Model checking the basic modalities of CTL with Description Logic

Shoham Ben-David Richard Trefler Grant Weddell

David R. Cheriton School of Computer Science
University of Waterloo

Abstract. Model checking is a fully automated technique for determining whether the behaviour of a finite-state reactive system satisfies a temporal logic specification. Despite the fact that model checking may require analyzing the entire reachable state space of a protocol under analysis, model checkers are routinely used in the computer industry. To allow for the analysis of large systems, different approaches to model checking have been developed, each approach allowing for a different class of systems to be analyzed. For instance, some model checkers represent program state spaces and transitions explicitly, others express these concepts implicitly. The determination of which flavour best suits a particular model must often be left to experimentation. Description Logic (DL) reasoners are capable of performing subsumption checks on large terminologies. In this paper, we show how to perform explicit state model checking with a DL reasoner. We formulate the check that a reactive system satisfies a temporal specification as a consistency check on a terminology in the DL $\mathcal{ALC}$ and demonstrate our method on an example.

1 Introduction

Model checking [4] (cf. [5]) is a fully automated technique for verifying that the behaviour of a finite-state reactive system satisfies a temporal logic specification. As such, it is extremely useful in verifying important aspects of safety critical reactive systems.

The main challenge in this area, known as the *state explosion problem*, arises because systems may have short textual descriptions encoding exponentially larger state spaces that are analyzed by the model checker. While model checking techniques are widely used in industry [1, 6, 11], methods of attacking the state explosion problem and increasing the applicability of this technique are of substantial interest.

Given a finite state model M (a non-deterministic Kripke structure), a model checker verifies that the behaviours of M satisfy a temporal logic specification φ, typically given in Computation Tree Logic (CTL) [4], i.e. $M \models \varphi$.[1] The following is a typical specification: *it is always the case that at most one of several concurrent processes has access to a shared resource*. Letting c_i indicate that process i has access to the shared resource of interest, the specification is expressed succinctly in CTL as $\mathsf{AG}\neg(c_1 \wedge c_2)$. Here, the basic modality AG is composed of two temporal operators A, representing *all future paths*, and G, representing *all states on a path*. Formulas of the form $\mathsf{AG}(p)$, with p being a Boolean expression, are of special importance. In our experience, over 90% of the formulas written in practice can be converted into $\mathsf{AG}(p)$ form.

To cope with the state explosion problem several different techniques have been proposed. The different approaches work well on large classes of examples but no one technique best

[1] The symbol $\models$ is overloaded. We use it in this paper both in the context of model checking and in the context of DL reasoning.

suits all models. One method describes the model *symbolically* by representing the system under verification by boolean functions. Two main symbolic methods are used to perform model checking. The first, known as *SMV* [10], was introduced by McMillan in 1992. This method is based on Binary Decision Diagrams (BDDs) [3] for representing the state space as well as for performing the model checking procedure. The second is known as Bounded Model Checking [2]. Using this method, the model under verification is translated into a Boolean formula, and a satisfiability solver is applied to it to find a satisfying assignment. Such an assignment, if found, demonstrates a bug in the model.

Other important methods represent states and transitions explicitly. Explicit state methods appear to be more amenable to sophisticated state space reduction techniques. In this paper we show how to use Description Logic reasoning to perform explicit state model checking of the basic modalities of CTL on a synchronous model of computation, with the hope of benefiting from the powerful DL reasoners currently available [7–9]. Let MD be a model description, whose sematics is given by the Kripke structure M_{MD}, and let φ be a specification. We formulate a terminology $\mathcal{T}_{MD,\varphi}$ over the DL dialect $\mathcal{ALC}$, and define a concept C_{MD} such that by checking if $\mathcal{T}_{MD,\varphi} \models C_{MD}$ is consistent, we can determine whether $M_{MD} \models \varphi$.

In the next section we give the necessary background and definitions. Section 3 presents the translation of a model checking problem into a terminology over $\mathcal{ALC}$, and demonstrate it through an example. Section 4 concludes the paper.

2 Background and Definitions

Definition 1 (Description Logic $\mathcal{ALC}$) *Let* NC *and* NR *be sets of atomic concepts* $\{A_1, A_2, \ldots\}$ *and atomic roles* $\{R_1, R_2, \ldots\}$*, respectively. The set of* concepts C *of the description logic* $\mathcal{ALC}$ *is the smallest set including* NC *that satifies the following.*

- *If* $C_1, C_2 \in$ C*, then so are*
  - $\neg C_1$ • $C_1 \sqcap C_2$
- *If* $C \in$ C*, and* $R \in$ NR*, then so is*
  - $\exists R.C$

Additional concepts are defined as syntactic sugaring of those above:
- $\top = A \sqcup \neg A$*, for some* A • $\forall R.C = \neg\exists R.\neg C$
- $C_1 \sqcup C_2 = \neg(\neg C_1 \sqcap \neg C_2)$

An inclusion dependency *is an expression of the form* $C_1 \sqsubseteq C_2$*. A* terminology $\mathcal{T}$ *consists of a finite set of inclusion dependencies.*

The semantics *of expressions is defined with respect to a structure* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$*, where* $\Delta^{\mathcal{I}}$ *is a non-empty set, and* $\cdot^{\mathcal{I}}$ *is a function mapping every concept to a subset of* $\Delta^{\mathcal{I}}$ *and every role to a subset of* $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ *such that the following conditions are satisfied.*
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ • $(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \sqcap C_2^{\mathcal{I}}$
- $\exists R.C = \{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} \text{ s.t. } (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$

A structure satisfies an inclusion dependency $C_1 \sqsubseteq C_2$ *if* $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$*. The* consistency problem for $\mathcal{ALC}$ *asks if* $\mathcal{T} \models C$ *holds; that is, if there exists* $\mathcal{I}$ *such that* $C^{\mathcal{I}}$ *is non-empty and such that* $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ *holds for each* $C_1 \sqsubseteq C_2$ *in* $\mathcal{T}$*.*

Definition 2 (Kripke Structure) *Let* V *be a set of Boolean variables. A* Kripke structure M *over* V *is a four tuple* $M = (S, I, R, L)$ *where*

1. *S is a finite set of states.*
2. *$I \subseteq S$ is the set of initial states.*
3. *$R \subseteq S \times S$ is a transition relation that must be total, that is, for every state $s \in S$ there is a state $s' \in S$ such that $R(s, s')$.*
4. *$L : S \to 2^V$ is a function that labels each state with the set of variables true in that state.*

In practice, the full Kripke structure of a system is not explicitly given. Rather, a model is given as a set of Boolean variables $V = \{v_1, ..., v_n\}$, their initial values and their next-state assignments. The definition we give below is an abstraction of the input language of *SMV* [10].

Definition 3 (Model Description) *Let $V = \{v_1, ..., v_k\}$ be a set of Boolean variables. A* Model Description *over V is a tuple $MD = (I_{MD}, [\langle c_1, c'_1\rangle, ..., \langle c_k, c'_k\rangle])$, where $I_{MD} = \{v_1 = b_1, ..., v_k = b_k\}$, $b_i \in \{0, 1\}$, and c_i, c'_i are Boolean expressions over V.*

The semantics of a model description is a Kripke structure $M_{MD} = (S, I_M, R, L)$, where $S = 2^V$, $L(s) = s$ for $s \in S$, $I_M = \{I_{MD}\}$ and $R = \{(s, s') : \forall 1 \leq i \leq k,\ s \models c_i$ implies $s' \models (v_i = 0)$ and $s \models c'_i \wedge \neg c_i$ implies $s' \models (v_i = 1)\}$.

Intuitively, a pair $\langle c_i, c'_i\rangle$ defines the next-state assignment of variable v_i in terms of the current values of $\{v_1, ..., v_k\}$. That is,

$$\text{next}(v_i) = \begin{cases} 0 & \text{if } c_i \\ 1 & \text{if } c'_i \wedge \neg c_i \\ \{0, 1\} & \text{otherwise} \end{cases}$$

where the assignment $\{0, 1\}$ indicates that for every possible next-state value of variables $v_1, ...v_{i-1}, v_{i+1}, ..., v_n$ there must exist a next-state with $v_i = 1$, and a next-state with $v_i = 0$.

We give the definition of Basic CTL formulas below. Our definition differs from full CTL in that temporal operators cannot be nested. Note though, that most of the formulas written in practice can be converted into $AG(p)$ form, which is included in Basic CTL.

Definition 4 (Basic CTL formulas [4]) – *The formula ($v = 1$) is an atomic CTL formula*
- *If p and q are atomic CTL formulas, then so are*
  - $\neg p$ • $p \wedge q$
- *If p and q are atomic CTL formulas then the following are Basic CTL formulas:*
  - EXp • AXp • $E[pVq]$ • $A[pVq]$
- *If φ is a Basic CTL formula then so is* • $\neg\varphi$.

Additional operators are defined as syntactic sugaring of those above:

• $E[pUq] = \neg A[\neg pV\neg q]$ • $A[pUq] = \neg E[\neg pV\neg q]$ • $AFp = A[true\ Up]$
• $EFp = E[true\ Up]$ • $AGp = \neg EF\neg p$ • $EGp = \neg AF\neg p$

The semantics of a CTL formula is defined with respect to a Kripke structure $M = (S, I, R, L)$ over a set of variables $V = \{v_1, ..., v_k\}$. A path in M is an infinite sequence of states $(s_0, s_1, ...)$ such that each successive pair of states (s_i, s_{i+1}) is an element of R. The notation $M, s \models \varphi$, means that the formula φ is true in state s of the model M.

- $M, s \models (v = 1)$ iff $v \in L(s)$
- $M, s \models p \wedge q$ iff $M, s \models p$ and $M, s \models q$
- $M, s \models \neg\varphi$ iff $M, s \not\models \varphi$

- $M, s_0 \models AXp$ iff for all paths $(s_0, s_1, ...)$, $M, s_1 \models p$
- $M, s_0 \models EXp$ iff for some path $(s_0, s_1, ...)$, $M, s_1 \models p$
- $M, s_0 \models A[pVq]$ iff for all paths $(s_0, s_1, ...)$, either for all $i \geq 0$, $M, s_i \models q$ or there exists $n \geq 0$ such that $M, s_n \models p$ and for all $0 \leq i \leq n, M, s_i \models q$
- $M, s_0 \models E[pVq]$ iff for some path $(s_0, s_1, ...)$, either for all $i \geq 0$, $M, s_i \models q$ or there exists $n \geq 0$ such that $M, s_n \models p$ and for all $0 \leq i \leq n, M, s_i \models q$

We say that a Kripke structure $M = (S, I, R, L)$ satisfies a Basic CTL formula φ ($M \models \varphi$) if for all $s_i \in I$, $M, s_i \models \varphi$.

Definition 5 (Formula type) *Let φ be a Basic CTL formula, expressed in terms of EX, AX, $E[pVq]$ or $A[pVq]$. We say that φ is of Type A if the outermost path quantifier is A, and Type E otherwise. We say that φ is a* negated *formula if its path quantifier is preceded by an odd number of negations.*

3 Model Checking Using Description Logic Reasoning

We give a linear reduction of a model checking problem into a consistency check over $\mathcal{ALC}$. Let $MD = (I, [\langle c_1, c'_1\rangle, ..., \langle c_k, c'_k\rangle])$ be a model description for the model $M_{MD} = (S, I, R, L)$, over $V = \{v_1, ...v_k\}$. Let φ be a Basic CTL formula. We generate a terminology $\mathcal{T}_{MD,\varphi}$, linear in the size of MD and constant in the size of φ, and a concept $Init$, such that by checking if $\mathcal{T}_{MD,\varphi} \models Init$ is consistent, we can determine whether $M_{MD} \models \varphi$.

We construct $\mathcal{T}_{MD,\varphi}$ as the union of three terminologies, $\mathcal{T}_{MD,\varphi} = \mathcal{T}_k \cup \mathcal{T}_{MD}^{type} \cup \mathcal{T}_\varphi$ where $\mathcal{T}_k$ depends only on the number k of variables in V, the terminology $\mathcal{T}_{MD}^{type}$ depends on the model description as well as on the *type* of the formula φ (with *type* being *A* or *E*), and $\mathcal{T}_\varphi$ depends only on the formula φ.

We start by describing the primitive concepts and roles which are used in all of the terminologies, and then provide the construction for each of the three terminologies. We conclude this section with a proposition that relates the consistency of the concept $Init$ with respect to $\mathcal{T}_{MD,\varphi}$ to the satisfaction of φ in the model M_{MD}.

Concepts and Roles We introduce one primitive role R corresponding to the transition relation of the model. For each variable $v_i \in V$ we introduce three primitive concepts: V_i, V_iN and V_iT. The concept V_i corresponds to the variable v_i, where V_i denotes $v_i = 1$ and $\neg V_i$ denotes $v_i = 0$. The concept V_iN corresponds in a similar way to the next-state value of V_i. The concept V_iT is needed to encode an execution step of the model as a sequence through R, as will be explained in the sequel. Finally, one primitive concept C_φ is introduced to assist the encoding of the specification φ. Depending on φ, this concept is not always needed. In total, for a set V with k variables, the terminology T_{MD}^{φ} will consist of $3k + 1$ primitive concepts and one role.

Constructing $\mathcal{T}_k$ For a transition $(s, s')\in R$ in the model M_{MD}, the state s may differ from s' in the values of some or all of the variables $v_1, ..., v_k$. That is, in one transition, many variables may simultaneously change their values. To achieve this with our single role R, we encode every transition of the model M_{MD} as a series of "micro-transitions" through R, each of which determines the next-state value of one variable only. To ensure that every variable

makes a move only on its turn, we use the concepts V_iT. We allow exactly one of $V_1T, ..., V_kT$ to hold on each state, and in the correct order, by introducing the following concept inclusions.

- $V_iT \sqsubseteq (\forall R.V_{i+1}T) \quad \neg V_iT \sqsubseteq (\forall R.\neg V_{i+1}T)$ for $1 \leq i < k$ and
  $V_kT \sqsubseteq (\forall R.V_1T) \quad \neg V_kT \sqsubseteq (\forall R.\neg V_1T)$

The actual model states correspond to where concept V_1T holds, after a cycle where each of the variables has taken a turn. Thus, we have to make sure to propagate V_i, V_iN values appropriately, by introducing the following inclusions for $1 \leq i \leq k$.

- When V_1T holds, V_i should assume its next value that has been stored in V_iN.
  $(V_1T \sqcap V_iN) \sqsubseteq V_i \quad (V_1T \sqcap (\neg V_iN)) \sqsubseteq (\neg V_i)$
- Propagate V_i at every step in the cycle, except the last one.
  $((\neg V_kT) \sqcap V_i) \sqsubseteq (\forall R.V_i) \quad ((\neg V_kT) \sqcap (\neg V_i)) \sqsubseteq (\forall R.(\neg V_i))$
- Propagate V_iN, unless V_iT holds, (in which case a new value is computed).
  $((\neg V_iT) \sqcap V_iN)) \sqsubseteq (\forall R.V_iN) \quad ((\neg V_iT) \sqcap (\neg V_iN)) \sqsubseteq (\forall R.(\neg V_iN))$

In total, for k variables, $\mathcal{T}_k$ will consist of $8k$ concept inclusions, each of them of constant size.

Constructing $\mathcal{T}_{MD}^{type}$ We now translate the model description $MD = (I, [\langle c_1, c_1' \rangle, ..., \langle c_k, c_k' \rangle])$. Let $I = \{v_1 = b_1, ..., v_k = b_k\}, b_i \in \{0, 1\}$. To encode the initial condition of the model, we introduce the concept inclusion

- $Init \sqsubseteq (D_1 \sqcap, ..., \sqcap D_k \sqcap V_1T \sqcap \neg V_2T \sqcap, ..., \sqcap \neg V_kT)$

Where $D_i = V_i$ if $(v_i = 1) \in I$, and $D_i = \neg V_i$ if $(v_i = 0) \in I$. The V_iTs are needed to ensure the initial condition corresponds to a real model state (V_1T), and that only V_1 is allowed to make a move ($\neg V_iT$).
Let the pair $\langle c_i, c_i' \rangle$ describe the next state behavior of the variable v_i. That is,

$$\text{next}(v_i) = \begin{cases} 0 & \text{if } c_i \\ 1 & \text{if } c_i' \wedge \neg c_i \\ \{0, 1\} & \text{otherwise} \end{cases}$$

where c_i, c_i' are Boolean expressions over $v_1, ..., v_k$, and $\{0, 1\}$ is a non-deterministic assignment, allowing v_i to assume both 0 and 1 in the next state. Let C_i be the concept generated by replacing every v_i in c_i with the concept V_i, and $\wedge$ with $\sqcap$. Let C_i' be the concept corresponding to c_i' in the same way. We introduce the following concept inclusions.
$(V_iT \sqcap C_i) \sqsubseteq \exists R.\neg V_iN$
$(V_iT \sqcap \neg C_i \sqcap C_i') \sqsubseteq \exists R.V_iN$
The encoding of the non-deterministic assignment as a concept inclusion, has two flavors, depending on the *type* of the formula φ.

- If φ is of type A, we call the terminology $\mathcal{T}_{MD}^{A}$, and introduce the inclusion:
  $(V_iT \sqcap \neg C_i \sqcap \neg C_i') \sqsubseteq (\exists R.V_iN \sqcap \exists R.\neg V_iN)$.
- If φ is of type E, we call the terminology $\mathcal{T}_{MD}^{E}$, and introduce the inclusion:
  $(V_IT \sqcap \neg C_i \sqcap \neg C_i') \sqsubseteq (\exists R.V_iN \sqcup \exists R.\neg V_iN)$.

In total, $\mathcal{T}_{MD}^{type}$ will consist of one concept, $Init$, of size $2k$, and 3 concept inclusions of constant size.

Constructing $\mathcal{T}_\varphi$ Let φ be the formula to be verified, written in terms of EXp, AXp,$E[pVq]$ or $A[pVq]$. Let ψ be the formula derived from φ by peeling off all negations preceding the outermost path quantifier, as well as the path quantifier itself. (e.g., $\neg E[pVq]$ becomes $[pVq]$). ψ can be one of two formulas only: Xp or $[pVq]$, where p, q are Boolean combinations of variables v_i. Let P, Q be the translation of p, q using the corresponding concepts V_i.

- If $\psi = Xp$, we introduce one concept inclusion to $\mathcal{T}_\varphi$:
  $Init \sqsubseteq \underbrace{\forall R.\forall R...\forall R}_{k}.P$
  We need a sequence of k transitions since one transition in the model M_{MD} is translated into k micro-transition in our terminology.
- If $\psi = [pVq]$ we use the auxiliary concept C_φ introduced for this purpose. Intuitively, $[pVq]$ means "p *releases* q". That is, q must hold along an execution path, unless p appears at one stage, in which case q is *released*, and does not have to hold any longer. We introduce the following concept inclusions to $\mathcal{T}_\varphi$.

$$\begin{aligned} (C_\varphi \sqcap \neg P) &\sqsubseteq \forall R.C_\varphi \\ (C_\varphi \sqcap P) &\sqsubseteq \forall R.\neg C_\varphi \\ \neg C_\varphi &\sqsubseteq \forall R.\neg C_\varphi \\ V_1 T &\sqsubseteq (\neg C_\varphi \sqcup Q) \end{aligned}$$

  The first three concept inclusions above record the behavior of the concept P. C_φ holds in a state if and only if P has never held on the path leading to this state. The forth inclusion guarantees that on all the 'real' execution states (where $V_1 T$ holds), if P has not appeared until the previous state ($\neg C_\varphi$) then Q must hold.

In total, $\mathcal{T}_\varphi$ will consist of at most 4 concept inclusions, with one of them possibly of size k, and the rest of constant size. $\mathcal{T}_{MD,\varphi} = \mathcal{T}_k \cup \mathcal{T}_{MD}^{type} \cup \mathcal{T}_\varphi$ is therefore of size linear in k.

The following proposition relates the consistency of the concept $Init$ with respect to $\mathcal{T}_{MD,\varphi}$ to the satisfaction of φ in the model M_{MD}. Its proof will be given in a full version of the paper.

Proposition 6. *Let MD denote a model description for a model M_{MD}, and let φ be a Basic CTL specification. Then each of the following holds.*

1. *If φ is of type A and* not negated, *then $M_{MD} \models \varphi$ iff $\mathcal{T}_k \cup \mathcal{T}_{MD}^A \cup \mathcal{T}_\varphi \models$ init consistent.*
2. *If φ is of type E and* not negated, *then $M_{MD} \models \varphi$ iff $\mathcal{T}_k \cup \mathcal{T}_{MD}^E \cup \mathcal{T}_\varphi \models$init consistent.*
3. *If φ is of type A and* negated, *then $M_{MD} \models \varphi$ iff $\mathcal{T}_k \cup \mathcal{T}_{MD}^A \cup \mathcal{T}_\varphi \not\models$init consistent.*
4. *If φ is of type E and* negated, *then $M_{MD} \models \varphi$ iff $\mathcal{T}_k \cup \mathcal{T}_{MD}^E \cup \mathcal{T}_\varphi \not\models$init consistent.*

3.1 An Example

We illustrate our method by an example. Consider the model description

$$MD = (I, [\langle v_1 \wedge v_2, v_3\rangle, \langle \neg v_2, v_1 \wedge \neg v_1\rangle, \langle \neg v_1, v_1\rangle])$$

over $V = \{v_1, v_2, v_3\}$ with $I = \{v_1 = 0, v_2 = 1, v_3 = 0\}$. Figure 1 draws the states and transitions of the Kripke structure M_{MD} described by MD. Let the formula to be verified be

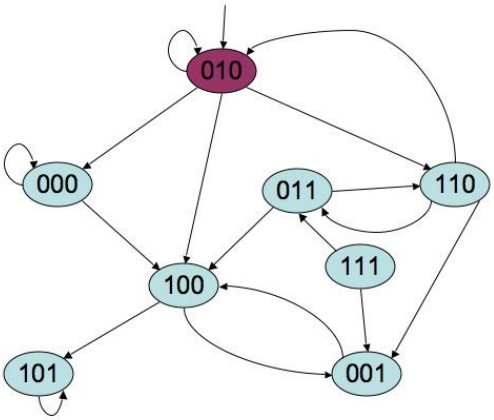

Fig. 1. A Kripke structure for MD

$\varphi = AG(\neg v_1 \vee \neg v_2 \vee \neg v_3)$. Note that $M_{MD} \models \varphi$, as can be seen in Figure 1, since the state $(1,1,1)$ can never be reached from the initial state. The terminology T^{φ}_{MD} derived from MD and φ will use the primitive concepts $\{V_1, V_2, V_3, V_1N, V_2N, V_3N, V_1T, V_2T, V_3T\}$ and the primitive role R. By the construction given in section 3, the set of concept inclusions will be the following.

The Construction of $\mathcal{T}_k$

- Concept inclusions ensuring control over the turn to make a move
  $V_1T \sqsubseteq (\forall R.V_2T)$ $\quad$ $\neg V_1T \sqsubseteq (\forall R.\neg V_2T)$
  $V_2T \sqsubseteq (\forall R.V_3T)$ $\quad$ $\neg V_2T \sqsubseteq (\forall R.\neg V_3T)$
  $V_3T \sqsubseteq (\forall R.V_1T)$ $\quad$ $\neg V_3T \sqsubseteq (\forall R.\neg V_1T)$
- Concept inclusions to propagate the values of V_i
  $((\neg V_3T) \sqcap V_1) \sqsubseteq (\forall R.V_1)$ $\quad$ $((\neg V_3T) \sqcap (\neg V_1)) \sqsubseteq (\forall R.(\neg V_1))$
  $((\neg V_3T) \sqcap V_2) \sqsubseteq (\forall R.V_2)$ $\quad$ $((\neg V_3T) \sqcap (\neg V_2)) \sqsubseteq (\forall R.(\neg V_2))$
  $((\neg V_3T) \sqcap V_3) \sqsubseteq (\forall R.V_3)$ $\quad$ $((\neg V_3T) \sqcap (\neg V_3)) \sqsubseteq (\forall R.(\neg V_3))$
- Concept inclusions to propagate the values of V_iN
  $((\neg V_1T) \sqcap V_1N) \sqsubseteq (\forall R.V_1N)$ $\quad$ $((\neg V_1T) \sqcap (\neg V_1N)) \sqsubseteq (\forall R.(\neg V_1N))$
  $((\neg V_2T) \sqcap V_2N) \sqsubseteq (\forall R.V_2N)$ $\quad$ $((\neg V_2T) \sqcap (\neg V_2N)) \sqsubseteq (\forall R.(\neg V_2N))$
  $((\neg V_3T) \sqcap V_3N) \sqsubseteq (\forall R.V_3N)$ $\quad$ $((\neg V_3T) \sqcap (\neg V_3N)) \sqsubseteq (\forall R.(\neg V_3N))$
- Concept inclusions to move V_i equal to V_iN whenever V_1T holds
  $(V_1T \sqcap V_1N) \sqsubseteq V_1$ $\quad$ $(V_1T \sqcap (\neg V_1N)) \sqsubseteq (\neg V_1)$
  $(V_1T \sqcap V_2N) \sqsubseteq V_2$ $\quad$ $(V_1T \sqcap (\neg V_2N)) \sqsubseteq (\neg V_2)$
  $(V_1T \sqcap V_3N) \sqsubseteq V_3$ $\quad$ $(V_1T \sqcap (\neg V_3N)) \sqsubseteq (\neg V_3)$

The Construction of $\mathcal{T}^{type}_{MD}$ We need to determine the *type* of φ. Note that $AG(p) = A[false\ Vp]$, thus φ can be written as $\varphi = A[false\ V(\neg v_1 \vee \neg v_2 \vee \neg v_3)]$. The type of φ is then A. We proceed to build $\mathcal{T}^{A}_{MD}$:

- The initial condition
  $INIT \sqsubseteq (\neg V_1 \sqcap V_2 \sqcap \neg V_3 \sqcap V_1T \sqcap \neg V_2T \sqcap \neg V_3T)$

– Computation of V_1
  $(V_1T \sqcap V_1 \sqcap V_2) \sqsubseteq (\exists R.(\neg V_1 N))$
  $(V_1T \sqcap (\neg(V_1 \sqcap V_2)) \sqcap V_3) \sqsubseteq (\exists R.V_1 N)$
  $(V_1T \sqcap (\neg(V_1 \sqcap V_2)) \sqcap (\neg V_3)) \sqsubseteq ((\exists R.V_1 N) \sqcap (\exists R.(\neg V_1 N)))$
– Computation of V_2
  $(V_2T \sqcap V_2) \sqsubseteq ((\exists R.V_2 N) \sqcap (\exists R.(\neg V_2 N)))$
  $(V_2T \sqcap (\neg V_2)) \sqsubseteq (\exists R.(\neg V_2 N))$
– Computation of V_3
  $(V_3T \sqcap V_1) \sqsubseteq (\exists R.V_3 N)$ $(V_3T \sqcap (\neg V_1)) \sqsubseteq (\exists R.(\neg V_3 N))$

The Construction of $\mathcal{T}_\varphi$ Since $\varphi = A[false\ V(\neg v_1 \vee \neg v_2 \vee \neg v_3)]$, then according to the translation in section 3, we need to introduce a concept C_φ, in order to record the behavior of $\bot$ (the translation of *false*). However, the behavior of $\bot$ cannot change along the execution path, and we get that $\neg C_\varphi$ is always false. Thus we can omit the concept C_φ altogether, and add only the concept inclusion: $V_1T \sqsubseteq (\neg V_1 \sqcup \neg V_2 \sqcup \neg V_3)$.

By Proposition 6 we get that $M_{MD} \models \varphi$ if and only if $\mathcal{T}_k \cup \mathcal{T}^A_{MD} \cup \mathcal{T}_\varphi \models$ *Init* consistent.

4 Summary

In this paper we have shown that model checking the basic modalities of CTL can be performed using DL reasoning. First experiments using the DL reasoner "Racer" [7] were conducted, with reassuring results. In the future, we plan to test our method on real models from the hardware industry. Several other related directions seem interesting for further investigation. Firstly, it seems possible, using additional calls to "Racer", to handle full CTL model checking. Secondly, we are interested in implementing symbolic model checking algorithms in $\mathcal{ALC}$ and applying these different model checking approaches to available test beds.

Acknowledgements

The authors gratefully acknowledge the support for this result provided by Nortel networks Ltd. and by NSERC Canada.

References

1. S. Ben-David, C. Eisner, D. Geist, and Y. Wolfsthal. Model checking at IBM. *Formal Methods in System Design*, 22(2):101–108, 2003.
2. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without bdds. In *TACAS'99*, 1999.
3. R. Bryant. Graph-based algorithms for boolean function manipulation. In *In IEEE Transactions on Computers*, volume c-35 no. 8.
4. E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logics of Programs*, LNCS 131, pages 52–71. Springer-Verlag, 1981.
5. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 2000.
6. F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Y. Vardi. Benefits of bounded model checking at an industrial setting. In *CAV'01*, july 2001.
7. V. Haarslev and R. Moller. Racer system description. In *International Joint Conference on Automated Reasoning (IJCAR'2001)*, volume 2083.
8. I. Horrocks. The FaCT system. pages 307–312, 1998.
9. I. Horrocks and U. Sattler. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. *Artificial Intelligence*, 160(1–2):79–104, Dec. 2004.
10. K. McMillan. Symbolic model checking, 1993.
11. K. Yorav, S. Katz, and R. Kiper. Reproducing synchronization bugs with model checking. In *CHARME*, 2001.