Hoare Logic and Model Checking

Model Checking
Lecture 11: Model checking for Computation Tree Logic

Dominic Mulligan
Based on previous slides by Alan Mycroft and Mike Gordon

Programming, Logic, and Semantics Group
University of Cambridge

Academic year 2016–2017
Learning outcomes

At the end of this lecture, you should:

- Understand the CTL model checking problem
- Understand the “satisfaction set” of states for CTL formulae
- Know the naïve recursive labelling algorithm for computing satisfaction sets
- Understand CTL model checking is a reachability problem
- Know the computational complexity of CTL model checking
The CTL model checking problem
Suppose $\mathcal{M} = \langle S, S_0, \rightarrow, \mathcal{L} \rangle$ is a CTL model.

Suppose also that $s \in S$ is a state, and $\Phi$ is a CTL state formula.

We want to establish whether $s \models \Phi$ (as efficiently as possible).

Importantly: we want to establish whether $s \models \Phi$ for all $s \in S_0$.

“All possible initial states satisfy $\Phi$”

This is the CTL model checking problem.
In $\mathcal{M}$, define:

$$Sat(\Phi) = \{s \in S \mid s \models \Phi\}$$

The “states that satisfy $\Phi$”

CTL model checking problem can be solved by:

1. Computing $Sat(\Phi)$ set for relevant CTL state formula
2. Checking whether $S_0 \subseteq Sat(\Phi)$

Then: how do we compute $Sat(\Phi)$?
Simple recursive algorithm
Reminder: Existential Normal Form

Recall from last lecture:

- Existential Normal Form formulae have negations “pushed in”
- Only use a subset of modalities
- Theorem: every CTL state formula $\Phi$ has an equivalent ENF formula

In this lecture, we work only with ENF formulae (fewer cases to cover)

To extend our algorithm implementations to full CTL:

- Wrap them in another function accepting a CTL formula,
- Use translation hidden in constructive proof of theorem above,
- Call the algorithm on this translated formula
Suppose $\Phi$ is ENF formula

Take a step back:

- We aim to algorithmically compute $Sat(\Phi)$ in order to check $s \models \Phi$ for $s \in S_0$
- But what is this set?

Need to first characterise $Sat(\Phi)$ to understand whether algorithm correct
Characterising $\text{Sat}(\Phi)$: the ‘easy’ cases

For $\mathcal{M} = \langle S, S_0, \rightarrow, \mathcal{L} \rangle$, we have:

\[
\begin{align*}
\text{Sat}(\top) &= S \\
\text{Sat}(p) &= \{ s \mid p \in \mathcal{L}(s) \} \\
\text{Sat}(\neg \Phi) &= S - \text{Sat}(\Phi) \\
\text{Sat}(\Phi \land \Psi) &= \text{Sat}(\Phi) \cap \text{Sat}(\Psi)
\end{align*}
\]

Here: $S - \text{Sat}(\Phi)$ is relative complement

Note, per setwise reasoning, we have $\text{Sat}(\Phi \lor \Psi) = \text{Sat}(\Phi) \cup \text{Sat}(\Psi)$

Other derived connectives similarly map onto setwise operations
Characterising $Sat(\Phi)$: the $\exists \Box \Phi$ case

For $\mathcal{M} = \langle S, S_0, \rightarrow, \mathcal{L} \rangle$, we have:

$$Sat(\exists \Box \Phi) = \{ s \in S \mid Post(s) \cap Sat(\Phi) \neq \{\}\}$$

Here, $Post(s) = \{ s' \mid s \rightarrow s' \}$
For $\mathcal{M} = \langle S, S_0, \rightarrow, \mathcal{L} \rangle$, we have:

$Sat(\exists(\Phi \text{ UNTIL } \Psi))$ is the smallest $T \subseteq S$, such that:

1. $Sat(\Psi) \subseteq T$,
2. If $s \in Sat(\Phi)$ with $Post(s) \cap T \neq \{\}$ then $s \in T$

Here, “smallest” is interpreted with respect to set inclusion order.
Correctness of characterisation of $Sat(∃(Φ \text{ UNTIL } Ψ))$ (1)

Suppose $T = Sat(∃(Φ \text{ UNTIL } Ψ))$

$∃(Φ \text{ UNTIL } Ψ)$ satisfies an “expansion law”:

$$∃(Φ \text{ UNTIL } Ψ) \equiv Ψ ∨ (Φ \land ∃⃝∃(Φ \text{ UNTIL } Ψ))$$

$$T = Sat(∃(Φ \text{ UNTIL } Ψ))$$

$$= Sat(Ψ ∨ (Φ \land ∃⃝∃(Φ \text{ UNTIL } Ψ)))$$

$$= Sat(Ψ) ∪ (Sat(Φ) \cap \{s ∈ S \mid Post(s) \cap Sat(∃(Φ \text{ UNTIL } Ψ)) \neq \{\}\})$$

$$= Sat(Ψ) ∪ (Sat(Φ) \cap \{s ∈ S \mid Post(s) \cap T \neq \{\}\})$$

So:

1. $Sat(Ψ) ⊆ T$
2. $s ∈ Sat(Φ)$ with $Post(s) \cap T \neq \{\}$ implies $s ∈ T$
Correctness of characterisation of $Sat(\exists (\Phi \cup \Psi))$ (2)

Suppose $T$ satisfies:

1. $Sat(\Psi) \subseteq T$,
2. If $s \in Sat(\Phi)$ with $Post(s) \cap T \neq \{\}$ then $s \in T$

Aim to show $Sat(\exists (\Phi \text{ UNTIL } \Psi)) \subseteq T$

Suppose $s \in Sat(\exists (\Phi \text{ UNTIL } \Psi))$

Work by cases on whether $s \in Sat(\Psi)$

One case is easy:

If $s \in Sat(\Psi)$ then $s \in T$ per (1) above
Correctness of characterisation of $Sat(\exists (\Phi \cup \Psi))$ (3)

Otherwise suppose $s \notin Sat(\Psi)$

Note $\pi = s_0, s_1, s_2, \ldots$ exists where $s = \pi[0]$ and $\pi \models \Phi \text{ UNTIL } \Psi$

Let $n > 0$ be such $\pi[n] \models \Psi$ and $\pi[i] \models \Phi$ for $0 \leq i < n$

Then $\pi[n] \in Sat(\Psi)$ and therefore $\pi[n] \in T$ per (1) above

Then $\pi[n - 1] \in Sat(\Phi)$ and $\pi[n - 1] \in T$ since $\pi[n] \in Post(\pi[n - 1]) \cap T$

Then $\pi[n - 2] \in Sat(\Phi)$ and $\pi[n - 2] \in T$ since $\pi[n - 1] \in Post(\pi[n - 2]) \cap T$

... 

Then $\pi[0] \in Sat(\Phi)$ and $\pi[0] \in T$ since $\pi[1] \in Post(\pi[0]) \cap T$

Therefore $s = \pi[0] \in T$, as required
Characterising $\text{Sat}(\Phi)$: the $\exists(\Box \Phi)$ case

For $\mathcal{M} = \langle S, S_0, \rightarrow, \mathcal{L} \rangle$, we have:

$\text{Sat}(\exists(\Box \Phi))$ is the largest $T \subseteq S$, such that:

1. $T \subseteq \text{Sat}(\Phi)$
2. If $s \in T$ then $\text{Post}(s) \cap T \neq \{\}$

Here, “largest” is interpreted with respect to set inclusion order.
Correctness of characterisation of $Sat(∃□Φ)$ (1)

Suppose $T = Sat(∃□Φ)$

$∃□Φ$ also satisfies an “expansion law”:

$$∃□Φ ≡ Φ ∧ ∃□ Φ$$

$$T = Sat(∃□Φ)$$
$$= Sat(Φ ∧ ∃□ Φ)$$
$$= Sat(Φ) ∩ \{s \in S \mid Post(s) ∩ Sat(∃□Φ) \neq \{\}\}$$
$$= Sat(Φ) ∩ \{s \in S \mid Post(s) ∩ T \neq \{\}\}$$

So:

1. $Sat(∃□Φ) ⊆ Sat(Φ)$
2. $s ∈ T$ implies $Post(s) ∩ T \neq \{\}$
Correctness of characterisation of $Sat(\exists \square \Phi)$ (2)

Suppose $T$ satisfies:

1. $T \subseteq Sat(\Phi)$
2. $s \in T$ implies $Post(s) \cap T \neq \emptyset$

Aim to show $T \subseteq Sat(\exists \square \Phi)$

Suppose $s \in T$ (for $T$ non-empty), define $\pi$:

$\pi[0] = s \in T$

$\pi[1]$ is some state $s_1 \in Post(s_0) \cap T$, which exists as $s_0 \in T$ per (2)

$\pi[2]$ is some state $s_2 \in Post(s_1) \cap T$, which exists as $s_1 \in T$ per (2)

...

Hence $\pi[i] \in T \subseteq Sat(\Phi)$ for all $i \geq 0$ and $\pi \models \square \Phi$ and $s \in Sat(\exists \square \Phi)$

As this applies to any $s \in T$, we have $T \subseteq Sat(\exists \square \Phi)$ as required
Recursive labelling algorithm

Pseudocode:

function \text{SAT}(\Phi):

\begin{align*}
\text{switch } & \Phi \text{ do:} \\
\text{case } & \top: \text{ return } S \\
\text{case } & p: \text{ return } \{ s \in S \mid p \in \mathcal{L}(s) \} \\
\text{case } & \neg \Psi: \text{ return } S - \text{Sat}(\Psi) \\
\text{case } & \Psi \land \Xi: \text{ return } \text{Sat}(\Psi) \cap \text{Sat}(\Xi) \\
\text{case } & \exists \bigcirc \Psi: \text{ return } \{ s \in S \mid \text{Post}(s) \cap \text{Sat}(\Psi) \neq \{\} \} \\
\text{case } & \exists (\Psi \text{ UNTIL } \Xi): \text{ return } \text{SatExistsUntil}(\Psi, \Xi) \\
\text{case } & \exists (\square \Psi): \text{ return } \text{SatExistsSquare}(\Psi)
\end{align*}

end function
Subprocedure SatExistsUntil

Pseudocode for SatExistsUntil:

function $\text{SatExistsUntil}(\Phi, \Psi)$:

$T \leftarrow \text{Sat}(\Psi)$

while $\{s \in \text{Sat}(\Phi) - T \mid P ost(s) \cap T \neq \{\}\} \neq \{\}$ do:

$s \leftarrow$ some state from $\{s \in \text{Sat}(\Phi) - T \mid P ost(s) \cap T \neq \{\}\}$

$T \leftarrow T \cup \{s\}$

end while

return $T$

end function
Pseudocode for SatExistsSquare:

function \texttt{SatExistsSquare}(\Phi):

\begin{align*}
T & \leftarrow \texttt{Sat}(\Phi) \\
\text{while } \{s \in T \mid \text{Post}(s) \cap T = \{\}\} \neq \{\}\text{ do} \\
& \quad s \leftarrow \text{some state from } \{s \in T \mid \text{Post}(s) \cap T = \{\}\} \\
& \quad T \leftarrow T - \{s\} \\
\text{end while} \\
\text{return } T
\end{align*}
Recall $Sat(\exists (\Phi \text{ UNTIL } \Psi))$ is smallest $T \subseteq S$:

$Sat(\Psi) \subseteq T$ \quad s \in Sat(\Phi) \text{ and } Post(s) \cap T \neq \{\} \text{ implies } s \in T$

This suggests an iterative procedure for computing $Sat(\exists (\Phi \text{ UNTIL } \Psi))$:

$T_0 = Sat(\Psi)$

$T_{i+1} = T_i \cup \{s \in Sat(\Phi) \mid Post(s) \cap T_i \neq \{\}\}$

Iterate until fixed point is reached

$T_i$ states can reach $\Psi$-state in at most $i$ steps along $\Phi$-path

SatExistsUntil implements this idea
Recall $\text{Sat}(\exists \Box \Phi)$ is largest $T \subseteq S$:

$$T \subseteq \text{Sat}(\Phi) \quad s \in T \implies \text{Post}(s) \cap T \neq \{\}$$

This suggests an iterative procedure for computing $\text{Sat}(\exists \Box \Phi)$:

$$T_0 = \text{Sat}(\Phi)$$

$$T_{1+i} = T_i \cap \{s \in \text{Sat}(\Phi) \mid \text{Post}(s) \cap T_i \neq \{\}\}$$

Iterate until fixedpoint is reached

$\text{SatExistsSquare}$ implements this idea
CTL model checking as reachability

SatExistsUntil and SatExistsSquare are both “backwards searches”

In both cases:

• We start with an initial “guess”
• Move backwards along → transitions, refining guess
• Until we stop

CTL model checking can therefore be seen as a reachability problem

Correctness of algorithm relies crucially on:

• Finiteness of CTL models
• Fixed-point characterisation of CTL
Above algorithm is naïve

Can improve performance by considering only strongly connected components during SatExistsSquare

Do not consider this here

Complexity of optimised variant of above algorithm is $O(|\Phi| \cdot (V + E))$:

- $V$ is number of states in model
- $E$ is number of transitions in model
- $|\Phi|$ is “size” of formula being checked
Summary

- CTL model checking is a reachability problem
- Can model check CTL formulae by computing Sat-set of ENF equivalent
- Satisfaction-set can be computed recursively using a “labelling algorithm”
- Correctness of algorithm depends on fixed-point characterisation of CTL formulae
- Rely crucially on finite models for termination
- Variant of labelling algorithm is $O(|\Phi| \cdot (V + E))$ complexity