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

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 Φ 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 Φ"

This is the CTL model checking problem

States that satisfy formulae

In $\mathcal{M}$, define:

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

The “states that satisfy Φ”

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 Φ 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

Characterising $Sat(\Phi)$

Suppose Φ 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 $Sat(\Phi)$: the 'easy' cases

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

$$
\begin{aligned}
Sat(\top) &= S \\
Sat(p) &= \{s \mid p \in \mathcal{L}(s)\} \\
Sat(\neg\Phi) &= S - Sat(\Phi) \\
Sat(\Phi \wedge \Psi) &= Sat(\Phi) \cap Sat(\Psi)
\end{aligned}
$$

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

Note, per setwise reasoning, we have $Sat(\Phi \vee \Psi) = Sat(\Phi) \cup Sat(\Psi)$

Other derived connectives similarly map onto setwise operations

Characterising $Sat(\Phi)$: the $\exists\bigcirc\Phi$ case

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

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

Here, $Post(s) = \{s' \mid s \rightarrow s'\}$

Characterising $Sat(\Phi)$: the $\exists(\Phi \texttt{ UNTIL } \Psi)$ case

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

$Sat(\exists(\Phi \texttt{ UNTIL } \Psi))$ 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 $Sat(\exists(\Phi \ \texttt{UNTIL} \ \Psi))$ (1)

Suppose $T = Sat(\exists(\Phi \ \texttt{UNTIL} \ \Psi))$

$\exists(\Phi \ \texttt{UNTIL} \ \Psi)$ satisfies an “expansion law”:

$$\exists(\Phi \ \texttt{UNTIL} \ \Psi) \equiv \Psi \vee (\Phi \wedge \exists \bigcirc \exists(\Phi \ \texttt{UNTIL} \ \Psi))$$

$$\begin{aligned} T &= Sat(\exists(\Phi \ \texttt{UNTIL} \ \Psi)) \\ &= Sat(\Psi \vee (\Phi \wedge \exists \bigcirc \exists(\Phi \ \texttt{UNTIL} \ \Psi))) \\ &= Sat(\Psi) \cup (Sat(\Phi) \cap \{s \in S \mid Post(s) \cap Sat(\exists(\Phi \ \texttt{UNTIL} \ \Psi)) \neq \{\}\}) \\ &= Sat(\Psi) \cup (Sat(\Phi) \cap \{s \in S \mid Post(s) \cap T \neq \{\}\}) \end{aligned}$$

So:

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

Correctness of characterisation of $Sat(\exists(\Phi \mathtt{U} \Psi))$ (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 $Sat(\exists(\Phi \; \mathtt{UNTIL} \; \Psi)) \subseteq T$

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

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 $Sat(\exists(\Phi \mathtt{U} \Psi))$ (3)

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

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

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 $Sat(\Phi)$: the $\exists(\Box\Phi)$ case

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

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

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

Here, “largest” is interpreted with respect to set inclusion order

Correctness of characterisation of $Sat(\exists\Box\Phi)$ (1)

Suppose $T = Sat(\exists\Box\Phi)$

$\exists\Box\Phi$ also satisfies an "expansion law":

$$\exists\Box\Phi \equiv \Phi \wedge \exists \bigcirc \exists\Box\Phi$$

$$\begin{aligned} T &= Sat(\exists\Box\Phi) \\ &= Sat(\Phi \wedge \exists \bigcirc \exists\Box\Phi) \\ &= Sat(\Phi) \cap \{s \in S \mid Post(s) \cap Sat(\exists\Box\Phi) \neq \{\}\} \\ &= Sat(\Phi) \cap \{s \in S \mid Post(s) \cap T \neq \{\}\} \end{aligned}$$

So:

1. $Sat(\exists\Box\Phi) \subseteq Sat(\Phi)$
2. $s \in T$ implies $Post(s) \cap T \neq \{\}$

Correctness of characterisation of $Sat(\exists\Box\Phi)$ (2)

Suppose T satisfies:

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

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

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

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

$\pi[1]$ 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 $\pi \models \Box\Phi$ and $s \in Sat(\exists\Box\Phi)$

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

Recursive labelling algorithm

Pseudocode:

function SAT(Φ):
    switch Φ **do**:
        case $\top$: **return** S
        case p: **return** $\{s \in S \mid p \in \mathcal{L}(s)\}$
        case $\neg\Psi$: **return** $S - Sat(\Psi)$
        case $\Psi \wedge \Xi$: **return** $Sat(\Psi) \cap Sat(\Xi)$
        case $\exists\bigcirc\Psi$: **return** $\{s \in S \mid Post(s) \cap Sat(\Psi) \neq \{\}\}$
        case $\exists(\Psi\ \mathtt{UNTIL}\ \Xi)$: **return** SatExistsUntil(Ψ, Ξ)
        case $\exists(\Box\Psi)$: **return** SatExistsSquare(Ψ)
end function

Subprocedure SatExistsUntil

Pseudocode for SatExistsUntil:

```
function SatExistsUntil(Φ, Ψ):
    T ← Sat(Ψ)
    while {s ∈ Sat(Φ) − T | Post(s) ∩ T ≠ {}} ≠ {} do:
        s ← some state from {s ∈ Sat(Φ) − T | Post(s) ∩ T ≠ {}}
        T ← T ∪ {s}
    end while
    return T
end function
```

Subprocedure SatExistsSquare

Pseudocode for SatExistsSquare:

function SatExistsSquare(Φ):
    $T \leftarrow Sat(\Phi)$
    while $\{s \in T \mid Post(s) \cap T = \{\}\} \neq \{\}$ do
        $s \leftarrow$ some state from $\{s \in T \mid Post(s) \cap T = \{\}\}$
        $T \leftarrow T - \{s\}$
    end while
    return T
end function

Correctness of recursive labelling algorithm (1)

Recall $Sat(\exists(\Phi \texttt{ UNTIL } \Psi))$ is smallest $T \subseteq S$:

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

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

$$\begin{aligned} T_0 &= & Sat(\Psi) \\ T_{1+i} &= & T_i \cup \{s \in Sat(\Phi) \mid Post(s) \cap T_i \neq \{\}\} \end{aligned}$$

Iterate until fixed point is reached

T_i states can reach Ψ-state in at most i steps along Φ-path

SatExistsUntil implements this idea

Correctness of recursive labelling algorithm (2)

Recall $Sat(\exists\Box\Phi)$ is largest $T \subseteq S$:

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

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

$$\begin{aligned} T_0 &= Sat(\Phi) \\ T_{1+i} &= T_i \cap \{s \in Sat(\Phi) \mid Post(s) \cap T_i \neq \{\}\} \end{aligned}$$

Iterate until fixedpoint is reached

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 $\rightarrow$ 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

Computational complexity

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 $O(\mid \Phi \mid \cdot (V + E))$:

- V is number of states in model
- E is number of transitions in model
- $\mid \Phi \mid$ 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 $O(\mid \Phi \mid \cdot (V + E))$ complexity