CTL

Chapter 6 – Part 2
Overview

- Review CTL Model Checking
- CTL model Checking algorithms for
  - $\exists(\phi \cup \Psi)$
  - $\exists \Box \phi$
- Counter Examples and witnesses
- Symbolic Model Checking (Thursday)
  - Binary Decision Trees
  - Encoding Transition Systems by Switching functions
  - Symbolic Computation
  - Ordered Binary Decision Diagrams
Model checking CTL

- Convert the formula $\Phi'$ into the equivalent $\Phi$ in ENF

- How to check whether state $TS$ satisfies $\Phi$?
  - compute \textit{recursively} the set $Sat(\Phi)$ of states that satisfy $\Phi$
  - check whether all initial states belong to $Sat(\Phi)$

- Recursive \textit{bottom-up} computation:
  - consider the \textit{parse-tree} of $\Phi$
  - start to compute $Sat(\alpha)$, for all leafs in the tree
  - then go one level up in the tree and check the formula of these nodes
  - then go one level up and check the formula of these nodes
  - and so on........ until the root of the tree (i.e., $\Phi$) is checked
Example parse tree for CTL

\[ \Phi = \exists \Box a \land \exists (b \lor \Box \neg c) \]

Basic idea of CTL model checking

Input: finite transition system $TS$ and CTL formula $\Phi$ (both over $AP$)
Output: $TS \models \Phi$

(*) compute the sets $Sat(\Phi) = \{ s \in S \mid s \models \Phi \}$ *)

for all $i \leq |\Phi|$ do
  for all $\Psi \in Sub(\Phi)$ with $|\Psi| = i$ do
    compute $Sat(\Psi)$ from $Sat(\Psi')$ (* for maximal genuine $\Psi' \in Sub(\Psi)$ *)
  od
od
return $I \subseteq Sat(\Phi)$
Computing $\text{Sat}(\exists (\Phi \cup \Psi))$

*Input:* finite transition system $TS$ with state-set $S$ and CTL-formula $\exists (\Phi \cup \Psi)$

*Output:* $\text{Sat}(\exists (\Phi \cup \Psi)) = \{ s \in S \mid s \models \exists (\Phi \cup \Psi) \}$

$$E := \text{Sat}(\Psi); \quad \text{(* $E$ administers the states $s$ with $s \models \exists (\Phi \cup \Psi)$ *)}$$

$$T := E; \quad \text{(* $T$ contains the already visited states $s$ with $s \models \exists (\Phi \cup \Psi)$ *)}$$

while $E \neq \emptyset$ do

let $s' \in E$;

$E := E \setminus \{ s' \}$;

for all $s \in \text{Pre}(s')$ do

if $s \in \text{Sat}(\Phi) \setminus T$ then $E := E \cup \{ s \}; T := T \cup \{ s \};$ endif

od

od

return $T$
Example

let's check the CTL-formula $\exists \diamond ((p = r) \land (p \neq q))$

$\exists \diamond ((p=r) \land (p \neq q)) = \exists (true \ U ((p=r) \land (p \neq q)))$

Trace Algorithm on Whiteboard...
The computation in snapshots

(a) graph with nodes labeled by \{p, q, r\} and edges connecting them.

(b) graph with nodes labeled by \{q\} and \{p\}.

(c) graph with nodes labeled by \{q, r\} and \{p, q\}.

(d) final graph with nodes labeled by \{p, r\}.
Computing $\text{Sat}(\exists \Box \Phi)$

\[ E := S \setminus \text{Sat}(\Phi); \quad (* E \text{ contains any not visited } s' \text{ with } s' \not\models \exists \Box \Phi *) \]

\[ T := \text{Sat}(\Phi); \quad (* T \text{ contains any } s \text{ for which } s \models \exists \Box \Phi \text{ has not yet been disproven} *) \]

\[
\begin{array}{l}
\text{for all } s \in \text{Sat}(\Phi) \text{ do } c[s] := | \text{Post}(s) |; \od \\
\text{while } E \neq \emptyset \text{ do } \\
\quad \text{let } s' \in E; \\
\quad E := E \setminus \{ s' \}; \\
\quad \text{for all } s \in \text{Pre}(s') \text{ do } \\
\quad \quad \text{if } s \in T \text{ then } \\
\quad \quad \quad c[s] := c[s] - 1; \\
\quad \quad \quad \text{if } c[s] = 0 \text{ then } \\
\quad \quad \quad \quad T := T \setminus \{ s \}; E := E \cup \{ s \}; \\
\quad \quad \fi \\
\quad \fi \\
\od \\
\text{return } T
\end{array}
\]

\[ (* \text{initialize array } c *) \]

\[ (* \text{loop invariant: } c[s] = | \text{Post}(s) \cap (T \cup E) |; *) \]

\[ (* s' \not\models \Phi *) \]

\[ (* s' \text{ has been considered} *) \]

\[ (* \text{update counter } c[s] \text{ for predecessor } s \text{ of } s' *) \]

\[ (* s \text{ does not have any successor in } T *) \]
Compute $\exists \square q$

Trace Algorithm on Whiteboard...
Example

(a)

(b) $\mathcal{K}[q]$

(c) SCC

(d)
Alternative algorithm

1. Consider only state $s$ if $s \models \Phi$, otherwise eliminate $s$
   - change $TS$ into $TS[\Phi] = (S', Act, \rightarrow', I', AP, L')$ with $S' = Sat(\Phi)$,
   - $\rightarrow' = \rightarrow \cap (S' \times Act \times S')$, $I' = I \cap S'$, and $L'(s) = L(s)$ for $s \in S'$
   $\Rightarrow$ all removed states will not satisfy $\exists \square \Phi$, and thus can be safely removed

2. Determine all non-trivial strongly connected components in $TS[\Phi]$
   - non-trivial SCC = maximal, connected subgraph with at least one transition
   $\Rightarrow$ any state in such SCC satisfies $\exists \square \Phi$

3. $s \models \exists \square \Phi$ is equivalent to “some SCC is reachable from $s$”
   - this search can be done in a backward manner
Counter Examples and Witnesses

• Counter Examples
  • Indicate the refutation of universally quantified path formulae

• Witness
  • Indicate the satisfaction of existentially quantified path formulae

\[ \text{for } \forall \varphi, \text{ a prefix of } \pi \text{ with } \pi \nvdash \varphi \text{ acts as counterexample} \]
\[ \text{for } \exists \varphi, \text{ a prefix of } \pi \text{ with } \pi \models \varphi \text{ acts as witness} \]
The wolf-goat-cabbage problem

- A goat (g), a cabbage (c) and a wolf (w) and two riverbanks (0 and 1)
  - A boat with ferryman (f) that can carry at most two occupants
  - Only the ferryman can steer the boat
  - Goat and cabbage, goat and wolf should neither travel nor left together

- Is there a schedule such that brings c, g, and w to the other side?

- ... Model this as a CTL model-checking problem
  - transition system $TS = (\text{wolf} \ || \ || \ \text{goat} \ || \ || \ \text{cabbage}) \ || \ \text{ferryman}$
  - check whether $TS \models \exists \varphi$ with

$$
\varphi = \left( \bigwedge_{i=0,1} (w_i \land g_i \rightarrow f_i) \land (c_i \land g_i \rightarrow f_i) \right) \cup (c_1 \land f_1 \land g_1 \land w_1)
$$
The wolf-goat-cabbage problem

\[ TS = (\text{wolf} \ || \ \text{goat} \ || \ \text{cabbage}) \ || \ \text{ferryman} \]
Wolf-goat-cabbage problem

A witness of $\exists \varphi$ with:

$$\varphi = \left( \bigwedge_{i=0,1} (w_i \land g_i \rightarrow f_i) \land (c_i \land g_i \rightarrow f_i) \right) \cup (c_1 \land f_1 \land g_1 \land w_1)$$

is a path fragment from initial state $\langle c_0, f_0, g_0, w_0 \rangle$ to target state $\langle c_1, f_1, g_1, w_1 \rangle$ such that $g$, $c$ and $g$, $w$ are not left on a single riverbank. Such as:

$\langle c_0, f_0, g_0, w_0 \rangle$  goat to riverbank 1
$\langle c_0, f_1, g_1, w_0 \rangle$  ferryman comes back to riverbank 0
$\langle c_0, f_0, g_1, w_0 \rangle$  cabbage to riverbank 1
$\langle c_1, f_1, g_1, w_0 \rangle$  goat back to riverbank 0
$\langle c_1, f_0, g_0, w_0 \rangle$  wolf to riverbank 1
$\langle c_1, f_0, g_0, w_0 \rangle$  goat to riverbank 1
$\langle c_1, f_0, g_0, w_1 \rangle$  ferryman comes back to riverbank 0
$\langle c_1, f_0, g_0, w_1 \rangle$  goat to riverbank 1
Counterexamples for $\bigcirc \Phi$

- A counterexample of $\bigcirc \Phi$ is a path fragment $s \, s'$ with
  - $s \in I$ and $s' \in Post(s)$ with $s' \not\models \Phi$

- A witness of $\bigcirc \Phi$ is a is a path fragment $s \, s'$ with
  - $s \in I$ and $s' \in Post(s)$ with $s' \models \Phi$

- **Algorithm**: inspection of direct successors of initial states
Counterexamples for $\Phi \cup \Psi$

- A witness is an initial path fragment $s_0s_1 \ldots s_n$ with
  - $s_n \models \Psi$ and $s_i \models \Phi$ for $0 \leq i < n$

- **Algorithm**: backward search starting in the set of $\Psi$-states

- A counterexample is an initial path fragment that indicates a path $\pi$:
  - for which either $\pi \models \Box(\Phi \land \neg \Psi)$ or $\pi \models (\Phi \land \neg \Psi) \cup (\neg \Phi \land \neg \Psi)$

- Counterexample is initial path fragment of either form:
  - $s_0 \ldots s_{n-1} \overset{\text{cycle}}{s_n s'_1 \ldots s'_r}$ with $s_n = s'_r$ or
  - $s_0 \ldots s_{n-1} s_n$ with $s_n \models \neg \Phi \land \neg \Psi$
Counterexample generation

Determine the SCCs by of the **digraph** $G = (S, E)$ where

$$E = \{ (s, s') \in S \times S \mid s' \in \text{Post}(s) \land s \models \Phi \land \neg \Psi \}$$

Each path in $G$ that starts in an initial state $s_0 \in S$ and leads to a non-trivial SCC $C$ in $G$ provides a counterexample of the form:

$$s_0 s_1 \ldots s_n s'_1 \ldots s'_r \quad \text{with} \quad s_n = s'_r$$

Each path in $G$ that leads from an initial state $s_0$ to a trivial terminal SCC

$$C = \{ s' \} \quad \text{with} \quad s' \not\models \Psi$$

provides a counterexample of the form $s_0 s_1 \ldots s_n$ with $s_n \models \neg \Phi \land \neg \Psi$
Counterexamples for $\square \Phi$

- Counterexample is initial path fragment $s_0 s_1 \ldots s_n$ such that:
  - $s_0, \ldots, s_{n-1} \models \Phi$ and $s_n \not\models \Phi$

- Algorithm: backward search starting in $\neg \Phi$-states

- A witness of $\varphi = \square \Phi$ consists of an initial path fragment of the form:
  - $s_0 s_1 \ldots s_n s'_1 \ldots s'_r$ with $s_n = s'_r$
    - satisfy $\Phi$

- Algorithm: cycle search in the digraph $G = (S, E)$ where the set of edges $E$:
  - $E = \{(s, s') \mid s' \in \text{Post}(s) \land s \models \Phi\}$
Example

\forall \left( \left( (n_1 \land n_2) \lor w_2 \right) \cup c_2 \right)

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: katoen@cs.rwth-aachen.de
January 14, 2009
Time complexity

Let $TS$ be a transition system $TS$ with $N$ states and $K$ transitions and $\varphi$ a CTL-path formula.

If $TS \not\models \forall \varphi$ then a counterexample for $\varphi$ in $TS$ can be determined in time $O(N+K)$.

The same holds for a witness for $\varphi$, provided that $TS \models \exists \varphi$. 
Symbolic Model Checking

- Original version of CTL model checking
  - single state and single transition at a time
  - record all the predecessors and successors in each state
  - iterative computation: union and intersection of sets
  - state explosion problem in large transition systems

- Symbolic version
  - sets of states and sets of transitions at a time
  - binary encoding of states
  - one boolean function for each satisfaction set
  - one boolean function for all the transitions
  - iterative computation: conjunction and disjunction of a sequence of bits
  - very efficient
Binary encoding \((enc)\) of states, as vectors of \(n\) bits:

\[
enc : S \rightarrow \{0, 1\}^n
\]

For example:
8 states \((s_0, s_1, ..., s_7)\) can be encoded with 3 bits
\[
\begin{align*}
    s_0 & : 000 \\
    s_1 & : 001 \\
    \vdots & \\
    s_7 & : 111
\end{align*}
\]
Symbolic Model Checking

$X_T$: to encode set of states $T \subseteq S$ (i.e. $T = \text{Sat}(\alpha)$):

$$X_T : \{0,1\}^n \rightarrow \{0,1\} \quad \text{s.t.} \quad X_T(s) = 1 \iff s \in T$$

$\Delta$: to encode set of transitions $\rightarrow \subseteq S \times S$:

$$\Delta : \{0,1\}^{2n} \rightarrow \{0,1\} \quad \text{s.t.} \quad \Delta(s,s') = 1 \iff s \rightarrow s'$$
Symbolic Model Checking

- Encoding states and Transitions
Symbolic Model Checking, $\exists(\phi \cup \Psi)$

- Algorithm

```
Algorithm 20 Symbolic computation of $\text{Sat}(\exists(C \cup B))$

$f_0(\overline{x}) := \chi_B(\overline{x})$;
$j := 0$;
repeat
    $f_{j+1}(\overline{x}) := f_{j+1}(\overline{x}) \lor (\chi_C(\overline{x}) \land \exists \overline{x}'. (\Delta(\overline{x}, \overline{x}') \land f_j(\overline{x}'))) )$;
    $j := j + 1$
until $f_j(\overline{x}) = f_{j-1}(\overline{x})$
return $f_j(\overline{x})$.
```

- Correction

- Main operation in loop

\[
T_{j+1} = T_j \cup \{ s \in S \mid \exists s' \in S \text{ s.t. } s' \in \text{Post}(s) \land s' \in T_j \}
\]
Symbolic Model Checking, $\exists (\phi \ U \ \Psi)$

$\exists \diamond ((a = c) \land (a \neq b)) = \exists (true \ U ((a = c) \land (a \neq b)))$

Trace Algorithm with BDDs on Whiteboard...

We need:

- $Sat(a)$,
- $Sat(b)$,
- $Sat(c)$,
- Transitions ($s, s'$)
Symbolic Model Checking, $\exists \Box \phi$

- Algorithm

```
Algorithm 21 Symbolic computation of Sat($\exists \Box B$)

\[
\begin{align*}
    f_0(\bar{x}) & := \chi_B(\bar{x}); \\
    j & := 0; \\
    \text{repeat} \\
        f_{j+1}(\bar{x}) & := f_{j+1}(\bar{x}) \land \exists \bar{x}'. (\Delta(\bar{x}, \bar{x}') \land f_j(\bar{x}')); \\
        j & := j + 1 \\
    \text{until} f_j(\bar{x}) = f_{j-1}(\bar{x}); \\
    \text{return } f_j(\bar{x}).
\end{align*}
```

- Correction

- Main operation in loop

\[
T_{j+1} = T_j \cap \{ s \in S \mid \exists s' \in S \text{ s.t. } s' \in \text{Post}(s) \land s' \in T_j \}
\]
Symbolic Model Checking, $\exists \square \phi$

$\exists \square \phi$

Trace Algorithm with BDDs on Whiteboard...

We need:

• $\text{Sat}(b)$,

• Transitions $(s, s')$
Symbolic Model Checking - $\exists \square q$

8 states ($s_0, s_1, ..., s_7$) can be encoded with 3 bits ($z_1 z_2 z_3$)

$X_T(z_1, z_2, z_3) \rightarrow \{0,1\}, T=\text{Sat}(b)$

i.e. $X_T(s_2) = X_T(0,1,0) = 1$

$\Delta(z_1, z_2, z_3, z'_1, z'_2, z'_3) \rightarrow \{0,1\}$

i.e. $\Delta(s_1, s_2) = \Delta(0,0,1,0,1,0) = 0$
May not have a data structure which is efficient for all computations

Data structures which yield compact representations for many computations that appear in practical applications

OBDDs - data structure that has been proven to be very successful for model checking purposes
  - particularly in the area of hardware verification
  - Besides yielding compact representation for many “realistic” transition systems, they enjoy the property that the Boolean connectives can be realized in time linear in the size of the input OBDDs
  - with appropriate implementation techniques, equivalence checking can even be performed in constant time.
BDT to OBDD

• yield a data structure for computations that relies on a compactification of binary decision trees.

• skip redundant fragments of a binary decision tree.
  • This means collapsing constant subtrees (i.e., subtrees where all terminal nodes have the same value) into a single node
  • identifying nodes with isomorphic subtrees
  • we obtain a directed acyclic graph of outdegree 2
    - the inner nodes are labeled by variables and their outgoing edges stand for the possible evaluations of the corresponding variable.
    - The terminal nodes are labeled by the function value.
BDD to OBDD

• Consider BDD

• To ODBBs

*Correction: line from z3 to 0 should be a dashed line*
BDD to OBDD
Notation 6.62. Variable Ordering

Let $\text{Var}$ be a finite set of variables. A variable ordering for $\text{Var}$ denotes any tuple $\varphi = (z_1, \ldots, z_m)$ such that $\text{Var} = \{z_1, \ldots, z_m\}$ and $z_i \neq z_j$ for $1 \leq i < j \leq m$. We write $\prec_\varphi$ for the induced total order on $\text{Var}$. I.e., for $\varphi = (z_1, \ldots, z_m)$ the binary relation $\prec_\varphi$ on $\text{Var}$ is given by $z_i \prec_\varphi z_j$ if and only if $i < j$. We write $z_i \preceq_\varphi z_j$ iff either $z_i \prec_\varphi z_j$ or $i = j$. ■
Definition 6.63. Ordered Binary Decision Diagram (OBDD)

Let $\varphi$ be a variable ordering for $\text{Var}$. An $\varphi$-ordered binary decision diagram ($\varphi$-OBDD for short) is a tuple

$$\mathcal{B} = (V, V_I, V_T, \text{succ}_0, \text{succ}_1, \text{var}, \text{val}, v_0)$$

consisting of

- a finite set $V$ of nodes, disjointly partitioned into $V_I$ and $V_T$ where the nodes in $V_I$ are called inner nodes, while the nodes in $V_T$ are called terminal nodes or drains;
- successor functions $\text{succ}_0, \text{succ}_1 : V_I \rightarrow V$ that assign to each inner node $v$ a 0-successor $\text{succ}_0(v) \in V$ and a 1-successor $\text{succ}_1(v) \in V$;
- a variable labeling function $\text{var} : V_I \rightarrow \text{Var}$ that assigns to each inner node $v$ a variable $\text{var}(v) \in \text{Var}$;
- a value function $\text{val} : V_T \rightarrow \{0, 1\}$ that assigns to each drain a function value 0 or 1;
- a root (node) $v_0 \in V$. 