Specifying and Deciding Quantified Discrete-time Duration Calculus Formulae using DCVALID

Paritosh K. Pandya
School of Technology and Computer Science
Tata Institute of Fundamental Research
Homi Bhabha Road, Colaba, Mumbai
email: pandya@tcs.tifr.res.in

Abstract

Quantified Discrete-time Duration Calculus (QDDC) is a logic for specifying properties of finite sequences of states. It provides novel interval based modalities to specify how a system evolves with time. In this note, we give the syntax and semantics of logic QDDC. We illustrate the ability of QDDC to model complex real-time requirements by an example of a mine pump. As our main result, we show that the class of models of a QDDC formula can be characterised by words accepted a finite state automaton which can be effectively constructed. Satisfiability (validity) of a QDDC formula can be established by searching for an accepting (rejecting) path within the (deterministic) automaton. Moreover, the automaton can be used as synchronous observer (or monitor) for model checking QDDC formulae. We briefly discuss our implementation of this decision procedure for QDDC into a tool called DCVALID. We report some experimental results obtained by using DCVALID.

1 Introduction

Quantified Discrete-time Duration Calculus (QDDC) [12] is a highly expressive logic for specifying properties of finite sequences of states (behaviours). It is closely related to the Interval Temporal Logic of Moszkowski [10] and Duration Calculus of Zhou *et al* [24]. It provides novel interval based modalities for describing behaviours. For example, the following formula holds for a behaviour σ provided for all fragments σ' of σ which have (a) P true in the beginning, (b) Q true at the end, and (c) no occurrences of Q in between, the number of occurrences of states in σ' where R is true is at most 3.

$$\Box(\lceil P \rceil^0 \frown [\![\neg Q]\!] \frown \lceil Q \rceil^0 \;\Rightarrow\; (\Sigma R \leq 3))$$

Here, $\Box$ modality ranges over all fragments of a behaviour. Operator $\frown$ is like concatenation (fusion) of behaviour fragments and $[\![\neg Q]\!]$ states invariance of

$\neg Q$ over the behaviour fragment. Finally, ΣR counts number of occurrences of R within a behaviour fragment. A precise definition of the syntax and semantics of QDDC is given in Section 2. Formula $\eta = 3$ states that the behaviour fragment has length 3 (i.e. it spans a sequence of 4 states).

QDDC is a discrete-time variant of Duration Calculus which was conceived as a highly expressive notation for capturing requirements of real-time systems. We believe that QDDC provides a convenient notation for expressing complex requirements of reactive and timed systems. As an example of this we model in QDDC a mine pump system, which has been used as a benchmark for comparing various formalisms for real-time systems. The expressive abilities of QDDC are also demonstrated by its use in capturing compositional semantics of complex languages such as Esterel [16]. (The reader is urged to imagine how this semantics would look in his favourite temporal logic to realise full import of this.)

In spite of its richness of expression, QDDC is decidable. An automata-theoretic decision procedure allows checking of satisfiability (validity) of QDDC formulae. This algorithm has been implemented into a tool called DCVALID [12]. The decision procedure relies on embedding (translation) of logic QDDC into Monadic logic over finite words, MLSTR. Logic MLSTR has automata theoretic decision procedure which can be used to decide QDDC. A well-known efficient impmentation of Monadic logic over finite words, MONA [7], provides the requisite algorithms to decide QDDC formulae.

The rest of this paper is organised as follows. We give the syntax and semantics of logic QDDC in Section 2. We formalise a mine pump system and its correctness under suitable environmental assumption in Section 3. The decidability of QDDC using automata theoretic approach is established in Section 4. The expressiveness of the logic is also characterised. Finally, we briefly describe the tool DCVALID which implements the above decision procedure in Section 5 and give some experimental results from its use. We end the paper with a discussion.

2 Quantified Discrete-Time Duration Calculus (QDDC)

Let $Pvar$ be a finite set of propositional variables representing some observable aspects of system state. Let

$$VAL(Pvar) \quad \stackrel{\text{def}}{=} \quad Pvar \to \{0,1\}$$

be the set of valuations assigning truth-value to each variable.

We shall identify behaviours with finite, nonempty sequences of valuations, i.e. $VAL(Pvar)^+$.

Example 2.1 *The following picture gives a behaviour over variables $\{p,q\}$. Each column vector gives a valuation, and the word is a sequence of such column vectors.*

```
p   1  0  1  1  0
q   0  0  0  0  1
```

The above word satisfies the property that p holds initially and q holds at the end but nowhere before that. QDDC is a logic for formalising such properties. Each formula specifies a set of such words.

Given a non-empty finite sequence of valuations $\sigma \in VAL^+$, we denote the satisfaction of a QDDC formula D over σ by

$$\sigma \models D$$

We now give the syntax and semantics of QDDC and define the above satisfaction relation.

Syntax of QDDC Formulae Let $Pvar$ be the set of propositional variables. Let p, q range over propositional variables, P, Q over propositions and D, D_1, D_2 over QDDC formulae.
The set of propositions $Prop$ has the syntax

$$0 \mid 1 \mid p \mid P \wedge Q \mid \neg P \mid -P \mid +P$$

Operators such as $\vee, \Rightarrow, \Leftrightarrow$ can be defined as usual.
The syntax of QDDC is as follows.

$$\lceil P \rceil^0 \mid \lceil\lceil P \rceil \mid D_1 \frown D_2 \mid D_1 \wedge D_2 \mid \neg D \mid \exists p.D$$
$$\eta \ op \ c \mid \Sigma P \ op \ c \qquad \text{where} \quad op \in \{>, =\}$$

Let $\sigma \in VAL(Pvar)^+$ be a behaviour. Let $\#\sigma$ denote the length of σ and $\sigma[i]$ the i'th element. For example, if $\sigma = \langle v_0, v_1, v_2 \rangle$ then $\#\sigma = 3$ and $\sigma[1] = v_1$. Let $dom(\sigma) = \{0, 1, \ldots, \#\sigma - 1\}$ denote the set of positions within σ.

Let $\sigma, i \models P$ denote that proposition P evaluates to true at position i in σ. We omit this obvious definition and give only a few clauses below.

$$\sigma, i \models p \quad \textbf{iff} \quad \sigma[i](p) = 1$$
$$\sigma, i \models -P \quad \textbf{iff} \quad i > 0 \text{ and } \sigma, i-1 \models P$$
$$\sigma, i \models +P \quad \textbf{iff} \quad i < (\#\sigma - 1) \text{ and } \sigma, i+1 \models P$$

The set of intervals in σ is given by $Intv(\sigma) = \{[b, e] \in dom(\sigma)^2 \mid b \leq e\}$ where each interval $[b, e]$ identifies a subsequence of σ between positions b and e.

We inductively define the satisfaction of a QDDC formula D for behaviour σ and interval $[b, e] \in Intv(\sigma)$ as follows. This is denoted by $\sigma, [b, e] \models D$.

$$\sigma, [b, e] \models \lceil P \rceil^0 \quad \textbf{iff} \quad b = e \text{ and } \sigma, b \models P$$
$$\sigma, [b, e] \models \lceil\lceil P \rceil \quad \textbf{iff} \quad b < e \text{ and } \sigma, i \models P \text{ for all } i : b \leq i < e$$
$$\sigma, [b, e] \models \neg D \quad \textbf{iff} \quad \sigma, [b, e] \not\models D$$
$$\sigma, [b, e] \models D_1 \wedge D_2 \quad \textbf{iff} \quad \sigma, [b, e] \models D_1 \text{ and } \sigma, [b, e] \models D_2$$
$$\sigma, [b, e] \models D_1 \frown D_2 \quad \textbf{iff} \quad \text{for some } m : b \leq m \leq e :$$
$$\sigma, [b, m] \models D_1 \text{ and } \sigma, [m, e] \models D_2$$

Entities η and ΣP are called *measurements*. Term η denotes the length of the interval whereas ΣP denotes the count of number of times P is true within the interval $[b, e]$ (we treat the interval as being left-closed right-open). Formally,

$$eval(\eta, \sigma, [b, e]) \stackrel{\text{def}}{=} e - b$$
$$eval(\Sigma P, \sigma, [b, e]) \stackrel{\text{def}}{=} \sum_{i=b}^{e-1} \left\{ \begin{array}{ll} 1 & if \quad \sigma, i \models P \\ 0 & otherwise \end{array} \right\}$$

Let t range over measurements. Then,

$$\sigma, [b, e] \models t \; op \; c \quad \textbf{iff} \quad eval(t, \sigma, [b, e]) \; op \; c$$

Call a behaviour σ' to be p-variant of σ provided $\#\sigma = \#\sigma'$ and for all $i \in dom(\sigma)$ and for all $q \neq p$, we have $\sigma(i)(q) = \sigma'(i)(q)$. Then,

$$\sigma, [b, e] \models \exists p.D \quad \textbf{iff} \quad \sigma', [b, e] \models D \;\; \text{for some } p\text{-variant } \sigma' \text{ of } \sigma$$

Finally,

$$\sigma \models D \quad \textbf{iff} \quad \sigma, [0, \#\sigma - 1] \models D$$

We can also define some derived constructs. Boolean combinators $\vee, \Rightarrow, \Leftrightarrow$ can be defined using $\wedge, \neg$ as usual.

- $\lceil\lceil P \rceil\rceil \stackrel{\text{def}}{=} (\lceil\lceil P \rceil \frown \lceil P \rceil^0)$ states that proposition P holds invariantly over the closed interval $[b, e]$ including the endpoint.
  $\lceil\lceil P \rceil\rceil^+ \stackrel{\text{def}}{=} (\lceil\lceil P \rceil\rceil \vee \lceil P \rceil^0)$. Similarly, $\lceil\lceil P \rceil^+$ etc.
- $\lceil \; \rceil \stackrel{\text{def}}{=} \lceil 1 \rceil^0$ holds for point intervals of the form $[b, b]$.
- $ext \stackrel{\text{def}}{=} \neg \lceil \; \rceil$ holds for extended intervals $[b, e]$ with $b < e$.
- $unit \stackrel{\text{def}}{=} ext \wedge \neg(ext \frown ext)$ holds for intervals of the form $[b, b + 1]$.
- $\lceil P \rceil^1 \stackrel{\text{def}}{=} \lceil 1 \rceil^0 \frown unit$ holds for one step (two state) intervals where P is true the beginning.
- $\Diamond D \stackrel{\text{def}}{=} true \frown D \frown true$ holds provided D holds for some subinterval.
- $\Box D \stackrel{\text{def}}{=} \neg\Diamond\neg D$ holds provided D holds for all subintervals.
- $t \geq c \stackrel{\text{def}}{=} t = c \vee t > c$. Also, $t < c \stackrel{\text{def}}{=} \neg(t \geq c)$.

We now define some more complex operators.

Formula D^* represents Kleene-closure of D under the $\frown$ operator.

$$D^* \stackrel{\text{def}}{=} (\exists p. \quad (\lceil p \rceil^0 \frown true \frown \lceil p \rceil^0) \wedge \Box((\lceil p \rceil^0 \frown unit \frown (\lceil\lceil \neg p \rceil \vee \lceil \; \rceil) \frown \lceil p \rceil^0) \quad \Rightarrow \quad D) \;)$$

It is not difficult to see that

$$\sigma, [b,e] \models D^* \quad \textbf{iff} \quad b = e \ \vee \\ b < e \text{ and } \exists n, b_0, \ldots, b_n. \\ (b = b_0 \text{ and } \forall 0 \leq i < n.\ b_i < b_{i+1} \text{ and } b_n = e \text{ and } \sigma, [b_i, b_{i+1}] \models D)$$

We define some "arrow operators". These are adapted from Ravn [17] where a systematic methodology for specifying and verifying real-time systems has been expounded.

$$\lceil\lceil P \rceil\rceil^+ \xrightarrow{\delta} \lceil Q \rceil^0 \quad \stackrel{\text{def}}{=} \quad \neg\Diamond(\lceil\lceil P \rceil\rceil^+ \wedge \eta \geq \delta \frown \lceil \neg Q \rceil^0)$$

This operator specifies that if P holds continuously for δ or more time, then Q must become true within the first δ time. Moreover, Q must then persist till P persists. Note that nothing is specified about the value of Q otherwise.

$$\lceil\lceil P \rceil\rceil^+ \overset{<w}{\hookleftarrow} \lceil Q \rceil^0 \quad \stackrel{\text{def}}{=} \quad \neg\Diamond(\lceil \neg - P \rceil^0 \frown (\lceil\lceil P \rceil\rceil^+ \wedge \eta < w) \frown \lceil \neg Q \rceil^0)$$

This operator specifies that once P becomes true, for the first δ time, while P persists Q will also be true.

It is possible to define many variants of above arrow operators. We can also specify stability of a proposition as follows.

$stable(P, \delta) \stackrel{\text{def}}{=} \Box(\lceil \neg - P \rceil^0 \frown \lceil P \rceil^0 \frown (\eta < \delta) \ \Rightarrow \ \lceil\lceil P \rceil\rceil^+)$

This states that once P becomes true, it must hold for δ time. Note that this also applies to P being true initially.

Example 2.2 *We now give some examples of properties taken from model checking literature. The first three properties can be found in [22] and the last two in [1]. All these properties can be easily formulated within QDDC.*

- *If Loadreq follows Storereq then Storeserve should occur before Loadserve.*

$$\neg\Diamond(\lceil Storereq \rceil^0 \frown \lceil\lceil \neg Storeserv \rceil\rceil \wedge \\ (true \frown \lceil Loadreq \rceil^0 \frown true \frown \lceil Loadserve \rceil^0))$$

- *If A is holding the bus and B is requesting, then it is not possible for A to release control of the bus and get it back, before B gets control of the bus. This is called Request Opportunity.*

$$\Box(\lceil Ahold \wedge Breq \rceil^0 \frown \lceil\lceil \neg Bhold \rceil\rceil \ \Rightarrow \ \neg\Diamond(\lceil\lceil \neg Ahold \rceil \frown ptpAhold))$$

- *If A requests the bus and the bus is free, then A gets control of the bus within 5 clock cycles unless B or C request the bus in the mean-time.*

$$\Box(\lceil Areq \wedge Busfree \rceil^0 \frown \lceil\lceil \neg(Breq \vee Creq) \rceil\rceil \wedge \eta = 4 \ \Rightarrow \ \Diamond \lceil Agrant \rceil^0)$$

- *If speed sensors indicate impossibly high speed at least three times and no RESET events occur, then ERRPAR will be generated.*

$$\Box(((\Sigma Imphigh \geq 2) \frown \lceil Imphigh \rceil^0 \wedge \lceil\lceil \neg Reset \rceil\rceil) \ \Rightarrow \ \Diamond \lceil ERRPAR \rceil^0)$$

- *If four SPEEDSENS events occur between two CLOCK500 events (while there is no RESET) then ERRPAR event is generated.*

$$\Box(\lceil CLOCK500\rceil^0 \frown unit \frown ((\Sigma SPEEDSENS \geq 3 \frown \lceil SPEEDSENS\rceil^0 \wedge [\![\neg CLOCK500 \wedge \neg RESET]\!]) \Rightarrow \Diamond ERRPAR$$

Note that the natural language specification is quite ambiguous in the following sense.

1. *If RESET occurs simultaneously with CLOCK500 followed by four SPEEDSENS should ERRPAR be generated? Our formula requires this.*
2. *If the fourth SPEEDSENS occurs simultaneously with next CLOCK500 should there be an ERRPAR? Our formula requires this too.*
3. *If SPEEDSENS occurs simultaneously with first CLOCK500 should this be counted? Our formula ignores this signal.*
4. *If RESET occurs simultaneously with fourth SPEEDSENS should ERRPAR be generated? Our formula does not require it to be generated.*

QDDC can specify safety and bounded-liveness properties of systems. Since it only specifies finite sequence of states, it cannot deal with general liveness properties. There are many extensions addressing this problem [11].

3 Mine pump

A mine has water seepage which must be removed by operating a pump. If the water is above danger-mark, an alarm must be sounded. There is a high water sensor. In response to water being high, a pump may be operated. The pump must not operate if the water is not high. The mine also has pockets of methane which escape. Presence of methane is detected by a sensor. When there is methane, alarm must be sounded. Moreover, all electrical activity including the pump must be shut down to prevent explosion.

We shall present a model the mine pump system in QDDC and establish that under suitable assumptions the water level never becomes dangerous.

- HH_2O Water level is high.
- DH_2O Water level is dangerous.
- HCH_4 Methane level is high.
- $Alarm$ Alarm is on.
- $PumpOn$ The pump is operating.

Alarm Control The alarm will sound within δ seconds of water level becoming dangerous. Alarm will persist till the water level is dangerous.

$$[\![\lceil DH_2O \rceil]\!]^+ \xrightarrow{\delta} \lceil Alarm \rceil^0$$

Similarly for Methane.

$$[\![\lceil HCH_4 \rceil]\!]^+ \xrightarrow{\delta} \lceil Alarm \rceil^0$$

Moreover, the alarm will stop within δ seconds, once the Methane and the Water levels are safe.

$$[\![\lceil \neg HCH_4 \wedge \neg DH_2O \rceil]\!]^+ \xrightarrow{\delta} \lceil \neg Alarm \rceil^0$$

The conjunction of above three formulae will be called $Alarmcontrol$.

Water Seepage Assumptions High water level occurs before Danger water level. (Sensor is reliable.)

$$(As_1) \qquad \Box([\![\lceil DH_2O \ \Rightarrow HH_2O \rceil]\!]^+)$$

It takes at least w seconds for the water level to turn dangerous after reaching the high water level

$$(As_2) \qquad \Box([\![\lceil HH_2O \rceil]\!]^+ \overset{<w}{\hookleftarrow} \lceil \neg DH_2O \rceil^0)$$

This property specifies a minimum separation of w time between water level becoming high and its becoming dangerous. The value of w must be calculated based on estimates of rate of water seepage.

Pump control Pump enabling condition: It is safe to operate the pump only when water is high and there is no methane.

$$SafePump \quad \overset{\text{def}}{=} \quad HH_2O \wedge \neg HCH_4$$

Pump is started within δ seconds of it being enabled.

$$[\![\lceil SafePump \rceil]\!]^+ \xrightarrow{\delta} \lceil PumpOn \rceil^0$$

Pump is stopped within δ seconds of being disabled.

$$[\![\lceil \neg SafePump \rceil]\!]^+ \xrightarrow{\delta} \lceil \neg PumpOn \rceil^0$$

Conjunction of above is called $Pumpcontrol$

Pump Capacity Assumption Pump can bring water level down below high water level within ϵ seconds (from any starting condition).

$$(As_3) \quad \overset{\text{def}}{=} \quad [\![\lceil PumpOn \rceil]\!]^+ \xrightarrow{\epsilon} \lceil \neg HH_2O \rceil^0$$

Methane Release Assumptions Between two occurances of Methane Release there is at least ζ seconds.

$$(As4) \quad \stackrel{\text{def}}{=} \quad \Box(\lceil\lceil HCH_4 \rceil \frown \lceil\lceil \neg HCH_4 \rceil \frown \lceil HCH_4 \rceil^0 \Rightarrow \eta > \zeta)$$

The high methane level lasts at most κ time units.

$$(As5) \quad \stackrel{\text{def}}{=} \quad \Box(\lceil\lceil HCH_4 \rceil\rceil \Rightarrow \eta < \kappa)$$

$$(As_6) \quad \stackrel{\text{def}}{=} \quad (2 * (\delta + \epsilon) + \kappa) < w < \zeta$$

Verification Condition

$$(As_1 \wedge As_2 \wedge As_3 \wedge As_4 \wedge As_5 \wedge As_6 \wedge PumpControl \wedge Alarmcontrol) \Rightarrow (\Sigma DH_2O = 0)$$

The reader is urged to convinced himself that the above verification condition is indeed valid. Liu Zhiming [9] has established correctness of a similar (dense time) specification of Mine pump using the proof rules of Duration calculus. The proof, however, is complex. In Section 5 we shall discuss how this condition can be automatically verified using a validity checker for QDDC.

4 Decidability of QDDC

The following theorem characterises the sets of models of a QDDC formula. Let $pvar(D)$ be the finite set of propositional variables occurring within a QDDC formula D. Let $VAL(Pvar) = Pvar \to \{0,1\}$ be the set of valuations over $Pvar$.

Theorem 4.1 *For every QDDC formula D, we can effectively construct a finite state automaton $A(D)$ over the alphabet $VAL(pvar(D))$ such that for all $\sigma \in VAL(pvar(D))^*$,*

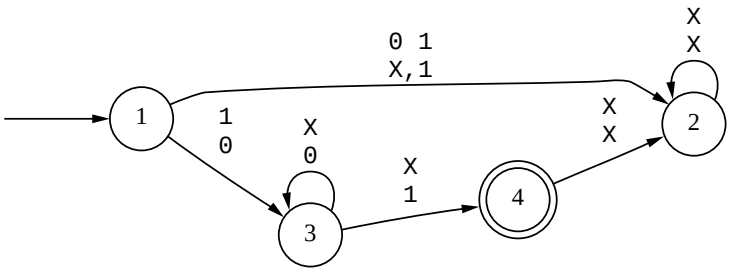
$$\sigma \models D \quad \textbf{iff} \quad \sigma \in L(A(D))$$

A proof of this theorem is given later in this Section. We first consider some of its consequences.

Corollary 4.2 *Satisfiability (validity) of QDDC formulae is decidable.*
Proof outline For checking satisfiability of $D \in QDDC$ we can construct the automaton $A(D)$. A word satisfying the formula can be found by searching for an accepting path within $A(D)$. Such a search can be carried out in time linear in the size (number of nodes + edges) of $A(D)$ by depth-first search. Similarly, a counter-model of the formula can be found by search for a nonempty path to a non-final state. •

The above corollary shows how we can use the automaton construction to visualise models and counter models of a QDDC formula, and to establish validity/unsatisfiability. This construction has been implemented into a tool called DCVALID.

Example 4.3 *The property of Example 2.1 can be stated in QDDC as formula* $\lceil P \rceil^0 \frown [\lceil \neg Q \rceil \frown \lceil Q \rceil^0$. *The automaton corresponding this formula is given below. Each edge is labelled with a column vector giving truth values of variables* P, Q *as in Example 2.1. Also, letter* X *is used to denote either* 0 *or* 1. *Note that the automaton is minimal, deterministic and total.*



From this automaton, we can see that a model of least length for the formula is as follows.

```
P  1 X
Q  0 1
```

A counter model of least length is given below. (Empty words are not considered models in QDDC.)

```
P  1
Q  0
```

We now establish that QDDC can encode extended regular expressions. Let $Pvar = \{p_1, \ldots, p_n\}$. As our alphabet we consider $VAL(Pvar)$. Each element $v \in VAL(Pvar)$ can be denoted canonically by a propositional formula $\overline{v} = \bigwedge(p_i \mid v(p_i) = 1) \wedge \bigwedge(\neg p_j \mid v(p_j) = 0)$. An extended regular expression over $VAL(Pvar)$ has the syntax

$$\epsilon \mid \overline{v} \mid re_1 \cdot re_2 \mid re_1 + re_2 \mid re_1^* \mid \neg re_1 \mid re_1 \cap re_2$$

In addition to usual regular expression operators, $\neg re$ denotes complement of regular language re and $re_1 \cap re_2$ denotes intersection of regular languages re_1 and re_2. Let $Lang(re)$ denote the language (set of words) of re.

We give a linear-time encoding of Extended regular expressions in QDDC as follows.

$$\kappa(\epsilon) = \lceil\ \rceil$$
$$\kappa(\overline{v}) = \lceil\overline{v}\rceil^0 \frown unit$$
$$\kappa(re_1 \cdot re_2) = \kappa(re_1) \frown \kappa(re_2)$$
$$\kappa(re_1 + re_2) = \kappa(re_1) \vee \kappa(re_2)$$
$$\kappa(re^*) = (\kappa(re))^*$$
$$\kappa(\neg re) = \neg\kappa(re)$$
$$\kappa(re_1 \cap re_2) = \kappa(re_1) \wedge \kappa(re_2)$$

In our encoding, model $\sigma \in VAL(Pvar)^+$ denotes the word $\sigma\downarrow = \sigma[0 : \#\sigma - 1]$, i.e. σ with last element removed. Let $Lang(D) = \{\sigma\downarrow \mid \sigma \models D\}$.

Proposition 4.4 *$Lang(re) = Lang(\kappa(re))$* •

Complexity Meyer and Stockmeyer have established a nonelementary lower bound on the size of automata accepting the language of extended regular expressions. Because of Proposition 4.4, this implies that there is a non-elementary *lower bound* on the size of the automaton $A(D)$ accepting word models of a QDDC formula D. In the worst case, the complexity of the output automaton can increase by one exponent for each alternation of $\neg$ and $\frown$ operators. However, such blowup is rarely observed in practice and we have been able to check validity of many formulae which are 5-6 pages long with our tool DCVALID [12].

4.1 Embedding QDDC in Monadic Logic Over Finite Words

The rest of this section is devoted to proving Theorem 4.1.

Let $QDDCR$ be the subset of $QDDC$ where measurement formulae of the form η *op* c and ΣP *op* c are not used.

Lemma 4.5 *For every $D \in QDDC$ we can effectively construct a $\alpha(D) \in QDDCR$ such that $\models D \Leftrightarrow \alpha(D)$.*

Proof outline Required $\alpha(D)$ can be constructed from D by recursively applying the following transformation.

$\alpha(\lceil P\rceil^0) = \lceil P\rceil^0$, $\alpha(\lceil\lceil P\rceil) = \lceil\lceil P\rceil$.
$\alpha(D_1 \wedge D_2) = \alpha(D_1) \wedge \alpha(D_2)$
$\alpha(D_1 \frown D_2) = \alpha(D_1) \frown \alpha(D_2)$
$\alpha(\neg D) = \neg(\alpha(D))$
$\alpha(\exists p.D) = \exists p.\alpha(D)$
$\alpha(\eta = 0) = \lceil\ \rceil$
$\alpha(\eta = 1) = unit$
$\alpha(\eta = m + 1) = unit \frown \alpha(\eta = m)$
$\alpha(\eta > m) = \alpha(\eta = m) \frown ext$
$\alpha(\Sigma P = 0) = \lceil\ \rceil \vee \lceil\lceil\neg P\rceil$

$$\begin{aligned}
\alpha(\Sigma P = 1) &= \alpha(\Sigma P = 0) \frown (\lceil\lceil P \rceil \wedge \mathit{unit}) \frown \alpha(\Sigma P = 0) \\
\alpha(\Sigma P = m+1) &= \alpha(\Sigma P = 1) \frown \alpha(\Sigma P = m) \\
\alpha(\Sigma P > m) &= \alpha(\Sigma P = m) \frown (\Diamond \lceil\lceil P \rceil)
\end{aligned}$$

•

Monadic Second-order Logic over Finite Words Syntax of MSO over finite words:

$$x < y \mid x = y \mid x \in Z \mid \phi \wedge \psi \mid \neg\phi \mid \exists x.\phi \mid \exists Z.\phi$$

Variables $x, y, z, \ldots$ (in lowercase) are called *individual variables* and $X, Y, Z, \ldots$ (in uppercase) are called *set variables* or *monadic predicates*.

We use $\phi(x_1, \ldots, x_n, Z_1, \ldots, Z_m)$ to denote that formula has at most the individual variables $x_1, \ldots, x_n$ and the monadic predicates $Z_1, \ldots, Z_n$ occurring free in it.

Recall that $Val(Pvar) = Pvar \to \{0, 1\}$. A *word model* over $VAR = (x_1, \ldots, x_n, Z_1, \ldots, Z_m)$ is $M = (\sigma, I)$ where $\sigma \in Val(Z_1, \ldots, Z_n)^*$ and $I(x_i) \in dom(\sigma)$ and $I(Z_j) \subseteq dom(\sigma)$. We define the semantics of MSO formulae.

$$\begin{aligned}
&\sigma, I \models x < y \quad \textbf{iff} \quad I(x) < I(y) \\
&\sigma, I \models x = y \quad \textbf{iff} \quad I(x) = I(y) \\
&\sigma, I \models x \in P \quad \textbf{iff} \quad \sigma(I(x))(P) = 1 \\
&\sigma, I \models \exists x.\phi \quad \textbf{iff} \quad \sigma, I' \models \phi \text{ for some } I' \text{ which is } x\text{-variant of } I \\
&\sigma, I \models \exists Z.\phi \quad \textbf{iff} \quad \sigma', I \models \phi \text{ for some } \sigma' \text{ which is } Z\text{-variant of } \sigma
\end{aligned}$$

The boolean combinators have their usual meaning.

In the following theorem we translate $QDDCR$ formula D into a formula $\phi(D)$ where the free variables $pvar(D)$ become monadic predicates of $\phi(D)$ and $\phi(D)$ has no individual free variables. The translation preserves models.

Theorem 4.6 *For every $D \in QDDCR$ we can effectively construct $\phi(D) \in MSO(pvar(D))$ such that for all $\sigma \in Val(pvar(D))^+$.*

$$\sigma \models D \quad \textbf{iff} \quad \sigma \models \phi(D)$$

Proof outline We define $\beta(D)$ as follows.

$$\begin{aligned}
&\beta(\lceil P \rceil^0) \stackrel{\text{def}}{=} x = y \wedge x \in P \\
&\beta(\lceil\lceil P \rceil) \stackrel{\text{def}}{=} x < y \wedge \forall z.(x \leq y < z \Rightarrow z \in P) \\
&\beta(D_1 \wedge D_2) \stackrel{\text{def}}{=} \beta(D_1) \wedge \beta(D_2) \\
&\beta(\neg D) \stackrel{\text{def}}{=} x \leq y \wedge \neg\beta(D) \\
&\beta(D_1 \frown D_2) \stackrel{\text{def}}{=} \exists z.\beta(D_1)[z/y] \wedge \beta(D_2)[z/x] \\
&\beta(\exists p.D) \stackrel{\text{def}}{=} \exists p.\beta(D)
\end{aligned}$$

The desired formula $\phi(D)$ is given by

$$\exists x, y.first(x) \wedge last(y) \wedge \beta(D)$$

We omit the proof that $\sigma \models D$ in QDDCR if and only if $\sigma \models \phi(D)$ in MSO. This should be obvious as the formula essentially encodes the semantics of QDDCR.

Note that empty word $\epsilon \not\models \phi(D)$. Hence, D and $\alpha(D)$ have exactly same word models. •

The decidability of MSO over finite words has been well-studied.

Theorem 4.7 (Buchi60-Elgot61) *For every formula $\phi \in MSO(VAL(Pvar)^+)$ we can effectively construct a finite state automaton $A(\phi)$ with alphabet $VAL(Pvar)$ such that*

$$\sigma \models \phi \quad \textbf{iff} \quad \sigma \in L(A(\phi))$$

We omit the proof of this well-known theorem [2, 3]. The reader can refer to excellent treatment of this in the survey articles by Thomas [18].

Theorem 4.1 follows immediately from Lemma 4.5, Theorem 4.6 and Theorem 4.7.

5 DCVALID

The reduction from formulae of QDDC to finite state automata as outlined in Theorem 4.1 has been implemented into a tool called DCVALID [12]. The tool generates automaton for a formula and also checks for the validity of formulae as in corollary 4.2. The automaton in Example 4.3 was automatically generated from the formula by the tool.

DCVALID is built on top of MONA [7]. MONA is an efficient and sophisticated implementation of the automata-theoretic decision procedure of Buchi and Elgot for monadic logic over finite words (MLSTR). MONA represents the whole transition table of an automaton as a multi-terminal BDD pointing to the next state. This allows it to represent complex automata with large alphabets. It has a library of procedures for operations such as determinisation, minimisation, projection and product of automata. Using these, MONA can construct and analyse automata for many large MLSTR specifications .

Much of the functionality and power of DCVALID is derived from underlying use of MONA. The tool basically works by reducing the QDDC formula into MLSTR as outlined in Lemma 4.5 and Theorem 4.6. Then, MONA is invoked to generate and analyse the automaton.

The mine pump specification of Section 3 can be given to DCVALID. The exact input to the tool is listed in Appendix A. Note that the tool cannot work with symbolic parameters $\delta, \kappa \ldots$ and these have to be given concrete values.

For $\delta = 2$, $w = 17$, $\epsilon = 7$, $\zeta = 25$, $\kappa = 2$ the tool generates a counter example given below. In this example, horizontal axis is time and value of each propositional variable is given as 1 (true), 0 (false) or X (either 0 or 1).

```
A counter-example of least length (19) is:
HH2O            X 1111111111111111110
DH2O            X 0000000000000000010
HCH4            X 0000000001000000000
ALARM           X XX0000000XXX00000XX
PUMPON          X 0011111110001111110
```

For $\delta = 2,\ w = 21,\ \epsilon = 7,\ \zeta = 25,\ \kappa = 2$ the tool reports that the verification condition is satisfied as follows.

```
Formula is valid
```

A number examples from Duration Calculus literature have been checked using DCVALID after suitably encoding them is discrete time. We give a brief summary of the performance of DCVALID on these examples. The experiments were conducted on 233Mhz Pentium Pro processor PC system with 128 Mbytes of main memory and running Linux 2.2.17 kernel.

The decision procedure works by constructing automata for sub-formulae in a bottom up manner. We record the maximum number of states and maximum number of BDD nodes encountered during the whole process. This is important because the tool always keeps automata in minimal, deterministic form at all stages. For a valid formula, the final automaton will always have exactly two states. It is the blow up encountered during its construction which must be recorded.

Example	Formula Size (lines)	Time (sec)	Automaton States (Maximum)	BDD Nodes (Maximum)
Minepump1	72	13.18	4056	30547
Minepump2	72	3.77	3141	22127
Minepump3	72	39.57	10205	62794
Minepump4	72	71.57	15887	112948
Lift Control	225	10.61	9086	77986
Fischer Protocol	236	34.78	10160	302876
Delay Insensitive Oscillator	62	165.44	168848	1220753
5-cell Synchronous Arbiter	74	0.45	58	6799

The exact specification and constants used in above trials can be found on the DCVALID web site [12]. In these trials Minepump1 refers to minepump with $\delta = 2,\ w = 17,\ \epsilon = 7,\ \zeta = 25,\ \kappa = 2$. Minepump2 refers to minepump with $\delta = 2,\ w = 21,\ \epsilon = 7,\ \zeta = 25,\ \kappa = 2$. Minepump3 is minepump with $\delta = 2,\ w = 40,\ \epsilon = 7,\ \zeta = 100,\ \kappa = 2$ where as Minepump4 is minepump with $\delta = 4,\ w = 40,\ \epsilon = 7,\ \zeta = 100,\ \kappa = 2$.

Note that the performance is very dependent on the timing constants used in the specification and the number of lines (or characters) of the source formula is not a good measure of the complexity of the automaton generated. In our trials, the automaton construction could not be completed for the following one line formula due to large requirements of memory (leading to thrashing).

$$\Box(\eta \leq 60 \ \Rightarrow\ \Sigma Leak \leq 3)$$

This formula is part of the the gas burner specification, which was the only example where the tool ran out of resources. One of the longest examples considered is the verification of a streaming audio-video protocol using DCVALID by Wang and Xu [23]. The system model was about 6 pages of QDDC formulae.

Thus, in spite of non-elementary lower bounds, the decision procedure appears to be usable and it performs well on many practical examples considered.

6 Discussion

In this paper, we have presented logic Quantified Discrete-time Duration Calculus (QDDC). We have illustrated its use in modelling timed systems by an example of a mine pump. We have also established the decidabilility of QDDC by giving a linear time embedding of this logic into Monadic Logic over Finite Strings (MLSTR). Finally, we have shown that QDDC has exactly the expressive power of Extended Regular Expressions by giving their linear-time embedding into QDDC. The reduction from QDDC to MLSTR has been implemented into a tool called DCVALID [12]. We have presented experimental results from verification of several QDDC specifications using DCVALID. These include the mine pump example presented earlier in this paper, a delay-insensitive oscillator and modelling of a synchronous bus arbiter.

DCVALID is a useful tool for the analysis of QDDC specifications. It can be used to visualise complex formulae as it has the ability to present models and counter-models of formulae. The tool can also represent models of a formula as a finite state automaton which can then be examined. This is important since we have found that often the specification formula does not adequately formalise the desired property. Some help in visualising the specification formulae is invaluable, especially with rich logics like QDDC. Finally, the constructed the automaton can be considered as synthesised program for the logical specifaction. The automaton can be translated into a programming notation. Our tool provides some support for this.

DCVALID can be used to model check designs by modelling the designs in QDDC and establishing their desired properties as logical consequnces of the design formula. This approach was used in the mine pump example. However, this is not the most convenient approach as modelling large and complex designs as formulae is prohibitively cumbersome, and it leads to formulae which are too large to handle. Usually, designs are large where as properties are small.

To alliviate this, we have recently extended DCVALID to support model checking [13]. The basic idea is that the automaton of a formula can be used as a kind of synchronous observer or monitor. This can be executed in parallel with the system and it detects the failure of formula during the execution by reaching appropriate error state. Existing model checking tools can be used to search for executions leading to such error states in the composite model+observer system. The approach allows us to model check formulae from a much richer logic CTL[DC] against SMV, VIS and Esterel designs [14]. Unlike QDDC, logic CTL[DC] can express liveness as well as branching time properties.

Considering all this practical use of DCVALID, the issue of complexity of QDDC merits discussion and further investigation. As stated in Section 2, QDDC has a non-elementary lower bound on the complexity of validity checking. Such high complexity can potentially be a source of in-feasibility and may sound

hopeless. However, this complexity is rarely seen in practice. In fact, as reported in Section 5, we have been able to verify many formulae which are 5-6 pages long with our tool. (see Pandya [12] for details.) At the same time, there are also very small formulae which cannot be checked as they require prohibitively large resources. The textual length of the specification is not a very good measure of the complexity of validity checking.

Related Work QDDC is a Discrete-time variant of Propositional Duration Calculus [24]. It inherits the richness and covenience of Duration Calculus in specifying complex requirements. QDDC is closely related to the Interval Temporal Logic (ITL) of Moszkowski [10]. It can be considered as a form of propositional fragment of Moszkowski's logic, although the use of measurement formulae like $\Sigma P <= c$ is not found in ITL. The quantification construct appeared in [11]. As shown here, it can be used to define iteration and more general tail recursion. It plays a crucial role in the Duration Calculus based compositional semantics of languages like Esterel [16] and Timed CSP [15].

Decidability of Propositional Duration Calculus for both discrete and dense time was established by Zhou, Hansen and Sestoft [26, 4] by reducing Duration Calculus formulae to star-free regular expressions. By contrast, our logic has the full power of Extended Regular Expressions.

We have chosen to factor our decidability proof by reduction to Monadic Logic over Finite Words. The later has an automata theoretic decision procedure as shown by Buchi[2] and Elgot [3]. The existence of efficient and sophisticated implementation of this decision procedure in tools such MONA [7] can be exploited to construct tools for QDDC. We have used this approach to construct our validity checker, DCVALID, for QDDC formulae.

The reduction from QDDC to QDDCR was considered by Hansen, Zhou [5] and by Franzle [6]. The embedding of QDDCR into Monadic Logic over Finite Words is an instance of a much more general linear-time embedding of Propositional Interval Tense Logic over any arbitrary class of linear orders into Monadic Logic over the same class of linear orders. This embedding was given by Pandya [11] for studying decidability and expressive power of Interval Tense Logic. In the present case, the class of linear orders consists of those obtained by taking initial prefix of natural numbers with standard order. This line of work has been further developed by Rabinovich [19].

There have been other implementations of decision procedures for DDC (without quantification). Especially notable is the early BDD-based DDC validity checker by Skakkebaek and Sestoft [20, 21], which has also been integrated into a PVS based proof assistant for Duration Calculus. But the scale of examples covered by DCVALID has not been considered elsewhere, and based on published performance figures [21], DCVALID seems to be an order of magnitude faster. (We must note these figures are not from trials on the same platform.) As we mentioned, DCVALID is built on top of MONA [7]. DCVALID derives much of its functionality and power from the underlying use of MONA [8].

Finally, some of our recent results show that the automata theoretic decision procedure presented here can in fact be extended to dense-time using timed and hybrid automata. DCVALID is being extended to a dense-time logic called Interval Duration Logic which can be model checked. This will be a topic of some forthcoming papers.

References

[1] F. Balarin, H. Hsieh *et al*, Formal Verification with POLIS Co-design environment for control-dominated embedded systems - A tutorial, December, 1996.

[2] J.R. Buchi, Weak second-order arithmetic and finite automata, *Z. Math. Logik Grundl. Math.* **6**, 1960.

[3] C.C. Elgot, Decision problems of finite automata design and related arithmetics, *Trans. Amer. Math. Soc.* **98**, 1961.

[4] M.R. Hansen, Model-Checking Duration Calculus, *Formal Aspects of Computing*, 1994.

[5] M.R. Hansen and Zhou Chaochen, Duration Calculus: Logical Foundations, *Journal of Formal Aspects of Computing* **9**, 1997.

[6] M. Franzle, Decidability of Duration Calculi on Restricted Model Classes, Technical Report, July 1996.

[7] J.G. Henriksen, J. Jensen, M. Jorgensen, N. Klarlund, B. 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*, LNCS 1019, 1996.

[8] N. Klarlund, A. Møller and M.I. Schwartzbach, MONA implementation secrets, *to appear in Proc. CIAA 2000*, 2000.

[9] Z. Liu, Specification and verification in the duration calculus, in M. Joseph (ed.), *Real-time Systems: Specification, Verification and Analysis*, Prentice Hall, 1996.

[10] B. Moszkowski, A Temporal Logic for Multi-Level Reasoning about Hardware, in *IEEE Computer*, **18**(2), 1985.

[11] P.K. Pandya, Some Extensions to Mean-Value Calculus: Expressiveness and Decidability, in *Proc. CSL'95*, Paderborn, Germany, LNCS 1092, Springer-Verlag, 1996.

[12] P.K. Pandya, DCVALID User Manual, Tata Institute of Fundamental Research, Bombay, 1997. (Available in revised version at http://www.tcs.tifr.res.in/~pandya/dcvalid.html)

[13] P.K. Pandya, Model checking CTL[DC], Technical Report TCS-00-PKP-2, Tata Institute of Fundamental Research, July 2000.

[14] P.K. Pandya, Model checking CTL[DC] specifications of SMV, Verilog and Esterel Designs, Technical Report, TCS-00-PKP-3, Tata Institute of Fundamental Research, September 2000.

[15] P.K. Pandya and H.V. Dang, A Duration Calculus of Weakly Monotonic Time, in *Proc. FTRTFT'98*, Lyngby, Denmark, (eds.) A.P.Ravn and H. Rischel, LNCS 1486, Springer-Verlag, 1998.

[16] P.K. Pandya, Y.S. Ramakrishna, R.K. Shyamasundar. A Compositional Semantics of Esterel in Duration Calculus. In *Proc. Second AMAST workshop on Real-time Systems: Models and Proofs*, Bordeux, June, 1995.

[17] A.P. Ravn, Design of Real-time Embedded Computing Systems, Department of Computer Science, Technical University of Denmark, 1994.

[18] W. Thomas, Automata, Logic, Languages.

[19] A. Rabinovich, Expressive Completeness of Duration Calculus. *Information and Computation*, Vol. 156, No. 1/2, pp. 320-344, 2000.

[20] J.U. Skakkebaek, A verification Assistant for Real-time Logic, Ph.D. Thesis, Department of Computer Science, Technical University of Denmark, 1994.

[21] J.U. Skakkebaek, Checking Validity of Duration Calculus Formulas, Technical Report ID/DTH JUS 3/1, Department of Computer Science, Technical University of Denmark, 1994.

[22] G. Swamy, VIS technology transfer course, University of Berkeley, May 1996. Available on Web at `http://www-cad.eecs.Berkeley.EDU/~vis/visttc.html`.

[23] J. Wang and Q. Xu, Modelling and verification of a network player system with DCVALID, In proc. *First asia-pacific conference on quality software*, Hong-kong, 2000.

[24] Zhou Chaochen, C.A.R. Hoare and A.P. Ravn, A Calculus of Durations, *Info. Proc. Letters*, **40**(5), 1991.

[25] Zhou Chaochen, M.R. Hansen, A.P. Ravn and H. Rischel, Duration Specification for Shared Processors, LNCS 571, Springer-Verlag, 1993.

[26] Zhou Chaochen, M.R. Hansen and P. Sestoft, Decidability and Undecidability Results for Duration Calculus, In *STACS'93*, LNCS 665, Springer-Verlag, 1993.

A The Mine pump Specification in DCVALID

```
-- mine pump controller

var HH2O, DH2O, HCH4, ALARM, PUMPON ;

const delta = 2, w = 17, epsilon=7 , zeta= 25, kappa=2 ;
-- delta   response time of PUMP and ALARMS after trigger
-- epsilon time taken by pump to bring water level to below HH2O
-- w       time taken for water level to be dangerous after high
-- zeta    minimum separation between two methane leaks.
-- kappa   maximum duration of methane leak.
-- correctness under 2*(epsilon + delta) + kappa < w < zeta
-- there is no dangerous wather level.

-- Alarm control

define alarm1 as
  {DH2O} =delta=> {ALARM} ;

define alarm2 as
  {HCH4} =delta=> {ALARM} ;

define alarm3 as
  { !HCH4 && !DH2O } =delta=> {!ALARM}   ;

-- Water seepage assumptions
define water1 as
   [] ( [[ DH2O => HH2O ]] ) ;

define water2 as
   {HH2O} <=w= {! DH2O}  ;

-- Pump control

define pump1 as
  { HH2O && !HCH4 } =delta=> {PUMPON}  ;

define pump2 as
  { !HH2O || HCH4 } =delta=> {!PUMPON} ;

-- Pump capacity assumption

define pumpcap1 as

  {PUMPON} =epsilon=> {!HH2O} ;
```

```
--  Methane Release assumptions

define methane1 as
   [] ( [HCH4]^[!HCH4]^<HCH4> => slen > zeta ) ;

define methane2 as
   [] ( [[HCH4]] => slen < kappa ) ;


infer
( alarm1 && alarm2 && alarm3 && water1 && water2 &&
  pump1 && pump2 && pumpcap1 && methane1 && methane2
)
=>  sdur DH2O = 0
.
```