Model Checking CTL*[DC]

Paritosh K. Pandya *

Tata Institute of Fundamental Research
Homi Bhabha Road, Colaba, Mumbai 400005, India
pandya@tcs.tifr.res.in

Abstract. We define a logic called CTL*[DC] which extends CTL* with ability to specify past-time and quantitative timing properties using the formulae of Quantified Discrete-time Duration Calculus (QDDC). Alternately, we can consider CTL*[DC] as extending logic QDDC with branching and liveness.
As our main result, we show a reduction of CTL*[DC] model checking problem to model checking of CTL* formulae. The reduction relies upon an automata-theoretic decision procedure for QDDC. Moreover, it preserves the subsets CTL and LTL of CTL*. The reduction is of practical relevance as model checking of CTL* as well as its subsets CTL and LTL are well studied and even implemented into a number of tools. We briefly discuss an implementation of a model checking tool for CTL[DC] called CTLDC, based on the above theory. CTLDC can model check SMV, Verilog and Esterel designs using tools SMV, VIS and Xeve, respectively.

1 Introduction

Logic CTL* is an expressive logic for the specification of properties of transition systems [8]. It has path quantifiers for specifying branching time properties as well as temporal operators for specifying how state of the system evolves along execution paths. For example, the following formula states that on all execution paths proposition P will hold infinitely often. (We only provide an intuitive explanation of what the properties states. A precise definition of the syntax and semantics of CTL* operators is given in Section 3.)

AGF P

Model checking algorithms for verifying CTL* properties of finite state transition systems are well studied [8]. Moreover, subsets CTL [4,5] and LTL [25] of CTL* have also been formulated and thoroughly investigated. Symbolic model checking algorithms for verifying formulae of these sub-logics have been implemented in tools such as SMV [18], VIS [2] and TLV [12].

In spite of its expressive abilities, there are situations where CTL* is restrictive. It has long been recognised [17] that availability of past modalities in

* Partially supported by the UNU/IIST offshore project *Semantics and verification of real-time programs using Duration Calculus: Theory and Practice*

T. Margaria and W. Yi (Eds.): TACAS 2001, LNCS 2031, pp. 559–573, 2001.

temporal logics considerably facilitates formulation of complex properties. Secondly, specification of reactive systems must often deal with quantitative timing constraints [9]. In this paper, we will address these issues in an integrated fashion.

Discrete-time Duration Calculus (QDDC) [21,22] is a highly expressive logic for specifying quantitative timing properties of finite sequences of states (behaviours). It is closely related to the Interval Temporal Logic of Moszkowski [19] and Duration Calculus of Zhou *et al* [26]. 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.

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

Here, $\Box$ modality ranges over all fragments of a behaviour. Operator $\frown$ is like concatenation (fusion) of behaviour fragments and $[\lceil \neg Q \rceil$ 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 convenient and highly expressive formalism for specifying quantitative timing properties. However, it cannot specify liveness or branching.

An automata-theoretic decision procedure allows checking of satisfiability (validity) of QDDC formulae [22]. This algorithm has been implemented into a tool called DCVALID [21]. The tool is built on top of MONA [11], which is an efficient and sophisticated BDD-based implementation of the Buchi-Elgot automata-theoretic decision procedure for Monadic Logic over Finite Words [3,7]. (See [13] for a recent paper on MONA.)

In this paper, we propose a straight-forward extension of CTL* where, in place of propositions, formulae of Quantified Discrete-time Duration Calculus $QDDC$ can be asserted within CTL* formulae. A $QDDC$ formula D holds for a node of a computation tree provided the unique path from the root of the tree to the node satisfies D. Thus, a QDDC formula D allows specification of the "past" of the node. Operators of CTL* allow specification of branching and liveness properties.

For example, the following formula states that on all execution paths QDDC formula D will become true infinitely often.

$$\mathbf{AGF}\ D$$

The following formula states that once there is overload for 5 steps, there will be alarm until reset occurs.

$$\mathbf{AG}\ (true \frown ([\lceil Overload \rceil \wedge \eta = 5) \quad \Rightarrow \quad \mathbf{A}(alarm\ \mathcal{U}\ reset))$$

Logic QDDC provides a useful extension to the expressive power of CTL* by allowing past-time and quantitative timing properties to be expressed. It also significantly increases the expressive power of QDDC which is unable to specify

liveness properties such as infinitely often D or branching. In a separate note [24], we list a number of real-life properties collected from model checking literature which are stated to be hard to formulate in CTL. We show that these properties can be easily captured using CTL[DC].

As our main result, we show a reduction of CTL*[DC] model checking problem to the model checking of pure CTL* formulae by effectively transforming the transition system and the property. This transformation relies on the automata-theoretic decision procedure for QDDC. Thus, in effect, we show that by combining the model-checking procedures for CTL* and QDDC, we obtain a model-checking procedure for CTL*[DC].

Our reduction of CTL*[DC] model checking to CTL* model checking preserves the subsets CTL and LTL of CTL*. That is, a CTL[DC] formula reduces to a CTL formula and LTL[DC] formula reduces to an LTL formula. This reduction is of practical relevance as model checking of CTL* [8] as well as its subsets CTL [5] and LTL [16] are well studied and even implemented into a number of tools such as SMV [18,6], VIS [2] and TLV [12]. Based on this reduction, we have implemented a model checking tool for CTL[DC] called CTLDC [23]. It permits model checking of SMV, Verilog and Esterel designs using SMV [18], VIS [2] and Xeve [1] tools, respectively.

CTLDC permits well established CTL model checking tools to be used for analysing complex properties involving past and quantitative timing constraints. While there have been several theoretical formulations extending LTL and CTL with past [14,15], CTLDC constitutes perhaps the first implementation of CTL with past. Moreover, the fact that we can integrate our approach with a wide variety of design notations such as SMV, Verilog, Esterel, and tools such as SMV, VIS, Xeve shows that the approach is rather generic and easy to build from components.

The rest of the paper is organised as follows. We provide a brief overview of the logic QDDC in Section 2. The syntax and semantics of CTL*[DC] are given in Section 3. The reduction from model checking of CTL*[DC] to model checking of CTL* is given in Section 4. Finally, some examples of use of CTLDC, the model checker for CTL[DC], are described in Section 5. We conclude the paper with a brief 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 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 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

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

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. 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.

Let $\sigma, i \models P$ denote that proposition P evaluates to true at position i in σ. We omit this obvious definition. We inductively define the satisfaction of QDDC formula D for behaviour σ and interval $[b, e] \in Intv(\sigma)$ as follows.

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

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.
- $\lceil\ \rceil \stackrel{\text{def}}{=} \lceil 1 \rceil^0$ 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 $[b, b + 1]$.
- $\Diamond D \stackrel{\text{def}}{=} true \frown D \frown true$ 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, $t < c \stackrel{\text{def}}{=} \neg(t \geq c)$.
- $D^* \stackrel{\text{def}}{=} (\exists p. \ (\lceil p \rceil^0 \frown true \frown \lceil p \rceil^0) \wedge$
  $\Box((\lceil p \rceil^0 \frown unit \frown (\lceil\!\lceil \neg p \rceil\!\rceil \vee \lceil\ \rceil) \frown \lceil p \rceil^0) \quad \Rightarrow \quad D)\ )$

Formula D^* represents Kleene-closure of D under the $\frown$ operator. It states that D^* holds for interval $[b, e]$ if there exists a partition of $[b, e]$ into a sequence of sub-intervals such that D holds for each sub-interval. Each sub-interval is characterised by p holding at both endpoints and nowhere in between, i.e. satisfying the formula $(\lceil p \rceil^0 \frown unit \frown (\lceil\!\lceil \neg p \rceil\!\rceil \vee \lceil\ \rceil) \frown \lceil p \rceil^0)$. It is not difficult to see that

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

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 $Pvar$.

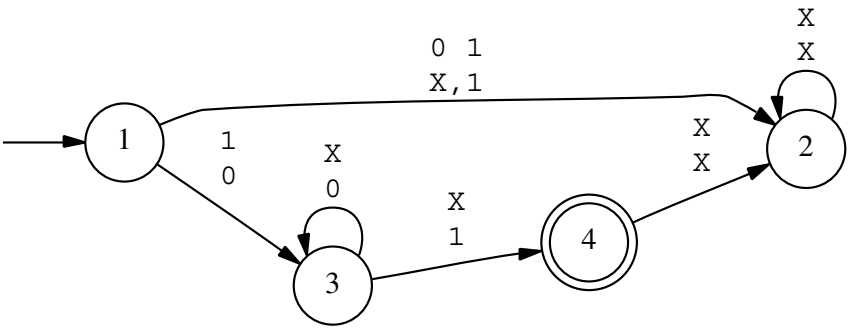
Theorem 1. *For every QDDC formula D, we can effectively construct a finite state automaton $A(D)$ over the alphabet $VAL(pvar(D))$ such that for all $\sigma \in VAL(pvar(D))^*$,*

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

We omit the proof of this theorem as it can be found elsewhere [22]. In outline, the proof relies on the following steps. Firstly, we can eliminate all measurement formulae of the form $\eta \ op \ c$ and $\Sigma P \ op \ c$ from a QDDC formula D and find an equivalent formula D' without these. Such formulae are said to belong to subset QDDCR. Next we embed QDDCR into monadic logic over finite words. Thus for every formula $D' \in QDDCR$ we construct a formula ψ of monadic logic over finite words which has the same set of models as D'. This embedding was first presented by Pandya [20]. Finally, the famous theorem due to Buchi [3] and Elgot [7] states that for every formula ψ of monadic logic over finite words, we can construct a finite state automaton which accepts exactly the word models of ψ. By combining these steps, we can obtain the automaton $A(D)$ accepting word models of QDDC formula D. □

Corollary 1. *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. □

Example 2. The property of Example 1 can be stated in QDDC as formula $\lceil P \rceil^0 \frown [\![\neg Q]\!] \frown \lceil Q \rceil^0$. 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 1. Also, letter X is used to denote either 0 or 1.



DCVALID The reduction from formulae of QDDC to finite state automata as outlined in Theorem 1 has been implemented into a tool called DCVALID [21], which also checks for the validity of formulae as in corollary 1. This tool is built on top of MONA [11,13]. MONA is a sophisticated and efficient BDD-based implementation of the automata-theoretic decision procedure for monadic logic over finite words [3,7]. DCVALID works by reducing QDDC formulae to this logic [22]. The automaton in Example 2 was automatically generated from the formula by this tool.

Complexity It must be noted 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 [21,22].

3 Logic CTL*[DC]

A transition system (also called Kripke structure) is a quadruple (S, R, L, S_0) where

S is the set of states,
$S_0 \subseteq S$ is the set of initial states.
$R \subseteq S \times S$ is the transition relation
$L : S \to VAL(Pvar)$ is the labelling function.

Recall that $VAL(Pvar) = Pvar \to \{0, 1\}$ is the set of valuations over $Pvar$. The labelling function $L(s)$ gives the truth-value of propositional variables at state s.

Syntax We have three sorts of formulae: $QDDC$ formulae, path formulae and state formulae. Let P range over propositions. Let D range over $QDDC$ formulae; α, β range over path formulae and ϕ, ψ range over state formulae.

Let $L(s) \models P$ denote that proposition P evaluates to true in state s with labelling function L. Also, for a given nonempty sequence of states $\langle s_0, \ldots, s_n \rangle$, let $L(\langle s_0, \ldots, s_n \rangle) \stackrel{\text{def}}{=} \langle L(s_0), \ldots, L(s_n) \rangle$ give the corresponding sequence of valuations in $VAL(Pvar)^+$. Hence for QDDC formula D over $Pvar$, we can define $L(\langle s_0, \ldots, s_n \rangle) \models D$ as in Section 2.

State Formulae of CTL*[DC]

$$P \mid D \mid \mathbf{A}\alpha \mid \mathbf{E}\alpha \mid \neg\phi \mid \phi \wedge \psi$$

Path Formulae of CTL*[DC]

$$\phi \mid \alpha \,\mathcal{U}\, \beta \mid \mathbf{X}\alpha \mid \alpha \wedge \beta \mid \neg\alpha$$

We can define some abbreviations for the path formulae. Let $\mathbf{F}\alpha \stackrel{\text{def}}{=} \mathit{true}\ \mathcal{U}\ \alpha$ and $\mathbf{G}\alpha \stackrel{\text{def}}{=} \neg\mathbf{F}\neg\alpha$.

Given $M = (S, R, L, S_0)$ and $s \in S$, let $Tr(M, s)$ denote the (unique) tree obtained by unfolding the state graph M starting with state s. In this tree, each node is labelled by a state from s. Moreover, a node n labelled s has as its immediate successors nodes labelled with s' for each distinct s' such that $R(s, s')$. We call such a tree a *computation tree*. Let $St(T, n)$ denote the state labelling the node n of tree T.

Given a computation tree T and an internal node n, let $hist(n)$ denote the finite sequence of states $s_0, \ldots, s_n$ labelling the nodes on the unique path from the root to n. A trajectory from n_0 is an infinite sequence of nodes $n_0, n_1, \ldots$ going into the future. Let $paths(n)$ be the set of all trajectories starting from n.

It should be noted that the label s of a node uniquely defines the subtree under it. However, distinct nodes n_1 and n_2 with same state label will have distinct $hist(n)$. The truth of formulae in our logic CTL*[DC] will depend upon both the subtree at n as well as $hist(n)$.

We now define the truth of state and path formulae. Let $T = Tr(M, s)$ be a computation tree. Let n be a node of T and let $\rho = n_0, n_1, \ldots$ be a trajectory in T. Then, the truth of state formula $T, n \models \phi$, and the truth of path formula $T, \rho \models \alpha$ are defined as follows.
State formulae:

$$
\begin{array}{l}
T, n \models P \quad \textbf{iff} \quad L(St(T, n)) \models P \\
T, n \models D \quad \textbf{iff} \quad L(hist(n)) \models D \\
T, n \models \mathbf{E}\alpha \quad \textbf{iff} \quad T, \rho \models \alpha \text{ for some } \rho \in paths(n) \\
T, n \models \mathbf{A}\alpha \quad \textbf{iff} \quad T, \rho \models \alpha \text{ for all } \rho \in paths(n)
\end{array}
$$

The boolean combinators have their usual meaning.
Path formulae: Let $\rho = n_0, n_1, \ldots$ denote a trajectory in T starting at a (not necessarily root) node n_0. For any $m \in Nat$, let ρ^m denote the suffix $n_m, n_{m+1}, \ldots$ of ρ starting with node n_m.

$$
\begin{array}{l}
T, \rho \models \phi \quad \textbf{iff} \quad T, n_0 \models \phi \\
T, \rho \models \mathbf{X}\alpha \quad \textbf{iff} \quad T, \rho^1 \models \alpha \\
T, \rho \models \alpha \,\mathcal{U}\, \beta \quad \textbf{iff} \quad \text{for some } m \in Nat, \\
\qquad T, \rho^m \models \beta, \quad \text{and} \quad T, \rho^j \models \alpha, \text{ for } j : 0 \leq j < m
\end{array}
$$

Finally,

$$
\begin{array}{l}
T \models \phi \quad \textbf{iff} \quad T, n_r \models \phi \quad \text{where } n_r \text{ is the root of the tree } T \\
M, s \models \phi \quad \textbf{iff} \quad Tr(M, s) \models \phi \\
M \models \phi \quad \textbf{iff} \quad M, s \models \phi \text{ for all } s \in S_0
\end{array}
$$

Subset CTL[DC] In this subset every temporal operator $\mathbf{X}$, $\mathcal{U}$, $\mathbf{G}$, $\mathbf{F}$ is preceded by a path quantifier $\mathbf{A}$, $\mathbf{E}$. If path formulae are restricted to the following syntax, we have the subset CTL[DC]

$$\phi \,\mathcal{U}\, \psi \;\mid\; \mathbf{X}\phi \;\mid\; \mathbf{F}\phi \;\mid\; \mathbf{G}\phi \quad \text{where } \phi, \psi \text{ are state formulae}$$

Subset LTL[DC] In this subset the formula is of the form $\mathbf{A}\alpha$ where the path formula α is free of path quantifiers $\mathbf{A}$, $\mathbf{E}$. If path formulae are restricted to the following syntax, we have the subset LTL[DC]

$$P \mid D \mid \alpha \,\mathcal{U}\, \beta \mid \mathbf{X}\alpha \mid \alpha \wedge \beta \mid \neg\alpha$$

Subsets CTL, CTL and LTL* If a formula of CTL*[DC] does not contain any QDDC formula, then it is called CTL* formula. Similarly, we can define CTL and LTL formulae.

Example 3. The following CTL[DC] formula states that on all nodes of the computation tree which are at even distance from the root, proposition P must be true. Nothing is said about the truth of P on nodes which are at odd distance from the root.

$$\mathbf{AG}((\eta = 2)^* \quad \Rightarrow \quad P)$$

This property cannot be expressed in logic CTL*. Thus our extension increases the expressive power of logic CTL*.

4 Decidability of Model Checking CTL*[DC]

Given a transition system M and CTL*[DC] formula ϕ, we construct a transformed transition system M' and CTL* formula ϕ' such that

$$M \models \phi \quad \textbf{iff} \quad M' \models \phi' \tag{1}$$

Thus, we reduce the model checking of CTL*[DC] to model checking of CTL*. In the rest of this section, we will define this transformation and prove its correctness.

Let the transition system $M = (S, R, L, S_0)$ and the CTL*[DC] formula be $\phi(D_1, \ldots, D_n)$, where $D_1, \ldots, D_n$ are the syntactically distinct $QDDC$ formulae occurring within ϕ. We construct the transformed transition system M' as follows.

Let $A(D_i)$ be the automaton recognising models of D_i as in Theorem 1. Such an automaton is called *synchronous observer* for D_i. We assume that $A(D_i)$ is in total and deterministic form. By this we mean that from any state q and for any valuation $v \in VAL(pvar(D))$ there is a unique transition leading to the state given by a total function $\delta_i(q, v)$.

We define the synchronous product of M with the list of automata $A(D_i)$. Since each $A(D_i)$ is a finite-state acceptor and M is a Moore machine, we define the product such that we get a Moore machine. Let $A(D_i) = (Q_i, \delta_i, q_i^0, F_i)$. If M starts in state $s \in S_0$, the observers $A(D_i)$ observe this state and go to state $q_s^i = \delta_i(q_i^0, L(s))$ respectively. Also, if M moves from state $s \to s'$, each $A(D_i)$ moves from state $q_i \to \delta_i(q_i, L(s'))$. The observable propositions of the resulting system are valuations over $Pvar \cup \{End_i \mid 1 \leq i \leq n\}$. Proposition End_i holds when automaton $A(D_i)$ is in its final state. Thus, End_i holds precisely when the

behaviour from start up to the current node satisfies the formula D_i. Formally, let

$$M' = (S', R', L', S'_0)$$

where

$$\begin{array}{l} S' \stackrel{\text{def}}{=} S \times Q_1 \times \ldots \times Q_n \\ S'_0 \stackrel{\text{def}}{=} \{\langle s, q_1^s, q_2^s, \ldots, q_n^s\rangle \mid s \in S_0 \text{ and } q_i^s = \delta_i(q_i^0, L(s))\} \\ R' \stackrel{\text{def}}{=} \{(\langle s, q_1, q_2, \ldots, q_n\rangle, \langle s', q'_1, q'_2, \ldots, q'_n\rangle) \mid R(s,s') \wedge q'_i = \delta_i(q_i, L(s'))\} \\ L'(\langle s, q_1, \ldots, q_n\rangle) \stackrel{\text{def}}{=} L(s) \cup \{END_i \mapsto (q_i \in F_i)\} \end{array}$$

The transformed formula $\phi(End_1, \ldots, End_n)$ is obtained by replacing each occurrence of QDDC sub-formula D_i by a proposition End_i which witnesses the truth of D_i.

Theorem 2. *Let* $\hat{s} = \langle s, q_1^s, \ldots, q_n^s\rangle$ *where* $q_i^s = \delta_i(q_i^0, L(s))$. *Then,*

$$M, s \models \phi(D_1, \ldots, D_n) \quad \textbf{iff} \quad M', \hat{s} \models \phi(End_1, \ldots, End_n)$$

Proof Outline Consider the computation tree $T = Tr(M, s)$ in M and corresponding computation tree $T' = Tr(M', \hat{s})$ in M'. There is a bijection $\pi : T \to T'$ between the nodes of T and T' as follows. For every node $k \in Tr(M, s)$ with state label $St(T, k) = s_k$, we have a node $\pi(k)$ with label
$(s_k, \overline{\delta_1}(q_1^0, L(hist(k))), \ldots, \overline{\delta_n}(q_1^0, L(hist(k))))$.
(Here, we have extended the transition function δ_i over VAL to $\overline{\delta_i}$ over VAL^+).
From the above bijection, it is easy to prove that,
$T, k \models P$ **iff** $T', \pi(k) \models P$.
The central property of T' is that
$T, k \models D_i$ **iff** $T', \pi(k) \models End_i$.
From these, by structural induction on ϕ, we can prove that
$T, k \models \phi(D_1, \ldots, D_n)$ **iff** $T', \pi(k) \models \phi(End_1, \ldots, End_n)$. □

Corollary 2. $M \models \phi(D_1, \ldots, D_n)$ **iff** $M' \models \phi(End_1, \ldots, End_n)$ □

Note that, if M is finite state then M' is also a finite state Kripke structure. Moreover, $\phi(End_1, \ldots, End_n)$ is a pure CTL* formula which can be model checked as follows.

Theorem 3 (Emerson and Lei [8]). *For a finite-state Kripke-structure* M' *and CTL* formula* ϕ', *there exists an algorithm to decide whether* $M' \models \phi'$.

Corollary 3. $M \models \phi(D_1, \ldots, D_n)$ *is decidable if* M *is finite-state.* □

5 CTLDC: A Tool for Model Checking CTL[DC]

We have implemented the reduction outlined in Corollary 2 into a tool called $CTLDC$. The tool is constructed on top of QDDC validity checker DCVALID [21,22]. The tool allows CTL[DC] specifications of SMV, Verilog and Esterel designs to be model checked in conjunction with verifiers SMV[18], VIS[2] and Xeve [1], respectively. A separate report gives the details of usage and working of this tool [23].

Given an SMV module M and a formula $\phi(D_1, \ldots, D_n) \in CTL[DC]$, our tool $CTLDC$ gives the transformed SMV modules corresponding to M' of Theorem 2 and also the formula $\phi(End_1, \ldots, End_n)$. We can then use SMV [18] to model check whether $M' \models \phi(End_1, \ldots, End_n)$. The tool works in a similar fashion for Verilog and Esterel designs. The reader may refer to [23] for details of these transformations.

5.1 An Example: Vending Machine

A vending machine accepts $5p$ and $10p$ coins. A chocolate costs $15p$. We model the working of such a machine by the following SMV module.

```
MODULE vend
VAR
  bal:{0,5,10,15};
  event: {p5,p10,choc, null};
INIT
  bal = 0 & !(event=choc)
TRANS
  next(bal) = case
                  bal <= 10 & event=p5          : bal+5 ;
                  bal <=5   & event=p10         : bal+10 ;
                  bal = 15 & event=choc         : 0  ;
                  event=null : bal;
              esac
```

The following QDDC formula holds for all behaviour fragments satisfying the condition that 15p worth of coins have been deposited and no chocolate has been obtained.

$$fifteenp \stackrel{\text{def}}{=} ([\![event \neq choc]\!] \ \wedge \\ (\Sigma(event = 5p) = 3 \ \vee \ (\Sigma(event = 5p) = 1 \wedge \Sigma(event = 10p) = 1)))$$

Then, a possible extension of any behaviour ending with $fifteenp$ is that a chocolate is obtained next.

$$\mathbf{AG}\ (true \frown fifteenp \quad \Rightarrow \quad \mathbf{EX}(event = choc))$$

Moreover, the only possible extensions are that a null event can occur or a chocolate can be obtained.

$$\mathbf{AG}\ (true \frown fifteenp \quad \Rightarrow \quad \mathbf{AX}(event = choc \vee event = null))$$

We consider the vending machine behaviours under a fairness condition which states that infinitely often non-null events are performed, i.e. $\mathbf{GF}(event \neq null)$ holds. The following specification holds for all fair behaviours of vending machine. Here the path quantifier $\mathbf{A}^f$ ranges over all fair behaviours.

$$\mathbf{A}^f\mathbf{G}\mathbf{A}^f\mathbf{F}\ ((fifteenp \frown unit \frown \lceil event = choc \rceil^0)^*)$$

The above three properties were checked using the tool CTLDC. In checking the last property, we made use of the fair CTL model checking abilities of SMV.

Synchronous Bus Arbiter In a more substantial verification using CTLDC, we checked some properties of the historic synchronous bus arbiter as modelled in SMV by McMillan [18]. A synchronous bus arbiter with n cells has request lines req_i and acknowledgement lines ack_i for $1 \leq i \leq n$. At any clock cycle a subset of the request lines are high. It is the task of the arbiter to set at most one of the corresponding acknowledgement lines high. Preferably, the arbiter should be fair to all requests. We refer the readers to McMillan's book [18] (Section 3.4.1) for a detailed description of a specific synchronous arbiter circuit. Here, we are mainly interested in its properties.

The following property states that if req_i is held high for any interval of m cycles then there must be an ack_i during such an interval.

$$\mathbf{AG}\ \Box((\lceil\lceil req_i \rceil\rceil \wedge (\eta = m) \quad \Rightarrow \quad true \frown \lceil ack_i \rceil^0 \frown ext)$$

For an $n = 5$ cell arbiter, we found using CTLDC that the property holds for the first cell for $m = n$. But for all other cells it, does not hold if $m < 2n$. For these cells, the property does holds for $m = 2n$. Hence we concluded that the first cell is guaranteed access to bus if its request is held high for n cycles whereas for all other cells, the request must be held high for $2n$ cycles to guarantee access.

The following property asserts that the arbiter will not service a request req_j first if an earlier request req_i is still pending (the so called "first come first serve" policy (see [27])).

Let $fifo(i, j)$ be defined as

$$\mathbf{AG}\ \Box\neg(\lceil \neg req_j \rceil^0 \frown \lceil\lceil req_i \wedge \neg ack_i \rceil\rceil => \neg\Diamond\lceil ack_j \rceil^0)$$

Surprisingly, McMillan's bus arbiter with 5-cells satisfies $fifo(i, j)$ for the following pairs and for no other pairs. This was determined experimentally using CTLDC.

$$(1,2), (1,3), (1,4), (1,5), (2,3), (3,4), (4,5)$$

A much more comprehensive analysis of the performance of McMillan's arbiter circuit, and its variants, can be found on the DCVALID web page [21].

6 Conclusions

In this paper, we have proposed an extension of the logic CTL* to CTL*[DC]. This extension allows specification of past-time properties in CTL* using formulae of Quantified Discrete-time Duration Calculus (QDDC). In our opinion, this

simple extension considerably facilitates formalisation of complex requirements. CTL*[DC] is especially useful for expressing past-time requirements and quantitative timing constraints. In a separate note, we give many such examples [24]. The properties in previous section are also illustrative. Formally, the expressive power of the logic CTL* is increased as shown by Example 3. CTL*[DC] can also be considered as a significant extension of QDDC which allows liveness and branching properties to be stated.

We have shown a reduction of CTL*[DC] model checking problem to CTL* model checking problem. The reduction relies upon the automata theoretic decision procedure for QDDC. We believe that this approach is practically relevant as a number of tools exist for CTL* and its subsets, CTL and LTL. We have implementated this reduction into a tool called CTLDC which permits model checking CTL[DC] specifications of finite-state transition systems.

The tool CTLDC can model check SMV, Verilog and Esterel designs by reducing the model checking to a form which can be checked by SMV, VIS and Xeve tools respectively. *In this sense, CTLDC is not a new model checker. It enhances the functionality of SMV [18], VIS [2] and Xeve [1] by adding ability to model check a much more richer logic CTL[DC]. It enables complex properties involving past and quantitative timing to be checked using existing checkers.* A separate report gives details of implementation [23]. Currently, CTLDC is one of the very few available tool for model checking CTL with past and timing. In context of Duration Calculi, CTLDC is the only tool allowing model checking. (It should be noted that our original tool DCVALID [21] only checked for the validity of QDDC formulae.)

The symbolic model checking algorithm for CTL has been extended to fair CTL model checking [18]. It is easy to see that our reduction of CTL*[DC] model checking to CTL* model checking by transforming the transition system M to M' (Theorem 2) preserves fair paths. Hence, our reduction also gives a method for CTL[DC] model checking under fairness constraints by reduction to Fair CTL.

An important aspect of model checking is error trace generation. In our reduction, an error trace of the transformed model M', in fact, gives an error trace for the original model M if we disregard (project out) the extra variables which have been added by the transformation. Hence, existing facilities of counter example generation in reduced model can be used for CTL[DC].

Our approach of combining QDDC with CTL* can equally be used with any other logic, say X, which specifies properties of finite state sequences. Moreover, if the logic permits an automata theoretic decision procedure, this can be used to reduce the model checking problem for CTL*[X] to CTL* by using exactly the same transformation proposed here. One could consider a form of LTL over finite sequence, or monadic logic over finite words which both have automata theoretic decision procedures. Or one could use a form of regular expressions. Hence, the approach presented here is quite generic. However, the expressive power (in a pragmatic sense) and facilities for quantitative timing constraint specifications which are found in QDDC may not be so easily available in all such logics.

The issue of complexity of CTL*[DC] merits discussion and further investigation. As stated in Section 2, even the subset QDDC of CTL*[DC] has a non-elementary lower bound on the complexity of validity checking [22]. The same lower bound carries over to model checking of CTL*[DC] formulae. 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, we have been able to check many formulae which are 5-6 pages long with our tool (see Pandya [21,22] for substantial examples and performance measurements). However, we have also encountered a few pathological formulae leading to state space explosion.

It has been long recognised [17] that availability of past time modalities in temporal logics can considerably facilitate formulation of complex properties. There have been several formulations extending LTL and CTL with past. Their model checking problem has also been investigated [14,15]. Extensions of CTL such as RTCTL [9] allow quantitative timing properties to be expressed and to be model checked using tools such as NuSMV [6]. A precise comparison of the formal expressive power of our logics CTL[DC] and CTL*[DC] with these logics is currently under investigation. We conjecture that CTL[DC] is strictly more expressive than the logic CTL_{lp} proposed by Kupferman and Pnueli [14].

References

1. A. Bouali, XEVE: An Esterel Verification Environment, *Proc. Computer Aided Verification, CAV'98*, Lecture Notes in Computer Science, Springer-Verlag, 1998.
2. R.K. Bryton, G.D. Hatchtel *et al*, VIS: A system for verification and synthesis, in *Proc. Computer Aided Verification, CAV'96*, Lecture Notes in Computer Science 1102, Springer-Verlag, 1996.
3. J.R. Buchi, Weak second-order arithmetic and finite automata, *Z. Math. Logik Grundl. Math.* **6**, 1960.
4. E.M. Clarke and E.A. Emerson, Synthesis of synchronisation skeletons for branching time temporal logic, *Proc. Logics of Programs*, Lecture Notes in Computer Science 131, Springer-Verlag, 1981.
5. E.M. Clarke, E.A. Emerson and A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, *ACM Transactions on Programming Languages and Systems*, **8**(2), 1986.
6. A. Cimatti, E.M. Clarke, F. Giunchiglia and M. Roveri, NuSMV: A Reimplementation of SMV, *Proc. International Workshop on Software Tools for Technology Transfer (STTT-98)*, BRICS Notes Series, NS-98-4, 1998.
7. C.C. Elgot, Decision problems of finite automata design and related arithmetics, *Trans. Amer. Math. Soc.* **98**, 1961.
8. E.A. Emerson and C-L. Lei, Modalities for Model checking: Branching time strikes back, *12th Symposium on Principles of Programming Languages*, New Orleans, La., (January) 1985.
9. E.A. Emerson, A.K. Mok, A.P. Sistla and J. Srinivasan, Quantitative temporal reasoning, *Proc. Computer Aided Verification, CAV'90*, Lecture Notes in Computer Science 531, Springer-Verlag, 1991.
10. M.R. Hansen and Zhou Chaochen, Duration Calculus: Logical Foundations, *Journal of Formal Aspects of Computing* **9**, 1997.

11. J.G. Henriksen, J. Jensen, M. Jorgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm, Mona: Monadic Second-Order Logic in Practice, *Proc. Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95*, Lecture Notes in Computer Science 1019, 1996.
12. Y. Kesten, A. Pnueli, and L. Raviv, Algorithmic Verification of Linear Temporal Logic Specifications, in *Proc. 25th International Colloquium on Automata, Languages, and Programming, ICALP'98*, Lecture Notes in Computer Science 1443, Springer-Verlag, 1998.
13. N. Klarlund, A. Møller and M.I. Schwartzbach, MONA Implementation Secrets, to appear in *Proc. Fifth Conference on Implementation and Application of Automata, CIAA 2000*, Springer-Verlag, 2000.
14. O. Kupferman and A. Pnueli, Once and for all, in *proc. 10th IEEE Symposium on Logics in Computer Science, LICS'95*, San Diago, June 1995.
15. F. Laroussinie and P. Schnoebelen, A Hierarchy of Temporal Logics with Past, *Theoretical Computer Science*, 140(1), 1995.
16. O. Lichtenstein and A. Pnueli, Checking that Finite State Concurrent Programs Satisfy Their Linear Specification. In *Prof. 12 ACM Symposium on Principles of Prog. Languages, POPL'85*, January 1985.
17. O. Lichtenstein, A. Pnueli and L. Zuck, The Glory of the Past, *Proc. Logics of Programs*, Lecture Notes in Computer Science 193, Springer-Verlag, 1985.
18. K. McMillan, *Symbolic Model Checking*, Kluwer Academic Publisher, 1993.
19. B. Moszkowski, A Temporal Logic for Multi-Level Reasoning about Hardware, *IEEE Computer*, **18**(2), 1985.
20. P.K. Pandya, Some Extensions to Mean-Value Calculus: Expressiveness and Decidability, *Proc. Computer Science Logic, CSL'95*, Paderborn, Germany, Lecture Notes in Computer Science 1092, Springer-Verlag, 1996.
21. 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)
22. P.K. Pandya, Specifying and Deciding Quantified Discrete-time Duration Calculus Formulae using DCVALID: An Automata Theoretic Approach, Technical Report TCS-00-PKP-1, Tata Institute of Fundamental Research, 2000.
23. P.K. Pandya, Model Checking CTL[DC] properties of SMV, Verilog and Esterel Designs using CTLDC, Technical Report TCS-00-PKP-3, Tata Institute of Fundamental Research, September 2000.
24. P.K. Pandya, Examples of complex properties from VIS literature, Technical Note PKP-TN-2000-02, Computer Science Group, Tata Institute of Fundamental Research, July 2000. Available on Web at http://www.tcs.tifr.res.in/~pandya/dccheck/demo/sample.html.
25. A. Pnueli, The Temporal Logic of Programs, *Proc. 18th IEEE Symposium on Foundations of Computer Science*, 1977.
26. Zhou Chaochen, C.A.R. Hoare and A.P. Ravn, A Calculus of Durations, *Info. Proc. Letters*, **40**(5), 1991.
27. Zhou Chaochen, M.R. Hansen, A.P. Ravn and H. Rischel, Duration Specification for Shared Processors, *Proc. Formal Techniques in Real-Time and Fault-Tolerent Computing, FTRTFT'92*, Lecture Notes in Computer Science 571, Springer-Verlag, 1992.