Università degli studi di Udine
Laurea Magistrale: Informatica
Lectures for April/May 2014
La verifica del software: temporal logic
Lecture 06 CTL* Model Checking

Guest lecturer: Mark Reynolds,
The University of Western Australia

May 14, 2014

Lecture 06

- CTL* model checking

CTL* Model Checking:

Not done in practice much as is quite computationally complex and also CTL* is not as widely known as either LTL or CTL.

We look at an approach from first principles. It does not seem to have been published anywhere, although the general idea is used as part of a tableau approach to CTL* satisfiability checking in [Rey11].

The algorithm can be used to do LTL and CTL model checking as well. It is a reasonable algorithm for LTL but can be simplified if used for CTL.

CTL* Syntax:

Set $\mathcal{L}$ of propositional atoms.

If $p \in \mathcal{L}$ then p is a wff.

If α and β are wff then so are $\neg\alpha$, $\alpha \wedge \beta$, $X\alpha$, $\alpha U \beta$ and $A\alpha$.

Read as: not, and(conjunction), tomorrow (or next), until and all paths.

CTL* Semantics:

Write $M, \sigma \models \alpha$ iff the formula α is true of the fullpath σ in the structure $M = (S, R, g)$ defined recursively by:

$$
\begin{array}{lll}
M, \sigma \models p & \text{iff} & p \in g(\sigma_0), \text{ for } p \in \mathcal{L} \\
M, \sigma \models \neg\alpha & \text{iff} & M, \sigma \not\models \alpha \\
M, \sigma \models \alpha \wedge \beta & \text{iff} & M, \sigma \models \alpha \text{ and } M, \sigma \models \beta \\
M, \sigma \models X\alpha & \text{iff} & M, \sigma_{\geq 1} \models \alpha \\
M, \sigma \models \alpha U \beta & \text{iff} & \text{there is some } i \geq 0 \text{ such that } M, \sigma_{\geq i} \models \beta \\
 & & \text{and for each } j, \text{ if } 0 \leq j < i \text{ then } M, \sigma_{\geq j} \models \alpha \\
M, \sigma \models A\alpha & \text{iff} & \text{for all fullpaths } \tau \text{ with } \tau_0 = \sigma_0,\ M, \tau \models \alpha
\end{array}
$$

Abbreviations:

Classical and linear temporal abbreviations $\top$, $\bot$, $\vee$, $\rightarrow$, $\leftrightarrow$, F, G.

Also,
$E\alpha \equiv \neg A \neg \alpha$ meaning that there is a path on which α holds.

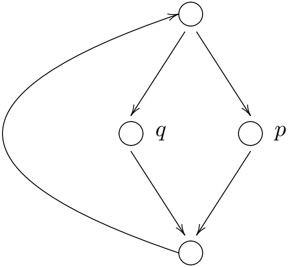
CTL* examples:

$$XEXp \rightarrow EXXp$$
$$AG(p \rightarrow Ap)$$
$$AXFp \rightarrow XAFp$$
$$E(pU(E(pUq))) \rightarrow E(pUq)$$
$$AG(p \rightarrow q) \rightarrow (EFp \rightarrow EFq)$$
$$AG(p \rightarrow EXFp) \rightarrow (p \rightarrow EGFp)$$

Model of a System:

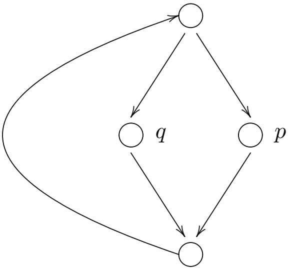
We suppose that we can model the system as a finite state machine: a set of states and a set of allowed transitions from some states to other states.

In order to allow abstraction of observable basic, or atomic properties, we also suppose that there are a set of atomic properties, and that some properties are true at some states.



Example of Model Checking:

Does the following formula hold in the structure below?
$AG(GFEXp \rightarrow GFp)$



Model Checking Task

We want to enter the description of the model as finite state machine with labelling of atoms true at each state: i.e. as a structure (S, R, g).

Then enter the CTL* formula ϕ representing the property to check.

The algorithm should eventually halt and either say "yes" or "no".

We want to find out whether the structure is a model of the formula.
But what exactly does that mean?

Model Checking Task

We want to find out whether there is a fullpath making the formula true or do we want to know if all fullpaths make the formula true? A fullpath? Or all fullpaths? Or ones starting at some initial state?

In fact these are all similar inter-translatable problems. For the CTL* algorithm that we meet first, it does not matter. The algorithm produces a data structure from which all such questions can be answered.

We will find out, for every state, whether there is a fullpath starting there which makes ϕ true and whether there is one starting there that makes ϕ false.

Induction on formulas:

We will try to determine truth (or otherwise) of all the subformulas of ϕ and their negations along all fullpaths in the structure.

$\mathbf{C}(\phi) = \{\psi, \neg\psi | \psi \leq \phi\}$ where $\psi \leq \phi$ means that ψ is a subformula of ϕ.

The algorithm proceeds from simpler formulas to more complicated ones so we order the subformulas of ϕ as $\alpha_0, \alpha_1, \alpha_2, ..., \alpha_N = \phi$ in some way such that if $\alpha_i \leq \alpha_j$ then $i \leq j$.

A_k:

So we work out truth at all fullpaths for α_1 then α_2 etc in that order until we have done so for $\phi = \alpha_N$.

Let $A_k = \{\alpha_i, \neg\alpha_i \mid i \leq k\}$. At the kth stage we will know the truth (or otherwise) of each formula in A_k along each fullpath.

Infinite numbers of fullpaths:

But there are an infinite number of fullpaths in general. How can we consider them all?

What we do is group them together into *equivalence classes* at each stage.

Two fullpaths σ and σ' are equivalent at the kth stage iff they start at the same state ($\sigma_0 = \sigma'_0$) and for every $\beta \in A_k$, $(S, R, g), \sigma \models \beta$ iff $(S, R, g), \sigma' \models \beta$.

An equivalence class is a maximal set of all fullpaths such that each pair from the set are equivalent.

Formula sets:

We work out at each stage the equivalence classes at that stage.

Actually we just work out which classes are non-empty and for each non-empty class, we work out the set of formulas from A_k that are true on the fullpaths in that class.

At each stage k, we make a record $S_k = (\gamma_k, \delta_k)$ of the classes we have found and some relationships between them.

Formula sets:

γ_k is a map from S to subsets of A_k. If $a \subseteq A_k$ and $a \in \gamma_k(s)$ then we have determined that there is a non-empty equivalence class of fullpaths which start at s and which make exactly the formulas in a true (out of the formulas in A_k).

So each pair (s, a) such that $a \in \gamma_k(s)$ will determine an equivalence class being the equivalence class of fullpaths that start at s and make exactly the formulas in a true.

δ:

To help us with the induction we also store a record of which equivalence classes follow on from which other ones.
δ_k will be a set of pairs $((s, a), (s', a'))$ where $s, s' \in S$, $a \in \gamma_k(s)$ and $a' \in \gamma_k(s')$.

We will put $((s, a), (s', a')) \in \delta_k$ iff (we have determined that) there is at least one fullpath σ such that σ is in the equivalence class determined by (s, a) and $\sigma_{\geq 1}$ is in the equivalence class determined by (s', a').

Base Case:

Initial record is $S_{-1} = (\gamma, \delta)$ where γ and δ are as follows.

For each $t \in S$, $\gamma(t) = \{\emptyset\}$. (Assume $A_{-1} = \emptyset$.)
For each $t, t' \in S$, if t' is a successor of t, i.e. $(t, t') \in R$ then we put $(t, \emptyset)\delta(t', \emptyset)$.

At the start all fullpaths through s are in the class determined by $(s, \emptyset)$.

Inductive Step:

Now we proceed by induction on $k \geq 0$.
So suppose that $S_{k-1} = (\gamma, \delta)$ has been built and we want to build $S_k = (\gamma', \delta')$ by considering α_k.

There are several cases depending on the form of α_k.

Case of p:

$\alpha_k = p \in \mathcal{L}$.
For each node $t \in S$, for each $a \in \gamma(t)$, we define a new set $u(t, a) \subseteq A_k$.
Just put $u(t, a) = a \cup \{p\}$ iff $p \in g(t)$.
Otherwise put $u(t, a) = a \cup \{\neg p\}$.
Let $\gamma'(t) = \{u(t, a) \mid a \in \gamma(t)\}$.
As for δ', δ' is (practically) unchanged from δ. To be precise, if δ related a pair of pairs then δ' relates the corresponding pair of updated pairs. That is,

$$((t, u(t, a)), (t', u(t', a'))) \in \delta' \text{ iff } ((t, a), (t', a')) \in \delta$$

Case of $\neg\alpha$:

$\alpha_k = \neg\alpha$. For each node $t \in S$, for each $a \in \gamma(t)$, we only need to do something if $\alpha \in a$. In that case put $a' = a \cup \{\neg\neg\alpha\}$. Otherwise we do not need to do anything as $\neg\alpha$ will already be in a: put $a' = a$. Let $\gamma'(t) = \{a' \mid a \in \gamma(t)\}$. δ is unchanged (i.e. as per atomic case).

Case of $\alpha \wedge \beta$:

$\alpha_k = \alpha \wedge \beta$. For each node $t \in S$, for each $a \in \gamma(t)$, if $\alpha \in a$ and $\beta \in a$, we put $a' = a \cup \{\alpha \wedge \beta\}$. Otherwise put $a' = a \cup \{\neg(\alpha \wedge \beta)\}$. Let $\gamma'(t) = \{a' \mid a \in \gamma(t)\}$. δ is unchanged.

Case of $X\alpha$:

$\alpha_k = X\alpha$. In this case we may need to split a formula-set into two. Let $a^+ = a \cup \{X\alpha\}$ and $a^- = a \cup \{\neg X\alpha\}$. For each $t \in S$ in some order, for each $a \in \gamma(t)$, we will replace a in $\gamma(t)$ by either a^+ or a^- or both in $\gamma'(t)$ and we will define δ'.

To decide which, consider the δ successors of (t, a) (it will always have some).

Case of $X\alpha$ (continued):

If $(t, a)\delta(t', b)$ and $\alpha \in b$ then we will put a^+ in $\gamma'(t)$ and put $(t, a^+)\delta'(t', b^\pm)$. (Note that by this we mean that we put $(t, a^+)\delta'(t', b^+)$ if $b^+ \in \gamma'(t')$ AND we put $(t, a^+)\delta'(t', b^-)$ if $b^- \in \gamma'(t')$.)

Also, if $(t, a)\delta(t'', c)$ and $\neg\alpha \in c$ then we will put a^- in $\gamma'(t)$ and put $(t, a^-)\delta'(t'', c^\pm)$.

We may need to do both or just one.

Case of $\alpha U\beta$ (overview):

$\alpha_k = \alpha\ U\beta$.

Again we may need to split formula-sets.

In this case we need to work on all the nodes together as δ' will connect up paths of new formula-sets. For each $t \in S$ and each $a \in \gamma(t)$ we will put either a new formula-set $a^+ = a \cup \{\alpha\ U\beta\}$ or a new formula-set $a^- = a \cup \{\neg(\alpha\ U\beta)\}$ or both into $\gamma'(t)$.

That's all we have time for:

To be continued tomorrow.

And that's all for the sixth lecture.

See you tomorrow for more model checking.

Reference:

Mark Reynolds.
A tableau-based decision procedure for CTL*.
Journal of Formal Aspects of Computing, pages 1–41, August 2011.