CS 745 / ECE 725
Computer Aided Verification

Lecture 8: BDDs

Jo Atlee

DC 2337, jmatlee@uwaterloo.ca

Office Hours: Mon 1:00-2:00, Wed 1:00-2:00

http://www.student.cs.uwaterloo.ca/~cs745

Review: Model Checking

Problem

Given a model $\mathcal{M}$ (usually a Kripke structure) that represents the behaviour of a system, and a temporal logic formula f that represents a desired property of the system, determine whether the model satisfies the formula:

$$\mathcal{M} \models f$$

Model checking algorithms we're studying:

- Explicit CTL Model Checking: labelling a graph
- Symbolic CTL Model Checking: representing sets of states using Boolean functions

Review: Symbolic Model Checking

1. Describe the model checking algorithm as computations over sets of states; abstract the operation of finding the set of states that can reach another set of states as a pre-image computation:

   function SAT_EX (f_1)
   K = SAT (f_1);
   Y = $\{t \in S \mid \exists s \bullet\ s \in K \wedge (t, s) \in R\}$
   return Y;

2. Use characteristic functions to represent sets of states

   $$S = (x_1 \cdot \overline{x_2}) + (\overline{x_1} \cdot x_2) + (\overline{x_1} \cdot \overline{x_2})$$

3. Represent the transition relation using a Boolean function over sets of current and next states.

   $$R = (\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_1'} \cdot \overline{x_2'}) + (\overline{x_1} \cdot \overline{x_2} \cdot x_1' \cdot \overline{x_2'}) + (\overline{x_1} \cdot x_2 \cdot \overline{x_1'} \cdot \overline{x_2'}) + (x_1 \cdot \overline{x_2} \cdot \overline{x_1'} \cdot x_2')$$

Today's Agenda

- CTL formulae as fixed point operations in the μ-calculus
- Binary Decision Diagrams – a data structure for manipulating Boolean functions

μ-Calculus

- Notation for describing properties of transition systems
- Uses fixed point operators in addition to logical connectives (no temporal operators)
- Many temporal logics can be expressed as μ-calculus formulae

We will only cover enough of the μ-calculus to show how to encode CTL formulae.

Fixpoints: Intuition

SAT_EU and SAT_EG are iterative until they reach a point where "no new relevant states are being considered". This corresponds to the notion of a fixed point.

function SAT_EG(f_1)
 K := SAT(f_1);
 do
  oldK := K;
  K := oldK $\cap$ pre$_\exists$ (oldK);
 until oldK = K;
 return K;

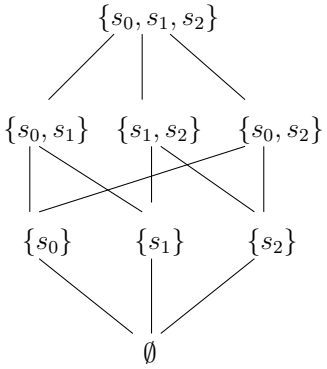
function SAT_EU (f_1,f_2)
 K = SAT(f_2); W = SAT(f_1);
 do
  oldK := K;
  K := oldK $\cup$ (W $\cap$ pre$_\exists$ (oldK))
 until oldK = K
 return K;

Power Set of a Set

For the Kripke structure $\mathcal{M} = (S, R, L)$, the set $\mathbb{P}(S)$ is the set of all subsets of S.

The set of all subsets forms a lattice with the ordering of set inclusion.

The least element is the empty set ($\emptyset$). The greatest element is the set S.



$$S = \{s_0, s_1, s_2\}$$

Monotonic Functions

It is possible to compute a fixed point for monotonic functions:

A function $f : \mathbb{P}(S) \rightarrow \mathbb{P}(S)$ is monotonic iff forall $P \subseteq S$ and $Q \subseteq S$

$$P \subseteq Q \Rightarrow f(P) \subseteq f(Q)$$

Fixed Points

A fixed point of a function f is an element x such that $f(x) = x$.

The least fixed point in a lattice for a function f is the least element that is a fixed point. y is the lfp of f in S iff

$$(f(y) = y) \wedge (\forall x \in S \bullet (f(x) = x) \Rightarrow (y \subseteq x))$$

The least fixed point of the function f is denoted as $\mu Z.f(Z)$

The greatest fixed point in a lattice for a function f is the greatest element that is a fixed point. y is the gfp of f in S iff

$$(f(y) = y) \wedge (\forall x \in S \bullet (f(x) = x) \Rightarrow (x \subseteq y)$$

The greatest fixed point of function f is denoted as $\nu Z.f(Z)$

Fixed Points Example

Example: $f_1(Y) = Y \cup \{s_0\}$

Y	$f_1(Y)$
$\{s_0, s_1, s_2\}$	$\{s_0, s_1, s_2\}$
$\{s_0, s_1\}$	$\{s_0, s_1\}$
$\{s_1, s_2\}$	$\{s_0, s_1, s_2\}$
$\{s_0, s_2\}$	$\{s_0, s_2\}$
$\{s_0\}$	$\{s_0\}$
$\{s_1\}$	$\{s_0, s_1\}$
$\{s_2\}$	$\{s_0, s_2\}$
$\emptyset$	$\{s_0\}$

Useful Theorems

Thm 1: If S is finite and f is a monotonic function, then for some i:

$$\mu Z.f(Z) = \cup_i f^i(\emptyset)$$

and

$$\nu Z.f(Z) = \cap_i f^i(S)$$

This gives us a way to compute the lfp and gfp.

$$\begin{aligned} \mu Z.f(Z) &= \cup_i f^i(\emptyset) \\ &= f(\emptyset) \cup f(f(\emptyset)) \cup f(f(f(\emptyset))) \ldots \end{aligned}$$

Calculating Least Fixed Points

Thm 2: If f grows monotonically, $\forall i.f^i(\emptyset) \subseteq f^{i+1}(\emptyset)$

1. $\emptyset \subseteq f(\emptyset)$ because $\emptyset \subseteq$ anything
2. $f(\emptyset) \subseteq f(f(\emptyset))$ by monotonicity
3. $f^{i+1}(\emptyset) \subseteq f^{i+2}(\emptyset)$ by induction hypothesis and montonicity

$$\begin{aligned} \mu Z.f(Z) &= \cup_i f^i(\emptyset) \\ &= f(\emptyset) \cup f(f(\emptyset)) \cup f(f(f(\emptyset))) \ldots \\ &= f^i(\emptyset) \text{ for some } i \end{aligned}$$

Because f is monotonic and S is finite, $\exists i.f^i(\emptyset) = f^{i+1}(\emptyset)$.
This is a fixed point and is the least fixed point of f!

Calculating Greatest Fixed Points

Thm 3: If f shrinks monotonically, $\forall i. f^{i+1}(S) \subseteq f^i(S)$

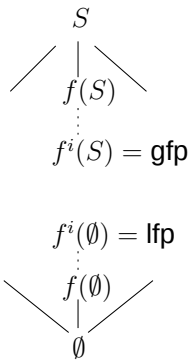
1. $f(S) \subseteq S$ because anything $\subseteq S$
2. $f(f(S)) \subseteq f(S)$ by monotonicity
3. $f^{i+2}(S) \subseteq f^{i+1}(S)$ by induction hypothesis and monotonicity

$$\begin{aligned} \nu Z.f(Z) &= \cap_i f^i(S) \\ &= f(S) \cap f(f(S)) \cap f(f(f(S))) \ldots \\ &= f^i(S) \text{ for some } i \end{aligned}$$

Because f is monotonic and S is finite, $\exists i. f^i(S) = f^{i+1}(S)$.
This is a fixed point and is the greatest fixed point of f!

Calculating Fixed Points

```
fp f e  =  if (e = f e)
           then e
           else fp f (f e)
lfp f   =  fp f ∅
gfp f   =  fp f S
```



Checking EG

function SAT_EG(f_1)
 K := SAT(f_1);
 do
  oldK := K;
  K := oldK ∩ pre$_\exists$ (oldK);
 until oldK = K;
 return K;

Unfolding the loop:
K = SAT (f_1)
K = SAT $(f_1) \cap$ pre$_\exists$(SAT(f_1))
K = SAT $(f_1) \cap$ pre$_\exists$(SAT$(f_1)) \cap$ pre$_\exists$(SAT$(f_1) \cap$ pre$_\exists$(SAT(f_1)))
. . . until it reaches a fixed point.

SAT_EG $f_1 = \nu Z.$ SAT$(f_1) \cap$ pre$_\exists$ Z

Fixed Points for CTL

Identify each CTL formula f with the set of states satisfying that formula:

- **EF** $f = \mu Z. f \vee$ pre$_\exists$ Z
- **EG** $f = \nu Z. f \wedge$ pre$_\exists$ Z
- **E**$[f_1$ **U** $f_2] = \mu Z. f_2 \vee (f_1 \wedge$ pre$_\exists$ $Z)$

Intuitively least fixed points correspond to eventualities, and greatest fixed points correspond to properties that should hold forever.

Model Checking Problem

Expanding definitions, each CTL formula encoded in μ-calculus includes the Boolean functions for the model's set of states S and transition relation R:

EG $f = \nu Z.f \wedge \text{pre}_\exists\ Z$

Expanding the definition of $\text{pre}_\exists$, we get:

EG $f = \nu Z(x_0, ..., x_n).\ f \cap Z(x'_0, ..., x'_n) \cap R(x_0, ..., x_n, x'_0, ..., x'_n)$

Today's Agenda

- CTL formulae as fixed point operations in the μ-calculus
- Binary Decision Diagrams – a data structure for manipulating Boolean functions

Representing Boolean functions

Both truth tables and propositional formulae are different ways of representing Boolean functions.

When implementing automated logical reasoning on a computer, we want to have compact representations of Boolean formulae with efficient operations on those formulae.

A representation for Boolean formulae plus a way of determining the validity of those formulae is a proof procedure for propositional logic.

Other relevant operations are:

- constructing formulae
- comparing two formulae for equality (a special case of validity)
- checking satisfiability (for counterexamples)

Worst Case Limits

Most of the rest of the presentation is based on Kropf [Kro99]

Every representation of Boolean functions has the same problem complexity:

> Determining the satisfiability of a formula is NP complete; determining the validity of a formula is co-NP complete.

This means in the worst case we have exponential run-time.

But can we do better for many cases?

Binary Decision Trees

A Boolean function can be represented by an decision tree (a rooted, directed tree).

Non-terminal vertices are labelled by Boolean variables and have two branches corresponding to the cases when the value of the variable is T and F.

The terminal vertices are labelled with T or F.

The value of the function for its inputs is obtained by traversing the decision nodes from root to leaf.

Reduced BDDs

The tree representation contains redundant information. Akers had the idea to merge redundant information to create reduced BDDs. Use a directed, acyclic graph (dag) instead of a tree to merge redundancies.

- All isomorphic subtrees are combined.
- Every node with isomorphic children is eliminated (and replaced by one of its children)

Algorithm to reduce BDD

Visit the nodes in a bottom-up order, combining nodes and subtrees where possible.

```
merge leaves with the same value;
for i := height(BDD) downto 1 do
  for all nodes v at level i do
    if left(v) = right(v)
       then replace v by left(v);

    if exists v'. (v' at level i
              and left(v') = left(v)
              and right(v') = right(v)
       then substitute v by v'
```


Reduction Example [Kro99], p. 43

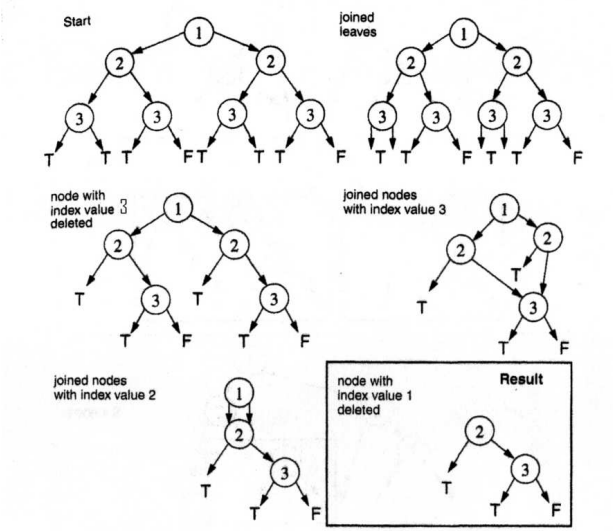
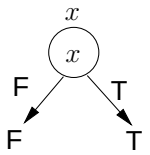


Fig. 2-5. ROBDD reduction example

Simplest ROBDDs

F T



Ordered BDDs

Bryant [Bry86, Bry92] had the idea to impose a variable ordering on the BDDs. On every path from the root to a terminal node, the variables appear in the same order with no repeated variables. All variables do not have to appear in all paths. Reduced Ordered BDDs are often just called BDDs.

The result is a canonical representation of a Boolean function for a given variable ordering.

Comparison, Satisfiability, Validity

Given a canonical representation for Boolean functions (ROBDDs), comparison of two ROBDDs can be done by checking if their representations have the same structure. Usually this means checking if two pointers are equal (constant time).

Validity: check if the ROBDD is equal to the ROBDD for the constant true function.

Satisfiability: check that the ROBDD isn't equal to the ROBDD for the constant false.

ROBDDs can be considered a proof procedure for propositional logic.

Operations on ROBDDs

- equality – pointer equality
- negation – complement the leaf values
- conjunction, disjunction – apply a binary operation
- pre-image – exists and disjunction

Apply

Intuitively, applying a Boolean operation $\star$ to two ROBDDs operates recursively on the structure of the operands:

1. Let v_i be the variable that is highest in the variable ordering, and that occurs in at least one of the operands
2. Consider the case where v_i is 0
3. Consider the case where v_i is 1

where $\star$ is any binary operation.

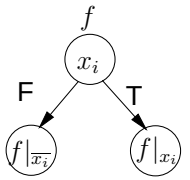
Uses Shannon's Expansion

Shannon's Expansion Theorem

The cofactor of f with regard to x_i, written as $f|_{x_i}$ is $f(x_1, \ldots, x_{i-1}, \mathsf{T}, x_{i+1}, \ldots, x_n)$

The cofactor of f with regard to $\overline{x_i}$, written as $f|_{\overline{x_i}}$ is $f(x_1, \ldots, x_{i-1}, \mathsf{F}, x_{i+1}, \ldots, x_n)$

Shannon's Expansion:
$f(x_1, \ldots, x_i, \ldots, x_n) = (x_i \cdot f|_{x_i}) + (\overline{x_i} \cdot f|_{\overline{x_i}})$



Apply

To determine the result of applying a Boolean operation to two ROBDDs use the Shannon expansion.

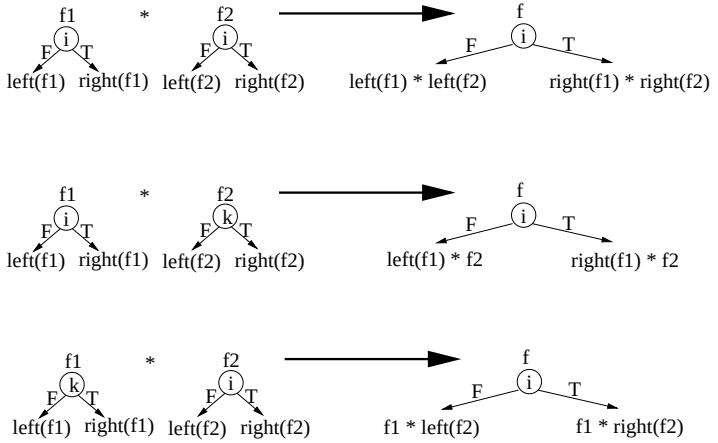
$f = f_1 \star f_2$

$$\begin{aligned} f &= x_i \cdot f|_{x_i} + \overline{x_i} \cdot f|_{\overline{x_i}} \\ &= x_i \cdot (f_1 \star f_2)|_{x_i} + \overline{x_i} \cdot (f_1 \star f_2)|_{\overline{x_i}} \\ &= x_i \cdot (f_1|_{x_i} \star f_2|_{x_i}) + \overline{x_i} \cdot (f_1|_{\overline{x_i}} \star f_2|_{\overline{x_i}}) \end{aligned}$$

(The cofactor is distributive with regard to Boolean operations.)

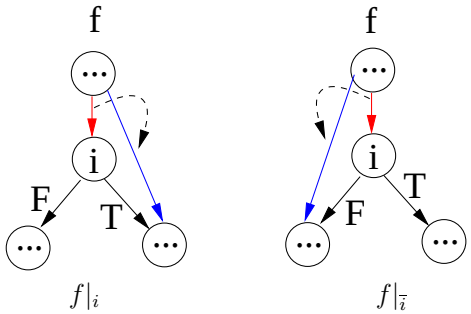
Apply [Kro99], p44

$i < k$



Cofactor (Restriction)

Cofactoring (restriction): $f|_i$
Eliminate all nodes labelled by i.



Pre-image

$$\chi \text{pre}_\exists Y(v_0, \ldots, v_n) =$$
$$\exists v'_0, \ldots, v'_n \bullet \chi_Y(v'_0 \ldots, v'_n) \cdot \chi_R(v_0, \ldots, v_n, v'_0, \ldots, v'_n)$$

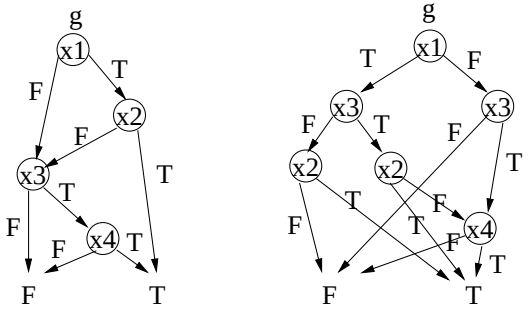
In ROBDDs, existential quantification can be implemented using disjunction and restriction: $\exists x. f = f|_{x=\mathsf{F}} \vee f|_{x=\mathsf{T}}$

We can compute the pre-image using $\exists$ and $\vee$ but this is such a frequent operation that we do better by implementing this operation directly.

The relational product is: $\exists x.\, R \wedge Y$, where R is a function of the Boolean vectors x and x', and Y is a function of x.
ROBDD packages include implementations of this operation.

Variable Order

Different variable orders can produce quite different ROBDDs.



Heuristics for Variable Order

Rough heuristic: group related variables together.

For example, in a ripple-carry adder, interleave the variables from each operand.

ROBDDs do not work well for representing multiplication because all variables are related to each other.

Lots of packages do dynamic variable reordering (on-the-fly attempts at reducing the size of the ROBDD through local optimizations).

ROBDD packages

- usually implemented using pointers
- multi-rooted ROBDD to represent many ROBDDs all using same variable ordering
- negative pointers to represent negation of a ROBDD

Existing packages:

- VIS `http://www-cad.eecs.berkeley.edu/Respep/Research/vis/` (does model checking also)
- CMU `http://www-2.cs.cmu.edu/ modelcheck/bdd.html`
- CUDD `http://vlsi.colorado.edu/ fabio/CUDD/cuddIntro.html`

Note: there are many variety of decision diagrams such as multi-way decision graphs [CZS+94], and binary moment diagrams [BC94].

Symbolic Model Checking

The first paper to describe symbolic model checking used ROBDDs to represent sets of states. This paper is by Burch, Clarke, McMillan, Dill, and Hwang [BCM+90].

Explicit state model checking could handle systems of size $10^3 - 10^6$ states.

Symbolic model checking could handle models of size 10^{20}.

(These numbers are from the 1990 paper – they are higher now.)

Summary

- CTL formulae as fixed point operations in the μ-calculus
  - Symbolic when we represent the sets of states by Boolean functions
- Binary Decision Diagrams – a data structure for manipulating Boolean functions

Model Checking

We've studied:

- explicit CTL model checking
- symbolic CTL model checking
- symbolic fixpoint CTL model checking

There are also LTL, CTL*, and μ-calculus model checkers.

Next class: Semantically configurable modelling notations

IBM, 1994. Also *Formal Methods in Systems Design*, 10(1) , pages 7-46 , 1997.

[Kro99] Thomas Kropf. Introduction to Formal Hardware Verification. Springer, 1999.

References

[BC94] Randal E. Bryant and Yirng-An Chen. Verification of arithmetic functions with binary moment diagrams. Technical Report CMU-CS-94-160, School of Computer Science, Carnegie Mellon University, June 1994.

[BCM+90] J. R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In Proceedings of the Fifth Annual Symposium on Logic in Computer Science, June 1990.

[Bry86] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers, C-35(8):677–691, August 1986.

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. ACM Computing Surveys, 24(3):293–318, September 1992. Preprint version published as CMU Technical Report CMU-CS-92-160.

[CZS+94] F. Corella, Z. Zhou, X. Song, M. Langevin, and E. Cerny. Multiway decision graphs for automated hardware verification. Technical Report RC19676,