The Symbolic Model Checking Algorithm

Gregory Gelfond

Knowledge Representation Lab
Texas Tech University
Overview

• Modeling Domains (Kripke Structures)
• Specifying Properties (Temporal Logic CTL)
  • Syntax
  • Semantics
• CTL Model Checking
  • Ordered Binary Decision Diagrams
  • Quantified Boolean Formulas
  • CTL Model Checking Algorithm
Kripke Structures

• In model checking, domains are represented by Kripke structures.

• Intuitively, a Kripke structure specifies the states of the world, and transitions from one state to another.
Definition

Let $AP$ be a set of atomic propositions. A Kripke structure, $M$, over $AP$ is a 4-tuple $(S, S_0, R, L)$ where:

- $S$ is a finite set of states
- $S_0 \subseteq S$ is the set of initial states
- $R \subseteq S \times S$ is a total transition relation
- $L : S \rightarrow 2^AP$ is a function that labels each state with the set of atomic propositions true in that state
Each state is labeled with the set of literals true in that state.

- \( S = \{(a,b), (a,\neg b), (\neg a,b)\} \)
- \( R = \{((a,b),(a,\neg b)), ((a,b),(\neg a,b)), ((\neg a,b),(a,b)), ((\neg a,b),(a,\neg b))\} \)
Synchronous Modulo-8 Counter
A Kripke structure, $M$, that represents our counter consists of:

- $AP = \{ v_i = x : x \in \{0,1\} \text{ and } i \in [0..2]\}$

- $S = \{ (x,y,z) : x, y, z \in \{0,1\} \}.$

- $L(s) = \{ v_2 = x, v_1 = y, v_0 = z \}$

- A transition relation, $R$, defined as follows:
First we define the transitions for each state variable:

- \( R_0(V, V') = (v'_0 \equiv \neg v_0) \)
- \( R_1(V, V') = (v'_1 \equiv v_0 \oplus v_1) \)
- \( R_2(V, V') = (v'_2 \equiv (v_0 \land v_1) \oplus v_2) \)

where \( v'_i \) are new variables.
Since our circuit is synchronous, the formula describing the transition relation for the entire circuit is:

$$R_0(V,V') \land R_1(V,V') \land R_2(V,V')$$

The corresponding Kripke structure is shown on the right:
The Temporal Logic CTL

• In model checking, properties of paths are specified using a temporal logic.

• The particular temporal logic we will introduce is called CTL.
Syntax

**CTL** formulas are composed with *path quantifiers*, and *temporal operators*.

Path quantifiers are used to describe the branching structure of trees. There are two path quantifiers:

- **A** - universal path quantifier
- **E** - existential path quantifier
Temporal operators are used to specify properties of a path. There are five temporal operators:

- **X** - the “next time” operator ($Xp$ specifies that $p$ holds in the second state of the path)

- **F** - the “future time” operator ($Fp$ specifies that $p$ holds at some state in the path)
• **G** - the “always” operator ($Gp$ specifies that $p$ holds at every state in the path)

• **U** - the “until” operator ($p U g$ specifies that $g$ holds at some state in the path, and $p$ is guaranteed to hold along the path up to the first state in which $g$ holds)

• **R** - the “release” operator ($p R g$ specifies that $g$ holds along the path up to and including the first state where $p$ holds)
Let $AP$ be the set of atomic propositions.

- If $p \in AP$, then $p$ is a formula.

- If $f$ and $g$ are formulas, then: $\neg f, f \lor g, \text{ and } f \land g$, are formulas.

- $EXf, EFf, EGf, E[f \ U \ g], E[f \ R \ g]$ are formulas.

- $AXf, AFf, AGf, A[f \ U \ g], A[f \ R \ g]$ are also formulas.
Examples

- The following are examples of valid CTL formulas ($p$ and $q \in AP$):

  $p, p \land q, AGp, EXPp \lor AGq$

- The following are not valid formulas:

  $Ap, Xq \lor Gr$
Semantics

The semantics of CTL will be given with respect to a Kripke structure $M$. A *path in $M$* is an infinite sequence of states, $\pi = s_0, s_1, \ldots$ such that for every $i \geq 0$, $(s_i, s_{i+1}) \in R$.

$\pi^i$ will be used to denote the *suffix* of $\pi$ starting at $s_i$.

If $f$ is a CTL formula, then $M, s \models f$ means that $f$ holds in state $s$ in the Kripke structure $M$. 
The relation $\Rightarrow$ is defined as follows:

- $M,s_0 \Rightarrow p \equiv p \in L(s_0)$
- $M,s_0 \Rightarrow \neg p \equiv \neg(M,s_0 \Rightarrow p)$
- $M,s_0 \Rightarrow p \land q \equiv M,s_0 \Rightarrow p \text{ and } M,s_0 \Rightarrow q$
- $M,s_0 \Rightarrow p \lor q \equiv M,s_0 \Rightarrow p \text{ or } M,s_0 \Rightarrow q$
• $M, s_0 \Rightarrow \mathbf{E X}_p \equiv \exists s_1 : (s_0, s_1) \in R$, and $M, s_1 \Rightarrow p$

• $M, s_0 \Rightarrow \mathbf{E G}_p \equiv \exists \text{ path } \pi \text{ starting at } s_0 \text{ such that } \forall s_i \in \pi M, s_i \Rightarrow p$

• $M, s_0 \Rightarrow \mathbf{E}[p \mathbf{U} g] \equiv \exists \text{ path } \pi \text{ starting at } s_0$, 
  $\exists i \geq 0$ such that $M, s_i \Rightarrow g$, and $\forall j : 0 \leq j < i$, $M, s_j \Rightarrow p$
There are seven CTL operators:

- $AX$ and $AF$
- $EF$ and $AG$
- $AU$, $AR$ and $ER$
Each of these operators can be viewed as shorthand for the following:

- \( AXf \equiv \neg EX(\neg f) \)
- \( EFf \equiv E[True \ U \ f] \)
- \( AGf \equiv \neg EF(\neg f) \)
- \( AFf \equiv \neg EG(\neg f) \)
• $A[f \mathcal{U} g] \equiv \neg E[\neg g \mathcal{U} (\neg f \wedge \neg g)] \wedge \neg E G(\neg g)$

• $A[f \mathcal{R} g] \equiv \neg E[\neg f \mathcal{U} \neg g]$

• $E[f \mathcal{R} g] \equiv \neg A[\neg f \mathcal{U} \neg g]$
Consider the following Kripke structure $M$:

- Each state is labeled with the set of literals true in that state.
- $S = \{(a,b), (a,\neg b), (\neg a,b)\}$
- $R = \{((a,b),(a,\neg b)),
((a,b), (\neg a,b)),
((a,\neg b),(a,b)),
((\neg a,b),(a,\neg b))\}$
• One method of representing $M$ would be to specify the transition relation as a table.

• Unfortunately, for complex structures the table becomes too large.

• Consequently, a more efficient data structure is needed.
R = {((a,b),(a,¬b)), ((a,b),(¬a,b)), ((a,¬b),(a,b)), ((¬a,b),(a,¬b))}

By introducing a pair of next-state variables, a’ and b’, we can obtain the formula F(R):

\[(a \land b \land a' \land \neg b') \lor (a \land b \land \neg a' \land b') \lor (a \land \neg b \land a' \land b') \lor \]

\[\neg a \land b \land a' \land b')\]

Valid transitions of M are models of F(R).

F(R) can be represented by a binary decision tree:
There is a drawback to the binary decision tree in that it stores a great deal of redundant information in the form of equivalent subtrees.

The following algorithm, implemented by a function Reduce, takes a binary decision tree as input and removes the redundant subtrees, giving us an ordered binary decision diagram:
• **Remove duplicate terminals** - we eliminate all but one leaf with a given label, and redirect all arcs to the eliminated vertices to their counterpart.

• **Remove duplicate nonterminals** - if two nonterminals \( u \) and \( v \) are roots of identical subtrees, then remove \( u \) and redirect its incoming arcs to \( v \).

• **Remove redundant checks** - if the children of a nonterminal \( v \) are roots of identical subtrees, then we remove \( v \), and redirect incoming arcs to one of its children.
Redundant Checks
Redundant Checks
Subtrees Whose Roots are Redundant Nonterminals
Subtrees Whose Roots are Redundant Nonterminals
Redundant Terminals
Given an OBDD for a boolean function $F$, we can construct an OBDD for the function that restricts the value of an argument $x$ of $F$ to a boolean value $b$ (denoted by $F|_{x=b}$) as follows:

- For any node $v$ that has an arc to a node $w$ labeled by $x$, redirect the arc to $\text{low}(w)$ if $b = 0$, and to $\text{high}(w)$ otherwise.

- We then reduce the OBDD as described previously.
Example

Let $F = a \land b$. The OBDD for $F$ is shown below:

Applying $F|_{b=1}$ yields the following OBDD:
• Let $f_1$ and $f_2$ denote boolean functions

• Let $v_1$ and $v_2$ denote the roots of the OBDDs representing $f_1$ and $f_2$

• Let $x_1$ and $x_2$ denote the variables labeling $v_1$ and $v_2$

• Let $\ast$ denote $\land$ or $\lor$
Given the OBDD’s for \( f_1 \) and \( f_2 \), we can obtain the OBDD representing \( F = f_1 \ast f_2 \) as follows:

- If \( v_1 \) and \( v_2 \) are terminal nodes then
  \[ f_1 \ast f_2 = \text{value}(v_1) \ast \text{value}(v_2) \]

- If \( v_1 \) and \( v_2 \) are both labeled by \( x \) then we construct a new OBDD whose root is a new node \( w \) labeled by \( x \)
  - \( \text{low}(w) = \text{the OBDD for } f_1 |_{x=0} \ast f_2 |_{x=0} \)
  - \( \text{high}(w) = \text{the OBDD for } f_1 |_{x=1} \ast f_2 |_{x=1} \)
• If $x_1 < x_2$ then we construct a new OBDD whose root is a new node $w$ labeled by $x_1$
  
  • $low(w) =$ the OBDD for $f_1|_{x_1=0} * f_2$
  
  • $high(w) =$ the OBDD for $f_1|_{x_1=1} * f_2$
  
  • Similarly if $x_1 > x_2$
  
  The preceding algorithm is implemented by the function $Apply$
Example

- Consider \( f_1 = a \land b \) and \( f_2 = a \land c \)
- Let \( a < b < c \)
- The OBDDs for \( f_1 \) and \( f_2 \) are as follows:
• The OBDDs have the same root, therefore we apply the second case of the algorithm:

• We obtain the following OBDD for $f_1|_{a=0} \land f_2|_{a=0}$:
• Similarly we obtain the following OBDDs for $f_1|_{a=1}$ and $f_2|_{a=1}$:
• Combining the OBDDs for $f_1|_{a=1}$ and $f_2|_{a=1}$ yields the following:
• Combing the OBDDs for $f_1|_{a=0} \land f_2|_{a=0}$ and $f_1|_{a=1} \land f_2|_{a=1}$ gives us the following:
• Reducing the graph gives us the final OBDD:
Given a boolean formula $F$, we can construct the OBDD representing $F$ using the following recursive algorithm that is based on the construction of $F$: 
• If $F = True$, then the OBDD is a single terminal node labeled by 1.

• If $F = a$, then the OBDD is as follows:
• If $F = \neg g$, we compute the OBDD for $g$, and invert the terminal nodes.

• Lastly, if $F = f_1 \ast f_2$ we compute the OBDDs for $f_1$ and $f_2$ and then obtain the OBDD for $f_1 \ast f_2$ as was previously discussed.
Quantified Boolean Formulas

- Given a set \( V = \{v_0 \ldots v_{n-1}\} \) of propositional variables, \( QBF(V) \) is the smallest set of formulas such that:
  
  - every variable in \( V \) is a formula
  
  - if \( f \) and \( g \) are formulas, then \( \neg f, f \lor g, \) and \( f \land g \) are formulas
  
  - if \( f \) is a formula, and \( \nu \in V \), then \( \exists \nu f \) and \( \forall \nu f \) are formulas
• A function $\sigma: V \rightarrow \{0, 1\}$ is a truth assignment for $QBF(V)$.

• If $a \in \{0, 1\}$, then $\sigma[v \leftarrow a](w)$ is defined as follows:
  
  • $a$ if $v = w$
  
  • $\sigma(w)$ otherwise
If $f \in QBF(V)$, and $\sigma$ is a truth assignment, $\sigma \Rightarrow f$ is defined as follows:

- $\sigma \Rightarrow v \equiv \sigma(v) = 1$
- $\sigma \Rightarrow \neg f \equiv \neg(\sigma \Rightarrow f)$
- $\sigma \Rightarrow f \land g \equiv \sigma \Rightarrow f$ and $\sigma \Rightarrow g$
- $\sigma \Rightarrow f \lor g \equiv \sigma \Rightarrow f$ or $\sigma \Rightarrow g$
- $\sigma \Rightarrow \exists v f \equiv \sigma[v \leftarrow 0] \Rightarrow f$ or $\sigma[v \leftarrow 1] \Rightarrow f$
- $\sigma \Rightarrow \forall v f \equiv \sigma[v \leftarrow 0] \Rightarrow f$ and $\sigma[v \leftarrow 1] \Rightarrow f$
We have already seen how to represent non-quantified formulas as OBDDs. We can also compute the OBDDs for quantified formulas using the following identities:

- $\exists x f = f|_{x=0} \lor f|_{x=1}$
- $\forall x f = f|_{x=0} \land f|_{x=1}$
The CTL model checking algorithm is implemented by a function `Check` that takes a Kripke structure $M$ and a CTL formula $f$ as parameters, and returns an OBDD representing the set $S = \{s : M, s \models f\}$.

$M$ satisfies $F$ if the set of initial states belongs to $S$. 
The function Check operates as follows:

- If $F$ is an atom, $\text{Check}(F) =$ the OBDD representing the set of states containing $F$.

- If $F = \neg f$, $\text{Check}(F) = \text{Apply}(\neg, \text{Check}(f))$

- If $F = f \ast g$, $\text{Check}(F) = \text{Apply}(\ast, \text{Check}(f), \text{Check}(g))$
• $\text{Check}(\text{EX } f) = \text{CheckEX}(\text{Check}(f))$

• $\text{Check}(\text{E}[f \textbf{ U } g]) = \text{CheckEU}(\text{Check}(f), \text{Check}(g))$

• $\text{Check}(\text{EG } f) = \text{CheckEG}(\text{Check}(f))$
The function `CheckEX` takes as a parameter an OBDD representing the set of states satisfying a formula $f$, and returns the OBDD for the quantified boolean formula $\exists v'[f(v') \land R(v,v')]$
The function $CheckEU$ takes as parameters the OBDDs representing the sets of states satisfying the formulas $f$ and $g$, and returns the OBDD corresponding to

$$\mu Z. \, g \vee (f \land EX \, z)$$

where $\mu Z. \, g \vee (f \land EX \, z)$ is the least fixpoint characterization of $E[f U g]$.
The function $\text{CheckEG}$ takes as parameters the OBDDs representing the set of states satisfying the formula $f$, and returns the OBDD corresponding to

$$\nu Z. f \land \text{EX} Z$$

where $\nu Z. f \land \text{EX} Z$ is the greatest fixpoint characterization of $\text{EG}$