. $\sigma(t) = s_i$ for all $t_i \leq t < t_{i+1}$, $i : 0..n-2$, and $\sigma(t) = s_{n-1}$ for all $t_{n-1} \leq t$.

Let θ be a timed state sequence of length n, as described above. The length of θ is denoted $|\theta|$; the i^{th} element of θ is denoted $\theta_i = (s_i, t_i)$; the set of intervals of θ is denoted $I(\theta) = \{\, [i, j] \mid i, j : 0..n-1, i \leq j \,\}$ and subsequences of θ are denoted $\theta_{[i,j]} = \theta_i \ldots \theta_j$, for any $[i, j] \in I(\theta)$. This notation extends to sequences of states and time instants. We use $p \in \theta_i$ to denote $p \in s_i$, for any $p \in \mathbb{P}$. The elapsed time in a subsequence $\theta_{[i,j]}$ is given by $\delta(\theta_{[i,j]}) = t_j - t_i$.

Let V be any set of propositional variables. Let s be a state, $\omega = \omega_0 \ldots \omega_n$ be a state sequence, and $\mathcal{L}$ a set of state sequences. The projection of s, ω and $\mathcal{L}$ over V is given by (resp.) $[s]_V = s \cap V$, $[\omega]_V = [\omega_0]_V \ldots [\omega_n]_V$ and $[\mathcal{L}]_V = \{\, [\omega]_V \mid \omega \in \mathcal{L} \,\}$. These definitions extend to timed state sequences.

Syntax and semantics. A proposition P over $\mathbb{P}$ is defined by the grammar:

$$P ::= \mathbf{0} \mid \mathbf{1} \mid p \mid \neg P \mid P \wedge P \mid \ominus P$$

where $p \in \mathbb{P}$. A proposition is interpreted over a timed state sequences, θ, and a position $i : 0..|\theta| - 1$, as follows.

$$\begin{array}{lll}
\langle \theta, i \rangle \not\models \mathbf{0} & & \\
\langle \theta, i \rangle \models p & \text{iff} & p \in \theta_i \\
\langle \theta, i \rangle \models \neg P & \text{iff} & \langle \theta, i \rangle \not\models P \\
\langle \theta, i \rangle \models P_1 \wedge P_2 & \text{iff} & \langle \theta, i \rangle \models P_1 \text{ and } \langle \theta, i \rangle \models P_2 \\
\langle \theta, i \rangle \models \ominus P & \text{iff} & i > 0 \text{ and } \langle \theta, i-1 \rangle \models P
\end{array}$$

Informally, $\ominus P$ holds in the current state if P holds in the previous state. As usual, we define $\mathbf{1} \equiv \neg\,\mathbf{0}$, $P_1 \vee P_2 \equiv \neg(\neg P_1 \wedge \neg P_2)$, $P_1 \Rightarrow P_2 \equiv \neg P_1 \vee P_2$ and $P_1 \Leftrightarrow P_2 \equiv (P_1 \Rightarrow P_2) \wedge (P_2 \Rightarrow P_1)$. Events can also be defined in terms of propositions. For instance, the following propositions denote that the truth value of P has changed in the current state (w.r.t. the previous state, if any).

$$\uparrow P \equiv (\ominus\neg P) \wedge P \quad \downarrow P \equiv (\ominus P) \wedge \neg P \quad \Uparrow P \equiv (\neg \ominus P) \wedge P \quad \Downarrow P \equiv (\neg \ominus \neg P) \wedge \neg P$$

A LIDL$^-$ formula D over $\mathbb{P}$ is defined by the grammar:

$$D ::= \lceil P \rceil^0 \mid \lceil\!\lceil P \rceil \mid D \frown D \mid D \wedge D \mid \neg\, D \mid \exists\, p.D \mid D^* \mid \eta \sim c \mid \Sigma P \sim c \mid \overleftarrow{\Diamond} D \mid P \rightsquigarrow \ell \sim c$$

where $c \in \mathbb{N}$ is a constant; $\sim \in \{<, >, =, \leq, \geq\}$; P is a proposition and $p \in \mathbb{P}$. Formulae of the form $P \rightsquigarrow \ell \sim c$ are referred to as *located constraints*, where P is the *anchor*. All variables in anchors must be free (i.e., not bound under the scope of quantifiers). A LIDL$^-$ formulae is interpreted over a timed state sequence, θ, and an interval $[i, j]$, as follows.

$\langle\theta,[i,j]\rangle \not\models false$
$\langle\theta,[i,j]\rangle \models \lceil P\rceil^0$ iff $i = j$ and $\langle\theta, i\rangle \models P$
$\langle\theta,[i,j]\rangle \models \lceil\!\lceil P\rceil$ iff $i < j$ and for all t, $i \le t < j$, $\langle\theta, t\rangle \models P$
$\langle\theta,[i,j]\rangle \models D_1 \frown D_2$ iff $i = j$ and there is a m, $i \le m \le j$, s.t. $\langle\theta,[i,m]\rangle \models D_1$ and $\langle\theta,[m,j]\rangle \models D_2$
$\langle\theta,[i,j]\rangle \models D_1 \wedge D_2$ iff $\langle\theta,[i,j]\rangle \models D_1$ and $\langle\theta,[i,j]\rangle \models D_2$
$\langle\theta,[i,j]\rangle \models \neg D$ iff $\langle\theta,[i,j]\rangle \not\models D$
$\langle\theta,[i,j]\rangle \models \exists p.\ D$ iff there is a p-variant θ' s.t. $\langle\theta',[i,j]\rangle \models D$
$\langle\theta,[i,j]\rangle \models D^*$ iff there are $i \le k_1 \le \ldots \le k_m \le j$ s.t. $\langle\theta,[k_r,k_{r+1}]\rangle \models D$ for all $r : 1..m$
$\langle\theta,[i,j]\rangle \models \eta \sim c$ iff $(j - i) \sim c$
$\langle\theta,[i,j]\rangle \models \Sigma P \sim c$ iff $(\Sigma_{k:i..j}\theta_k(P)) \sim c$
$\langle\theta,[i,j]\rangle \models \overleftarrow{\Diamond} D$ iff there is a i', $i' \le i$, s.t. $\langle\theta,[i',j]\rangle \models D$
$\langle\theta,[i,j]\rangle \models P \rightsquigarrow \ell \sim c$ iff $i = j$ and there is a i', $i' < i$, s.t. $\delta(\theta_{[i',i]}) \sim c$, $\langle\sigma(\theta), i'\rangle \models P$ and $\langle\sigma(\theta), i''\rangle \models \neg P$ for all $i' < i'' < i$

The function $\theta_k(P)$ is defined s.t. $\theta_k(P) = 1$ if $\langle\theta, k\rangle \models P$ and $\theta_k(P) = 0$ otherwise. A p-variant of θ, where $p \in \mathbb{P}$, is any θ' that is undistinguishable from θ except (possibly) for the value of p in the sequence. As usual, we define $true \equiv \neg\, false$, $D_1 \vee D_2 \equiv \neg(\neg D_1 \wedge \neg D_2)$, $D_1 \Rightarrow D_2 \equiv \neg D_1 \vee D_2$ and $D_1 \Leftrightarrow D_2 \equiv (D_1 \Rightarrow D_2) \wedge (D_2 \Rightarrow D_1)$.

Informally, the formula $\lceil P\rceil^0$ holds if P holds in the current state and the interval is a singleton. $\lceil\!\lceil P\rceil$ holds in a non-singleton interval where P holds everywhere in the interval except possibly in the last state. $D_1 \frown D_2$ holds in intervals which can be split into a pair of consecutive subintervals (sharing the last and first state) s.t. D_1 holds in the prefix and D_1 holds in the suffix. D^* holds if the interval can be divided into any number of consecutive subintervals, where each subinterval satisfies D (the equivalent of Kleene-star for regular languages). $\eta \sim c$ holds if the interval has $(\sim c)$-many states. $\Sigma P \sim c$ holds if the interval has $(\sim c)$-many P-states (i.e., states where P holds). $\overleftarrow{\Diamond} D$ holds in an interval which can be extended to the past to satisfy D. $P \rightsquigarrow \ell \sim c$ holds in a sequence where the time elapsed between the last time a (previous) state satisfying P was entered, and the time the current state was entered, satisfies the constraint $(\sim c)$.

A wealth of other operators can be derived in LIDL$^-$. In fact, formulae of the form $\eta \sim c$, $\Sigma P \sim c$ and D^* can be defined in terms of $\lceil P\rceil^0$, $\lceil\!\lceil P\rceil$ and $\exists p.D$ [22]. In this paper, we will use the following derived operators.

$$
\begin{array}{llll}
unit & \equiv \eta = 1 & & \\
\lceil P\rceil & \equiv unit \vee (unit \frown \lceil\!\lceil P\rceil) & \lceil\!\lceil P\rceil\!\rceil & \equiv \lceil\!\lceil P\rceil \frown \lceil P\rceil^0 \\
\Diamond D & \equiv true \frown D \frown true & \Box D & \equiv \neg\Diamond\neg D
\end{array}
$$

Satisfiability and Validity. Let D be a LIDL$^-$ formula. D is *satisfiable* is there exists a θ s.t. $\langle\theta,[0,|\theta|-1]\rangle \models D$. D is *valid* iff it is satisfiable in any θ. We will use $\mathcal{L}(D)$ to denote the models of D, $\mathcal{L}(D) = \{\, \theta \mid \theta \models D \,\}$ (these definitions assume that θ and D are defined over a common set of propositional variables).

Example. Consider the example of a mine pump that is able to detect high levels of methane [21]. The requirement *"After an occurrence of methane release, the level of methane remains low for at least ζ seconds."* can be expressed by the LIDL$^-$ formula,

$$D = \Box(\lceil\downarrow H\rceil^0 \frown \lceil\!\lceil\neg\, H\rceil \frown \lceil H\rceil^0 \Rightarrow true \frown (\downarrow H \rightsquigarrow \ell \ge \zeta))$$

where H is a propositional variable, which denotes "high level of methane".

3 Decidability of LIDL$^-$

This section proves the decidability of LIDL$^-$ via translation to deterministic timed automata [1].

3.1 Timed Automata

Preliminaries. Let $\mathbb{C}$ be a set of clocks (variables that range in the non-negative reals, $\mathbb{R}^{+0}$). Let Φ be the set of clock constraints over $\mathbb{C}$, s.t.

$$\phi ::= \mathit{true} \mid x \sim c \mid \phi \wedge \phi$$

where $\phi \in \Phi$, $x \in \mathbb{C}$, $\sim\, \in \{<, >, =, \leq, \geq\}$ and $c \in \mathbb{N}$. A *valuation* is a mapping from clocks to non-negative reals. Let $\mathbb{V}$ be the set of clock valuations over $\mathbb{C}$, and let $\models$ denote the satisfiability of constraints over valuations. For any $v \in \mathbb{V}$, $\delta \in \mathbb{R}^+$ and $r \subseteq \mathbb{C}$, we define the valuations $v + \delta \in \mathbb{V}$ s.t. $\forall\, x \in \mathbb{C}.\, (v + \delta)(x) = v(x) + \delta$, and $r(v) \in \mathbb{V}$ s.t. $\forall\, x \in r.\, r(v)(x) = 0$ and $\forall\, x \in \mathbb{C} \setminus r.\, r(v)(x) = v(x)$.

Syntax and semantics. A timed automaton is a tuple of the form $A = (L, l_0, \Sigma, T, C, F)$, where L is the set of locations, $l_0 \in L$ is the initial location, Σ is the alphabet, $T \subseteq L \times \Sigma \times 2^{\Phi} \times 2^{C} \times L$ is the set of transitions, $C \subseteq \mathbb{C}$ is the set of clocks and $F \subseteq L$ is the set of final locations. Given a transition $(l, a, g, r, l') \in T$, l is the source location, a is the label, g is the guard, r is the reset set and l' is the target location. A is *deterministic* if for every pair of distinct transitions (l, a, g, r, l') and (l, a, g', r', l'') with the same source location and label, the guards g and g' are disjoint.

A timed automaton A can be interpreted over a timed transition system [18], $(Q, q_0, \Sigma \cup \mathbb{R}^+, T_Q)$, where $Q \subseteq L \times \mathbb{V}$ is the set of states, $q_0 \in Q = [l_0, v_0]$ is the initial state (s.t. $\forall\, x \in C.\ v_0(x) = 0$) and $T_Q \subseteq L \times \Sigma \cup \mathbb{R}^+ \times L$ is the (semantic) transition relation. Transitions in T_Q correspond either to the execution of a transition in A, denoted $s \xrightarrow{a} s'$ ($a \in \Sigma$), or delays (the passage of time), denoted $s \xrightarrow{\delta} s'$ ($\delta \in \mathbb{R}^+$). T_Q is defined by the following rules.

$$\frac{(l,a,g,r,l') \in T,\ v \models g}{[l,v] \xrightarrow{a} [l', r(v)] \in T_Q} \qquad \frac{\delta \in \mathbb{R}^+}{[l,v] \xrightarrow{\delta} [l, v+\delta] \in T_Q}$$

A run of A is a finite path in the timed transition system, $\rho = q_0 \xrightarrow{\gamma_0} q_1 \ldots q_{n-1} \xrightarrow{\gamma_{n-1}} q_n$, where $q_i \in Q$ and $\gamma_i \in \Sigma \cup \mathbb{R}^+$. We use $\mathit{Runs}(A)$ to denote the set of all runs of A.

An *accepting* run is a run that ends in a state $[l, v]$ where $l \in F$. A location l in A is *rejecting* if no run from $[l, v]$ is accepting, for any $v \in \mathbb{V}$. For any timed state sequence θ over $\mathbb{P}$, $\theta = (s_0, t_0) \ldots (s_{n-1}, t_{n-1})$, we say that A *accepts* θ (where the alphabet of A is $2^{\mathbb{P}}$) if there exists $\rho \in \mathit{Runs}(A)$ s.t.

$$\rho = q_0 \xrightarrow{t_0} q_0' \xrightarrow{s_0} q_1 \xrightarrow{t_1 - t_0} q_1' \xrightarrow{s_1} q_2 \ldots q_{n-1} \xrightarrow{t_{n-1} - t_{n-2}} q_{n-1}' \xrightarrow{s_{n-1}} q_n$$

We will use $\mathcal{L}(A)$ to denote the set of timed state sequences accepted by A (equivalently, a timed state sequence over $\mathbb{P}$ can be seen as a timed word over $2^{\mathbb{P}}$).

3.2 From LIDL$^-$ to Timed Automata

Let D be a LIDL$^-$ formula over $\mathbb{P}$. Let $\Phi = \{\phi_1, \ldots, \phi_n\}$ be the set of all located constraints occurring in D, where $\phi_i = P_i \rightsquigarrow \ell \sim_i c_i$. Let $C = \{x_1, \ldots, x_n\}$ be a set of clocks. Let $\mathcal{B} = \{B_1, \ldots, B_n\}$ and $\mathcal{E} = \{E_1, \ldots, E_n\}$ be two sets of propositional variables, s.t. $\mathcal{B} \cap (\mathcal{E} \cup \mathbb{P}) = \mathcal{E} \cap (\mathcal{B} \cup \mathbb{P}) = \emptyset$. We refer to propositions in $\mathcal{B}$ as the *anchor witnesses*, and to propositions in $\mathcal{E}$ as *final witnesses*. In what follows, we explain how to construct a deterministic timed automaton, $A(D)$, s.t. $\mathcal{L}(A(D)) = \mathcal{L}(D)$.

From LIDL$^-$ to FSA. Let QDDC$_{\overleftarrow{\Diamond}}$ denote the extension of QDDC with the past operator $\overleftarrow{\Diamond}$ (the semantics of $\text{QDDC}_{\overleftarrow{\Diamond}}$ can be obtained by adding the semantic definition of $\overleftarrow{\Diamond}$ given in § 2 to the the semantics of QDDC [22]). For any LIDL^{-} formula D, we define the $\text{QDDC}_{\overleftarrow{\Diamond}}$ formula $\mathcal{D}$, as follows.

$$\mathcal{D} = D[\mathit{witness}(\phi_i)/\phi_i]_{i:1..n} \wedge \llbracket \bigwedge_{i:1..n} P_i \Leftrightarrow B_i \rrbracket$$

$$\mathit{witness}(\phi_i) = \lceil E_i \rceil^0 \wedge \overleftarrow{\Diamond}(\lceil P_i \rceil^0 \frown \lceil \neg P_i \rceil)$$

where $D[\beta/\alpha]$ denotes substitution of β for α in D.

In turn, $\text{QDDC}_{\overleftarrow{\Diamond}}$ is expressible in WS1S, as follows.[1] Let F be a formula in $\text{QDDC}_{\overleftarrow{\Diamond}}$. We interpret the propositional variables of ψ as second-order variables. The WS1S formula that encodes ψ, $\alpha(\psi)$, is defined as follows.

$$\alpha(\psi) = \exists w, x, y.\ \mathit{first}(w) \wedge \mathit{last}(y) \wedge x = w \wedge \beta(\psi)$$

where w, x, y respectively encode the positions of the first, current and last states in the model (the positions between the first and current states refer to *past* states), $\mathit{first}(x) \equiv \forall y.x \leq y$, $\mathit{last}(x) \equiv \forall y.y \leq x$, and $\beta(\psi)$ is the WS1S formula defined as follows (we show only a minimal subset of operators; all other operators can be derived from this subset).

$$\begin{array}{ll}
\beta(p) & = x \in p \\
\beta(\neg P) & = \neg\beta(P) \\
\beta(P \wedge Q) & = \beta(P) \wedge \beta(Q) \\
\beta(\ominus P) & = w < x \wedge \exists z.(x = z + 1 \wedge \beta(P)[z/x]) \\
\beta(\lceil P \rceil^0) & = x = y \wedge \beta(P) \\
\beta(\llbracket P \rrbracket^0) & = x < y \wedge \forall z.(x \leq z < y \Rightarrow \beta(P)[z/x]) \\
\beta(\neg\psi) & = w \leq x \leq y \wedge \neg\beta(\psi) \\
\beta(\psi_1 \wedge \psi_2) & = \beta(\psi_1) \wedge \beta(\psi_2) \\
\beta(\exists p.\psi) & = \exists p.\beta(\psi) \\
\beta(\psi_1 \frown \psi_2) & = \exists z.(x \leq z \leq y \wedge \beta(\psi_1)[z/y] \wedge \beta(\psi_2)[z/x]) \\
\beta(\overleftarrow{\Diamond}\psi) & = \exists z.(w \leq z < x \wedge \beta(\psi)[z/x])
\end{array}$$

Given the WS1S formula $\mathcal{D}$, there exists a minimal, deterministic FSA $\mathcal{A}(\mathcal{D})$ s.t. $\mathcal{L}(\mathcal{A}(\mathcal{D})) = \mathcal{L}(\mathcal{D})$. We will not describe the construction of $\mathcal{A}(\mathcal{D})$ here [7, 8]; it suffices to mention that $\mathcal{A}(\mathcal{D})$ can be automatically generated from $\alpha(\mathcal{D})$ by the MONA tool [10, 12].

Importantly, note that, whenever located constraints occur in the LIDL^{-} formula D, the conjunct $\llbracket \bigwedge_{i:1..n} P_i \Leftrightarrow B_i \rrbracket$ occurs in WS1S formula $\mathcal{D}$, and (due to minimality) the FSA $\mathcal{A}(\mathcal{D})$ is guaranteed to contain a rejecting location (which is reached for those input words where $\llbracket \bigwedge_{i:1..n} P_i \Leftrightarrow B_i \rrbracket$ does not hold). On the other hand, if located constraints do not occur in D, $\mathcal{A}(\mathcal{D})$ contains *at most* one rejecting location.[2]

From FSA to timed automata. Let $\mathcal{V} = \mathbb{P} \cup \mathcal{B} \cup \mathcal{E}$ be the set of propositional variables in $\mathcal{D}$. Let $\mathcal{A}(\mathcal{D}) = (L, l_0, 2^{\mathcal{V}}, T, F)$, where L is the set of locations, l_0 is the initial location, $2^{\mathcal{V}}$ is the alphabet, $T \subseteq L \times 2^{\mathcal{V}} \times L$ is the transition relation and $F \subseteq L$ is the set of final locations. Let $l_{\mathrm{R}} \in L$ be the unique rejecting location of $\mathcal{A}(\mathcal{D})$ (if it exists).

We construct a timed automaton $A_0 = (L, l_0, 2^{\mathcal{V}}, T_0, C, F)$, where:

[1] We simply extend the encoding of QDDC into WS1S given in [22], with a WS1S formula for $\overleftarrow{\Diamond}D$, where D is a formula in $\text{QDDC}_{\overleftarrow{\Diamond}}$.

[2] We assume the standard definition of accepting runs on FSA, and define a rejecting location as we did in § 3.1; i.e., a rejecting location is one which does not admit accepting runs. Note that, in the literature of formal languages, final locations are sometimes referred to as *accepting states*, and all other locations are referred to as *rejecting states*. Hence, in this paper, rejecting locations are rejecting states, but the converse is not necessarily true.

$$T_0 = \{\, l \xrightarrow{a,\ g,\ r} l' \mid l \xrightarrow{a} l' \in T,\ g = \bigwedge_{i:1..n} G(a,i),\ r = \{\, x_i \in C \mid B_i \in a,\ i:1..n \,\}\,\}$$

$$G(a,i) = \begin{cases} x_i \sim_i c_i & \text{if } E_i \in a \\ \neg\,(x_i \sim_i c_i) & \text{otherwise} \end{cases}$$

Let θ' be a timed state sequence over $\mathcal{V}$. We say that θ' is *$\mathcal{B}$-consistent* if $B_i \in \theta'_j \Leftrightarrow \langle \theta', j\rangle \models P_i$, for all $i : 1..n$ and $j : 1..|\theta'|$. We say that θ' is *$\mathcal{E}$-consistent* if $E_i \in \theta'_j \Leftrightarrow \exists\, j'.\ \langle \theta', [j', j]\rangle \models \phi_i$, for all $i : 1..n$ and $j : 1..|\theta'|$. We say that θ' is *consistent* if it is both $\mathcal{B}$-consistent and $\mathcal{E}$-consistent. Let θ be a timed state sequence over $\mathbb{P}$. A $\mathcal{B}$-consistent *extension* of θ is a $\mathcal{B}$-consistent θ' s.t. $\theta = [\theta']_{\mathbb{P}}$. A $\mathcal{B}$-consistent extension of a language is the set of $\mathcal{B}$-consistent extensions of the language's sequences. ($\mathcal{E}$-consistent and consistent extensions are defined similarly).

THEOREM **3.1.** *A_0 is a deterministic timed automaton s.t. $\mathcal{L}(A_0)$ is the consistent extension of $\mathcal{L}(D)$.*

Proof. Any two transitions in A_0, with the same source location and different target locations, must necessarily have different labels (because $\mathcal{A}(\mathcal{D})$ is deterministic). Hence, A_0 is deterministic. For any timed state sequence θ over $\mathbb{P}$, there exists a consistent extension θ' s.t. $\sigma(\theta') \in \mathcal{L}(\mathcal{D})$ iff $\theta \in \mathcal{L}(D)$ (by definition of $\mathcal{D}$), iff $\sigma(\theta')$ is recognized by an accepting path π in A_0 (by definition of A_0 from $\mathcal{A}(\mathcal{D})$, and $\mathcal{L}(\mathcal{A}(\mathcal{D})) = \mathcal{L}(\mathcal{D})$). In addition, for any timed state sequence ω that is accepted by π, the following holds (by definition of guards and resets in A_0): (a) x_i is reset in transition π_j (the transition in π that is visited in the j^{th} step during recognition of ω) iff $B_i \in \omega_j \Leftrightarrow \langle \omega, j\rangle \models P_i$, and (b) $x_i \sim_i c_i$ guards the transition π_j iff $E_i \in \theta_j \Leftrightarrow \exists\, j'.\ \langle \theta, [j', j]\rangle \models \phi_i$, for all $i : 1..n$, $j : 1..|\omega|$. Hence, $[\theta']_{\mathbb{P}} \in \mathcal{L}(D)$ iff $\theta \in \mathcal{L}(A_0)$, i.e., $\mathcal{L}(A_0)$ is the consistent extension of $\mathcal{L}(D)$. $\square$

The effective construction of A_0 suffices to prove the decidability of LIDL^-_Δ. However, since the alphabet include witnesses ($\mathcal{B}$ and $\mathcal{E}$), A_0 is not itself a useful observer for LIDL^-_Δ properties. Next, we construct $A(D)$ from A_0. This is achieved by removing from A_0 all transitions that target the rejecting location (if this location exists), s.t. another transition exists in the same source location with a different target, and whose label differs only in the actual value of anchor witnesses (if all such transitions target the rejecting location, w.l.o.g, we remove all but the one where all anchor witnesses are false). As a final step, the alphabet is projected over $\mathbb{P}$. We define $A(D) = (L, l_0, 2^{\mathbb{P}}, T_1, C, F)$, where:

$$\begin{aligned} T_1 = \{\, l \xrightarrow{b,g,r} l' \mid \quad & l \xrightarrow{a,g,r} l' \in T_0,\ b = [a]_{\mathbb{P}}, \\ & l' \neq l_{\mathrm{R}} \vee (a \cap \mathcal{B} = \emptyset \wedge (\forall\, t \in T_0.\ (src(t) = l \wedge [lab(t)]_{\mathbb{P}\cup\mathcal{E}} = [a]_{\mathbb{P}\cup\mathcal{E}}) \Rightarrow tgt(t) = l'))\,\} \end{aligned}$$

THEOREM **3.2.** *$A(D)$ is a deterministic timed automaton, s.t. $\mathcal{L}(A(D)) = \mathcal{L}(D)$.*

Proof. Assume (by contradiction) that located constraints occur in D, and that $A(D)$ is non-deterministic. Then, there exists a pair of transitions $t_1, t_2 \in T_1$, $t_1 = (l, b, g_1, r_1, l_1)$ and $t_2 = (l, b, g_2, r_2, l_2)$, s.t. $g_1 \cap g_2 \neq \emptyset$ and $l_1 \neq l_2$. By construction of A_1, this implies $g_1 = g_2$, $l_1 \neq l_{\mathrm{R}}$ and $l_2 \neq l_{\mathrm{R}}$. Necessarily, $t_1, t_2 \in T_1$ were derived from a pair of transitions $t'_1, t'_2 \in T_0$, $t'_1 = (l, a_1, g_1, r_1, l_1)$ and $t'_2 = (l, a_2, g_2, r_2, l_2)$, s.t. $[a_1]_{\mathbb{P}\cup\mathcal{E}} = [a_2]_{\mathbb{P}\cup\mathcal{E}}$ and $[a_1]_{\mathcal{B}} \neq [a_2]_{\mathcal{B}}$ (because A_0 is deterministic). Let ω_1 and ω_2 denote any two input prefixes, s.t. ω_1 is recognized by a path in A_0 from l_0 to l_1, and ω_2 is recognized by a path in A_0 from l_0 to l_2. Thus, $\omega_1 \models \llbracket \bigwedge_{i:1..n} P_i \Leftrightarrow B_i \rrbracket$ iff $\omega_2 \not\models \llbracket \bigwedge_{i:1..n} P_i \Leftrightarrow B_i \rrbracket$, which implies either $l_1 = l_{\mathrm{R}}$ or $l_2 = l_{\mathrm{R}}$. This is a contradiction.

Now, we prove that $\mathcal{L}(A) = \mathcal{L}(D)$. By definition of A, $[\mathcal{L}(A_0)]_{\mathbb{P}} = \mathcal{L}(A)$ (because the accepting paths of A correspond exactly to the accepting paths in A_0, projected over $\mathbb{P}$). By Theorem 3.1, $\mathcal{L}(A_0)$ is the consistent extension of $\mathcal{L}(A)$, hence $[\mathcal{L}(A_0)]_{\mathbb{P}} = \mathcal{L}(D)$, and thus $\mathcal{L}(A) = \mathcal{L}(D)$. $\square$

4 The logic LIDL^-_Δ

This section introduces the syntax and semantics of LIDL^-_Δ. LIDL^-_Δ is a variant of LIDL^- where the use of $\overleftarrow{\Diamond}$ is disallowed and timing constraints take the form of future located constraints.

Syntax and semantics. A LIDL^-_Δ formula D over $\mathbb{P}$ is defined by the grammar:

$$D ::= \lceil P \rceil^0 \mid \lceil\!\lceil P \rceil \mid D \frown D \mid D \wedge D \mid \neg D \mid \exists p.D \mid D^* \mid \eta \sim c \mid \Sigma P \sim c \mid P\Delta_{\sim c}$$

where $c \in \mathbb{N}$ is a constant; $\sim \in \{<, >, =, \leq, \geq\}$; P is a proposition over $\mathbb{P}$ (defined as in § 2) and $p \in \mathbb{P}$. Formulae of the form $P\Delta_{\sim c}$ are referred to as future located constraints, where P is the anchor. All variables in anchors must be free (i.e., not bound under the scope of quantifiers). The semantics of LIDL^- and LIDL^-_Δ coincide for propositions and common operators; future located constraints are interpreted as follows.

$$\langle \theta, [i, j] \rangle \models P\Delta_{\sim c} \quad \text{iff} \quad \delta(\theta_{[i,j]}) \sim c,\ \langle \theta, i \rangle \models P, \text{ and } \langle \theta, i' \rangle \models \neg P \text{ for all } i',\ i < i' < j$$

Informally, $P\Delta_{\sim c}$ holds if P holds in the current state, P does not hold anywhere else in the interval except possibly in the last state, and the elapsed time in the interval satisfies $(\sim c)$. The satisfiability and validity of LIDL^-_Δ formulae are defined as for LIDL^-.

Example. The LIDL^- property D_{MP} of § 2, which expresses a requirement on a mine pump, can be expressed by the LIDL^-_Δ property D'_{MP}, as follows.

$$D'_{MP} = \Box(\lceil \downarrow H \rceil^0 \frown \lceil\!\lceil \neg H \rceil \frown \lceil H \rceil^0 \Rightarrow \downarrow H \Delta_{\geq \zeta} \wedge (\mathit{true} \frown \lceil \uparrow H \rceil^0))$$

4.1 Decidability

The decidability of LIDL^-_Δ follows from the decidability of LIDL^-, because LIDL^-_Δ can be encoded in LIDL^-.

THEOREM **4.1.** *For any LIDL^-_Δ formula D, a deterministic timed automaton can be constructed which accepts the models of D (the time and space complexity of this construction is non-elementary). In addition, an event-clock automaton can be constructed which expresses D.*

Proof. LIDL^-_Δ is expressible in LIDL^-, by the equivalence: $P\Delta_{\sim c} \equiv \lceil P \rceil^0 \frown (\eta \geq 1) \frown (P \rightsquigarrow \ell \sim c)$. □

Given a LIDL^-_Δ formula D, by Theorem 4.1, we can express D as a LIDL^- formula D' and use the construction of § 3 to obtain the equivalent timed automaton. The translation to LIDL^- is not necessary, however, because the construction of § 3 can be applied over D (with trivial modifications). It suffices to define $\mathit{witness}(P_i \Delta_{\sim_i c_i}) = \lceil P_i \rceil^0 \frown \lceil \neg P_i \rceil \frown \lceil E_i \rceil^0$. Moreover, the untimed formula $\mathcal{D} = D[\mathit{witness}(\phi_i)/\phi_i]_{i:1..n} \wedge \lceil\!\lceil \bigwedge_{i:1..n} P_i \Leftrightarrow B_i \rceil\!\rceil$ is a now QDDC formula, and thus, the DCVALID tool can be used to generate the FSA $\mathcal{A}(\mathcal{D})$ from it [22].

A note on DCVALID. As we mentioned, DCVALID automatically generates an FSA that accepts the models of a QDDC formula [22]. DCVALID translates an input QDDC formula into an equivalent

WS1S formula, and uses the MONA tool [10, 12].[3] to generate the equivalent FSA. This FSA is slightly different to the one considered in § 3. The main differences are discussed below.

1. Transitions are labeled with strings in $\mathbb{B}_{\mathbf{x}} = \{\mathbf{0}, \mathbf{1}, \mathbf{x}\}^{|\mathcal{V}|}$, where $\mathcal{V}$ is the set of propositional variables in the QDDC formula, $\mathcal{D}$. Given an enumeration of the variables in $\mathcal{V}$, $\varepsilon : [1..|\mathcal{V}|] \to \mathcal{V}$, a string $\omega \in \mathbb{B}_{\mathbf{x}}$ denotes the following set of valuations,

$$\omega = \{\, s \in 2^{\mathcal{V}} \mid \forall\, i : 1..|\mathcal{V}|.\ (w_i = \mathbf{0} \Rightarrow \varepsilon(i) \notin s) \wedge\ (w_i = \mathbf{1} \Rightarrow \varepsilon(i) \in s)\,\}$$

   This allows for a concise representation of multiple transitions between the same pair of locations.[4] For instance, given $\mathcal{V} = \{P, Q\}$ and $\varepsilon = \{(1, P), (2, Q)\}$, $\omega_1 = \mathbf{01}$ denotes the singleton $\omega_1 = \{\{Q\}\}$, while $\omega_2 = \mathbf{1x}$ denotes the set $\omega_2 = \{\{P\}, \{P, Q\}\}$.

2. Transitions at the initial location read the Boolean variables in the WS1S formula [11]. However, the encoding of QDDC in WS1S results in formulae *without* Boolean variables. Hence, for any QDDC formula $\mathcal{D}$, the equivalent FSA generated by DCVALID, $\mathcal{M}(\mathcal{D})$, contains a transition of the form $t_0 :\ l_0 \xrightarrow{\mathbf{xx...x}} l_1$, where l_0 is the initial location, l_0 has no ingoing transitions, and t_0 is the only one outgoing transition from l_0. Correspondingly, the automaton $\mathcal{A}(\mathcal{D})$ of § 3 is obtained from $\mathcal{M}(\mathcal{D})$ by removing l_0 and t_0, and making l_1 the new initial location.

Example. Consider again the LIDL$^-_\Delta$ formula D'_{MP}. This formula contains only one future located constraint, namely, $\phi_1 = (\downarrow H)\Delta_{\geq\zeta}$. The corresponding QDDC formula is given by $\mathcal{D}$, as follows.

$$\mathcal{D} = \quad \Box(\lceil\downarrow H\rceil^0 \frown \lceil\!\lceil\neg H\rceil \frown \lceil H\rceil^0 \Rightarrow (\lceil\downarrow H\rceil^0 \frown \lceil\neg \downarrow H\rceil \frown \lceil E\rceil^0) \wedge (\mathit{true} \frown \lceil\uparrow H\rceil^0))\ \wedge\ \lceil\!\lceil\downarrow H \Leftrightarrow B\rceil\!\rceil$$

where $\mathbb{P} = \{H\}$, $\mathcal{B} = \{B\}$ and $\mathcal{E} = \{E\}$.

Figure 1 shows the FSA $\mathcal{A}(\mathcal{D})$. The figure depicts the automaton as generated by DCVALID (after removing the initial transitions). Edges are labeled with strings in $\{\mathbf{0}, \mathbf{1}, \mathtt{x}\}^3$, assuming the enumeration $\varepsilon = \{(1, H), (2, B), (3, E)\}$. S2, S4, S5 and S6 are the final locations, and S3 is the rejecting location. Figure 2 shows the timed automaton $A(D)$, where y is the clock associated with the located constraint $(\downarrow H)\Delta_{\geq\zeta}(\uparrow H)$, and we use the labels $\mathbf{0} = \emptyset$ and $\mathbf{1} = \{H\}$ (we omit guards that are trivially true and empty reset sets). To illustrate the construction of $A(D)$ from $\mathcal{A}(\mathcal{D})$, let us consider the following edges in $\mathcal{A}(\mathcal{D})$ (fig. 1):

$$t_1 = \mathtt{S4} \xrightarrow{\mathbf{01x}} \mathtt{S5} = \{\, \mathtt{S4} \xrightarrow{\{B,E\}} \mathtt{S5},\ \mathtt{S4} \xrightarrow{\{B\}} \mathtt{S5}\,\}$$
$$t_2 = \mathtt{S4} \xrightarrow{\mathbf{00x}} \mathtt{S3} = \{\, \mathtt{S4} \xrightarrow{\{E\}} \mathtt{S3},\ \mathtt{S4} \xrightarrow{\emptyset} \mathtt{S3}\,\}$$

Equivalently, if σ denotes the input state sequence, and $\sigma_{[0,i-1]}$ is the prefix of σ which has been read up to S4, then the automaton evolves as follows:

$$t_1 \text{ is executed if } \langle\sigma, i\rangle \models \neg H \wedge B$$
$$t_2 \text{ is executed if } \langle\sigma, i\rangle \models \neg H \wedge \neg B$$

Hence, if the automaton is currently in S4 and $\langle\sigma, i\rangle \models \neg H$, then no constraint should be enforced on y (because E is irrelevant), but y should be reset (because $B \notin \sigma_i$ rejects the input, i.e., $\langle\sigma, i\rangle \models\downarrow H$). Accordingly, t_1 and t_2 in $\mathcal{A}(\mathcal{D})$ are mapped to $t = \mathtt{S4} \xrightarrow{\emptyset, \mathit{true}, \{y\}} \mathtt{S5}$ in $A(D)$ (fig. 2).

To illustrate how clock constraints are added to the FSA, let us now consider the edges:

[3] http://www.brics.dk/mona/

[4] In MONA's jargon, $\mathbf{x}$ is referred to as a *"don't care"* value.

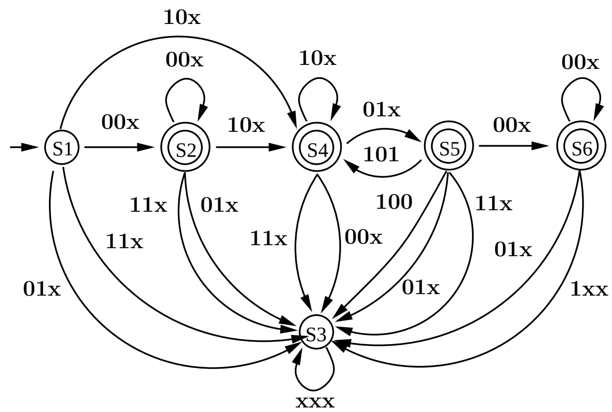

Figure 1: The FSA $\mathcal{A}(\mathcal{D})$, as generated by DCVALID

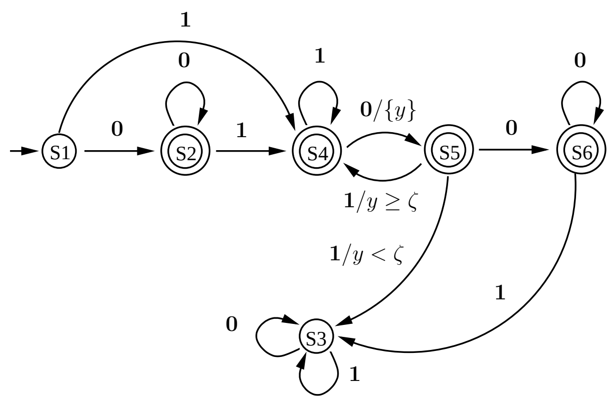

Figure 2: The timed automaton $A(D)$

$$
\begin{aligned}
t_1 &= \mathtt{S5} \xrightarrow{\mathbf{101}} \mathtt{S4} = \{\, \mathtt{S5} \xrightarrow{\{H,E\}} \mathtt{S4} \,\} \\
t_2 &= \mathtt{S5} \xrightarrow{\mathbf{100}} \mathtt{S3} = \{\, \mathtt{S5} \xrightarrow{\{H\}} \mathtt{S3} \,\} \\
t_3 &= \mathtt{S5} \xrightarrow{\mathbf{11x}} \mathtt{S3} = \{\, \mathtt{S5} \xrightarrow{\{H,B,E\}} \mathtt{S3},\ \mathtt{S5} \xrightarrow{\{H,B\}} \mathtt{S3} \,\}
\end{aligned}
$$

Equivalently,

$$
\begin{aligned}
&t_1 \text{ is executed if } \langle \sigma, i \rangle \models H \wedge \neg B \wedge E \\
&t_2 \text{ is executed if } \langle \sigma, i \rangle \models H \wedge \neg B \wedge \neg E \\
&t_3 \text{ is executed if } \langle \sigma, i \rangle \models H \wedge B
\end{aligned}
$$

Hence, if the automaton is currently in S5 and $\langle \sigma, i \rangle \models H$, then y must not be reset (because $B \in \sigma_i$ rejects the input, i.e., $\langle \sigma, i \rangle \not\models \downarrow H$), but timing constraints should be enforced on y (because, provided $B \notin \sigma_i$, the rejecting location is immediately reached iff $E \notin \sigma_i$). Accordingly, t_1, t_2 and t_3 in $\mathcal{A}(\mathcal{D})$ are mapped to $\mathtt{S5} \xrightarrow{\mathbf{1}, y \geq \zeta, \emptyset} \mathtt{S4}$ and $\mathtt{S5} \xrightarrow{\mathbf{1}, y < \zeta, \emptyset} \mathtt{S3}$ in $A(D)$ (fig. 2).

5 On the expressive power of LIDL$^-$ and LIDL$^-_\Delta$

Many complex, quantitative timing requirements can be expressed in LIDL$^-$ and LIDL$^-_\Delta$. For instance, the following LIDL$^-_\Delta$ properties are variants of the patterns described in [25, 21],

$$\begin{array}{ll} stable(P,\delta) & = \Box(\Uparrow P \Delta_{\leq\delta} \Rightarrow \llbracket P \rrbracket) \\ \llbracket P \rrbracket \hookleftarrow^{\delta} \lceil Q \rceil & = \neg\Diamond(\Uparrow P \Delta_{\geq\delta} \wedge \llbracket P \rrbracket \frown \lceil \neg Q \rceil^0) \end{array}$$

The first property above states that whenever P becomes true, it must hold for at least δ t.u. The second property states that whenever P holds for at least δ t.u. after it becomes true, then Q must become true within δ t.u. from P becoming true, and Q must hold for as long as P holds.

On the other hand, LIDL$^-$ and LIDL$^-_\Delta$ are limited by the semantics of located constraints, which may refer only to the time elapsed between *state changes*. In addition, located constraints are relative to the *last* occurrence of a particular state change (the anchor becoming true). In other words, in order to constrain the time elapsed between two state changes e_1 and e_2, e_1 is required not to occur again until e_2 occurs. Consider, for instance, $\theta = (\{P\},0)(\{Q\},0.5)(\{P\},2)(\{Q\},4)$. The requirement *"Q holds 1 t.u. after P holds"* is satisfiable in θ, because Q holds continuously in the interval [0.5,2); but this cannot be expressed with located constraints, because there are no state changes 1 t.u. after P starts to hold. Similarly, the requirement *"Q holds 4 t.u. after P holds"* is satisfiable in θ, because 4 t.u. pass between the first time P starts to hold, and the second time Q starts to hold; but located constraints cannot relate the state changes at $(\{P\},0)$ and $(\{Q\},4)$ (P changes again at $(\{P\},2)$). Nonetheless, despite these limitations, we believe that LIDL$^-$ and LIDL$^-_\Delta$ are expressive enough to describe a practically useful class of properties.

We conclude this section by comparing the expressive power of LIDL$^-$ and LIDL$^-_\Delta$, relative to some well-known linear-time logics.

LIDL$^-$ and LIDL$^-_\Delta$. We know that, LIDL$^-$ is at least as expressive as LIDL$^-_\Delta$ (see Theorem 4.1). However, whether LIDL$^-$ is strictly more expressive than LIDL$^-_\Delta$, remains an open problem. We observe that, MTL$_S$, which denotes MTL [13] extended with the past operator *since* ($\mathcal{S}$), is more expressive than MTL [24].[5] In [24], it is proved that L_{last_a} can be expressed in MTL$_S$ but not in MTL, where L_{last_a} is the language of all finite timed words over the alphabet $\Sigma = \{a, b\}$ that have an action occurrence at time 1, and a is the last action to occur in the interval $(0, 1)$. This may suggest that LIDL$^-_\Delta$ is less expressive than LIDL$^-$. However, L_{last_a} is already expressible in LIDL$^-_\Delta$, and thus cannot be used as a witness. L_{last_a} can be expressed by the LIDL$^-_\Delta$ formula ϕ_{last_a}, which is defined as follows.[6]

$$\begin{array}{ll} \phi_{last\_a} & = ((\phi_{a\in(0,1)} \frown \llbracket \neg b \rceil) \wedge \phi_{at\_1}) \frown true \\ \phi_{a\in(0,1)} & = (\phi_0 \Delta_{>0} \wedge \phi_0 \Delta_{<1}) \frown \lceil a \rceil^0 \\ \phi_{at\_1} & = \phi_0 \Delta_{=1} \wedge (true \frown \lceil a \vee b \rceil^0) \\ \phi_0 & = \lceil \ominus false \rceil^0 \end{array}$$

where ϕ_0 denotes the first state of an interval, $\phi_{a\in(0,1)}$ denotes an interval that ends with the occurrence of a in $(0, 1)$, and ϕ_{at_1} denotes the occurrence of an event exactly at time 1.

MTL. LIDL$^-$ and LIDL$^-_\Delta$ are expressively incomparable to MTL. On the one hand, MTL cannot express the language L_{even_b}, which consists of all timed words in $\Sigma = \{a, b\}$ with an even number of b's [24] (in fact, [24] proves that L_{even_b} is inexpressible even in MTL$_{S_I}$, a more expressive logic than MTL$_S$ where the $\mathcal{S}$ operator can be restricted to a time interval I). L_{even_b} can be expressed by the LIDL$^-_\Delta$ formula ϕ_{even_b}, which is defined as follows.

$$\phi_{even\_b} = \llbracket \neg b \rceil^* \frown (\lceil b \rceil^0 \frown \lceil \neg b \rceil \frown \lceil b \rceil^0 \frown \lceil \neg b \rceil)^*$$

[5]Throughout this section, we consider MTL interpreted over finite timed words with pointwise semantics [24]. This allows us to compare MTL with LIDL$^-_\Delta$ and LIDL$^-$, which are interpreted over finite timed state sequences.

[6]We assume that actions $a, b, \ldots$, in a timed word are represented by propositions (events) $\uparrow A, \uparrow B, \ldots$, in a timed state sequence, where $A, B, \ldots$, are propositional variables.

On the other hand, the MTL property, $\phi_{ev\_until} = \Diamond(a\mathcal{U}_{[1,\infty)}b)$, cannot be expressed in LIDL$^-$, because constraints may only be expressed on the time elapsed since the *last* occurrence of an anchor event (and no such event can be defined for ϕ_{ev_until}).

Simple DC* and Timed Regular Expressions. Both Simple DC* [9] (a fragment of Duration Calculus with iteration) and timed regular expressions [5] are expressively complete for (non-deterministic) timed automata and hence more expressive than LIDL$^-$ and LIDL$^-_\Delta$. However, both Simple DC* and timed regular expressions are negation-free, which makes LIDL$^-$ and LIDL$^-_\Delta$ non-elementary more succinct [28, 27].

Test formulae. Test formulae, the subset of DC studied in [17], may express unanchored durations, which are inexpressible in either LIDL$^-_\Delta$ or LIDL$^-$. In contrast, test formulae are star-free and restrict the use of negation. For instance, the LIDL$^-_\Delta$ formula $\neg((\lceil P\rceil \vee \lceil Q\rceil)^\frown\lceil R\rceil)$, for some propositions P, Q and R, cannot be expressed by a test formula.

References

[1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] R. Alur, L. Fix, and T. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999.

[3] R. Alur, L. Fix, and T. A. Henzinger. A determinizable class of timed automata. In *Computer Aided Verification, CAV'94*, LNCS 818, pages 1–13. Springer, 1994.

[4] R. Alur and T. Henzinger. Logics and models of real time: A survey. In *Real-Time: Theory in Practice, REX Workshop*, pages 74–106. Springer, 1991.

[5] E. Asarin, P. Caspi, and O. Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.

[6] V. Braberman and D.V. Hung. On checking timed automata for linear duration invariants. In *Proc. of the 19th IEEE Real-time Systems Symposium*, pages 264–273, Piscataway, NJ, 1998. IEEE Press.

[7] J.R. Büchi. On a decision method in restricted second-order arithmetic. *Zeitschrift für Mathemathische Logik and Grundlagen der Mathematik*, 6:66–92, 1960.

[8] C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.

[9] D. Guelev and D. Van Hung. On the completeness and decidability of Duration Calculus with iteration. *Theoretical Computer Science*, 337(1-3):278–304, 2005.

[10] J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95*, volume 1019 of *LNCS*, pages 89–110. Springer-Verlag, 1995.

[11] N. Klarlund and A. Möller. *MONA Version 1.4 User Manual.* BRICS, University of Aarhus, Denmark, January 2001. Notes Series NS-01-1. Available from `http://www.brics.dk/mona/`. Revision of BRICS NS-98-3.

[12] N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA implementation secrets. *International Journal of Foundations of Computer Science*, 13(4):571–586, 2002. World Scientific Publishing Company. Earlier version in Proc. 5th International Conference on Implementation and Application of Automata, CIAA '00, Springer-Verlag LNCS vol. 2088.

[13] R. Koymans. Specifying real-time properties with Metric Temporal Logic. *Real-Time Systems*, 2(4):255–299, 1990.

[14] R. Koymans, J. Vytopil, and W.P. de Roever. Real-time programming and asynchronous message passing. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 187–197. ACM, 1983.

[15] Y. Li and D. V. Hung. Checking temporal duration properties of timed automata. *Journal of Computer Science and Technology*, 17(6):689–698, 2002.

[16] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218. Springer-Verlag, 1985.

[17] R. Meyer, J. Faber, J. Hoenicke, and A. Rybalchenko. Model checking Duration Calculus: a practical approach. *Formal Aspects of Computing*, 20(4-5):481–505, 2008.

[18] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *Proc. of CONCUR 1990*, pages 401–415. Springer-Verlag New York, Inc., 1990.

[19] B. Moszkowski. A temporal logic for multilevel reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.

[20] P. Pandya. Model checking CTL*[DC]. In *Proc. TACAS 2001*, volume 2031 of *LNCS*, pages 559 – 573. Springer-Verlag, 2001.

[21] P. Pandya. Interval Duration Logic: Expressiveness and decidability. *ENTCS*, 65(6), 2002.

[22] P. K. Pandya. Specifying and deciding Quantified Discrete Duration Calculus formulae using DCVALID. Technical Report TCS00-PKP-1, Tata Institute of Fundamental Research, 2000.

[23] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium Foundations of Computer Science (FOCS 1977)*, pages 46–57, 1977.

[24] P. Prabhakar and D. D'Souza. On the expressiveness of MTL with past operators. In *Formal Modeling and Analysis of Timed Systems, FORMATS'06*, LNCS 5215, pages 322–336. Springer, 2006.

[25] A. Ravn. *Design of Embedded Real-Time Computing Systems*. PhD thesis, Department of Computer Science, Technical University of Denmark, 1994.

[26] R. Rosner and A. Pnueli. A choppy logic. In *Proceedings of the First Annual IEEE Symposium on Logics in Computer Science*, pages 306–314. IEEE, 1986.

[27] L. Stockmeyer. *The complexity of decision problems in automata and logic*. PhD thesis, MIT, Cambridge, MA, 1974.

[28] L. Stockmeyer and A. Meyer. Word problems requiring exponential time (preliminary report). In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9. ACM, 1973.

[29] P. H. Thai and D. V. Hung. Verifying linear duration constraints of timed automata. In *Proc. of ICTAC'04*, volume 3407 of *LNCS*, pages 295–309. Springer, 2004.

[30] B.A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *AMS Transl.*, 59:23–55, 1966. (first appeared in Siberian Math. Journal 3:103-131, 1962, in Russian).

[31] C. Zhou and M. Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems*. EATCS Series of Monographs in Theoretical Computer Science. Springer, 2004.

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