Model Checking

Lecture #19: Symbolic Model Checking with BDDs

[Baier & Katoen, Chapter 6.7]

Joost-Pieter Katoen

Software Modeling and Verification Group

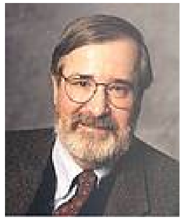
Model Checking Course, RWTH Aachen, WiSe 2019/2020

Overview

Overview

Inventors of BDD-Based Model Checking

Randy Bryant (USA)



Edmund Clarke Jr. (USA)

Ken McMillan (USA)

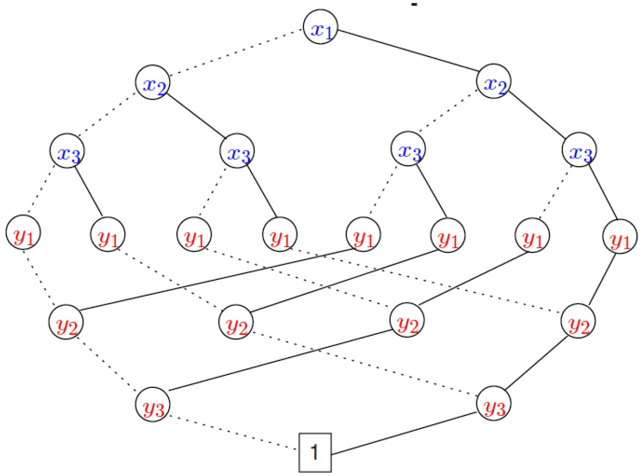
David Dill (USA)

Overview

Variable Ordering

- ROBDDs are canonical for a fixed variable ordering
  - the size of the ROBDD crucially depends on the variable ordering
  - $\#$ nodes in $\wp$- ROBDD $\mathfrak{B}$ = $\#$ of $\wp$-consistent co-factors of f
- Some switching functions have linear and exponential ROBDDs
  - e.g., the addition function, or the stable function (see below)
- Some switching functions only have polynomial ROBDDs
  - this holds, e.g., for symmetric functions (see next)
  - examples $f(\ldots) = x_1 \oplus \ldots \oplus x_n$, or $f(\ldots) = 1$ iff $\geq k$ variables x_i are true
- Some switching functions only have exponential ROBDDs
  - this holds, e.g., for the middle bit of the multiplication function

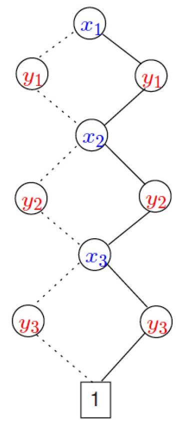
Function Stable with Exponential ROBDD



The ROBDD of $f_{stab}(\overline{x}, \overline{y}) = (x_1 \leftrightarrow y_1) \wedge \ldots \wedge (x_n \leftrightarrow y_n)$

has $3{\cdot}2^n{-}1$ vertices under ordering $x_1 < \ldots < x_n < y_1 < \ldots < y_n$

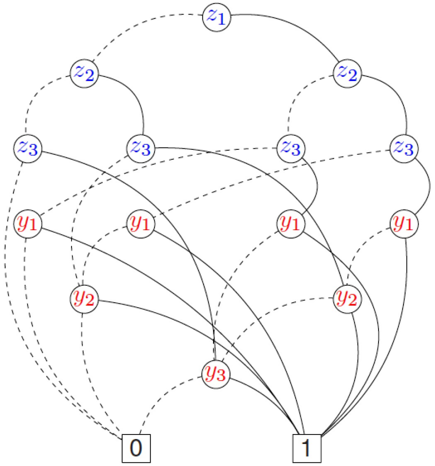
Function Stable with Linear ROBDD



The ROBDD of $f_{stab}(\overline{x}, \overline{y}) = (x_1 \leftrightarrow y_1) \wedge \ldots \wedge (x_n \leftrightarrow y_n)$

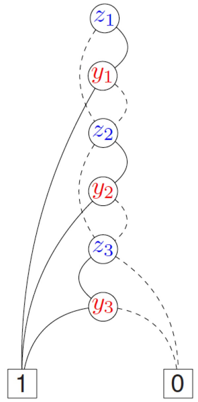
has $3{\cdot}n + 2$ vertices under ordering $x_1 < y_1 < \ldots < x_n < y_n$

Another Exponential Example



ROBDD for $f_3(\overline{z}, \overline{y}) \;=\; (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee (z_3 \wedge y_3)$
for the variable ordering $z_1 < z_2 < z_3 < y_1 < y_2 < y_3$

And An Optimal Linear ROBDD



- ROBDD for $f_3(\cdot) \;=\; (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee (z_3 \wedge y_3)$
- for ordering $z_1 < y_1 < z_2 < y_2 < z_3 < y_3$
- as all variables are essential for f, this ROBDD is optimal
- that is, for no variable ordering a smaller ROBDD exists

Symmetric Functions

Definition: symmetric function

Switching function $f \in Eval(z_1, \ldots, z_m)$ is symmetric if and only if

$$f([z_1 = b_1, \ldots, z_m = b_m]) \;=\; f([z_1 = b_{i_1}, \ldots, z_m = b_{i_m}])$$

for each permutation $(i_1, \ldots, i_m)$ of $(1, \ldots, m)$.

Example symmetric functions: $z_1 \vee z_2 \vee \ldots \vee z_m$, $z_1 \wedge z_2 \wedge \ldots \wedge z_m$, the parity function, and the majority function.

Let f be a symmetric function with m essential variables. Then: for each variable ordering $\wp$, the $\wp$-ROBDD for f has size $O(m^2)$.

Proof.

On the black board. □

The Even Parity Function

Definition: the even parity function

The switching function $f_{even} \in Eval(x_1, \ldots, x_n)$ defined by

$$f_{even}(x_1, \ldots, x_n) = 1 \text{ iff the number of variables } x_i \text{ with value 1 is even}$$

is called the even parity function.

f_{even} has exponential size truth table or propositional formula
but admits an ROBDD of linear size.

The Multiplication Function

- Consider two n-bit integers
  - let $b_{n-1}b_{n-2}\dots b_0$ and $c_{n-1}c_{n-2}\dots c_0$
  - where b_{n-1} is the most significant bit, and b_0 the least significant bit
- Multiplication yields a $2n$-bit integer
  - the ROBDD $\mathfrak{B}_{f_{n-1}}$ has at least 1.09^n vertices
  - where f_{n-1} denotes the $(n-1)$-st output bit of the multiplication

Optimal Variable Ordering

The size of ROBDDs strongly depends on the variable ordering.

The decision problem whether a given variable ordering is optimal is NP-complete.

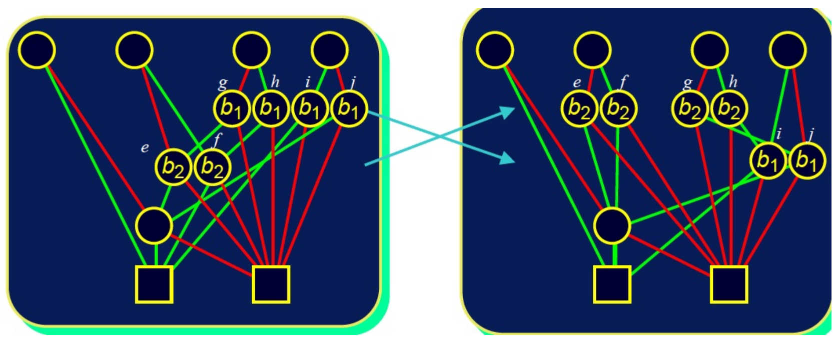
Proof.

Polynomial reduction from the optimal linear arrangement problem.
Rather involved. Outside scope of this lecture. For details, see [Bollig and Wegener, 1996]. □

Variable Ordering

- There are many switching functions with large ROBDDs
  - for almost all switching functions the minimal size is in $\Omega(\frac{2^n}{n})$
  - where n is the number of boolean variables
- How to deal with this problem in practice?
  - guess a variable ordering
  - rearrange the variable ordering during the ROBDD manipulations
  - not necessary to test all $n!$ orderings, best known algorithm in $O(3^n \cdot n^2)$

Variable Swapping



(courtesy: Bryant)

Variable swapping is a local operation only involving two adjacent levels

Variable Sifting [Rudell, 1993]

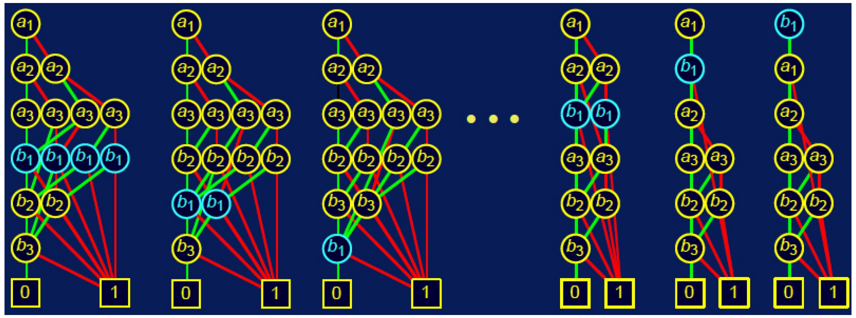
Dynamic variable ordering using repeated variable swapping:

1. Select a variable x_i in the ROBDD
2. Successively swap x_i to determine $size(\mathfrak{B})$ at any position for x_i
3. Shift x_i to position for which $size(\mathfrak{B})$ is minimal
4. Go back to the first step until no improvement is made

Characteristics:

- a variable may change position several times during sifting
- often yields a local optimum, but works well in practice
- in practice, dynamic variable ordering is applied periodically

Sifting



(courtesy: Bryant)

Experimental Results

Circuit	Good		Bad		Bad+Dynamic	
	#nodes	secs	#nodes	secs	#nodes	secs
16-bit rotator	81	<1	1081328	56	81	1
8-bit adder	36	<1	751	<1	36	<1
16-bit adder	76	<1	196575	16	123	1
32-bit adder	156	<1	>1000000	80	452	4
32-bit alu	8869	<1	>1000000	83.4	4341	8.2
64-bit alu	17829	<1	>1000000	81.4	9487	47.2
128-bit alu	35749	1.9	>1000000	79.1	18086	149.6
256-bit alu	71598	4.0	>1000000	82.2	44870	697.9
8-bit Min_Max	890	<1	79007	6	883	3
16-bit Min_Max	3310	<1	>1000000	50	3295	16
32-bit Min_Max	12566	2	>1000000	39	39265	86
12-bit multiplier	605883	255	1324674	340	1494828	2500

[Janssen, 1996] on an HP9000/s755 workstation

Interleaved Variable Ordering

- Which variable ordering to use for transition relations?
- The interleaved variable ordering:

  for encodings $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ of state s and t respectively:

  $$x_1 < y_1 < x_2 < y_2 < \ldots < x_n < y_n$$

- This variable ordering yields compact ROBDDs for binary relations

Overview

Idea

- Take a symbolic representation of a transition system (Δ and χ_B)
- Backward reachability $Pre^*(B) = \{ s \in S \mid s \models \exists \Diamond B \}$
- Initially: $f_0 = \chi_B$ characterizes the set $T_0 = B$
- Then, successively compute the functions $f_{j+1} = \chi_{T_{j+1}}$ for:

$$T_{j+1} = T_j \cup \{ s \in S \mid \exists s' \in S.\, s' \in Post(s) \wedge s' \in T_j \}$$

- Second set is symbolically given by: $\exists \overline{x}'.\, (\underbrace{\Delta(\overline{x}, \overline{x}')}_{s' \in Post(s)} \wedge \underbrace{f_j(\overline{x}')}_{s' \in T_j})$

$f_j(\overline{x}')$ arises from f_j by renaming x_i into their primed copies x_i'

Symbolic Computation of $Sat(\exists(C \cup B))$

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

Symbolic Computation of $Sat(\exists \Box B)$

Compute the largest set $T \subseteq B$ with $Post(t) \cap T \neq \varnothing$ for all $t \in T$

Take $T_0 = B$ and $T_{j+1} = T_j \cap \{ s \in S \mid \exists s' \in S.\, s' \in Post(s) \wedge s' \in T_j \}$

Symbolically this amounts to:

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

This can be efficiently done by ROBDD representations of switching functions

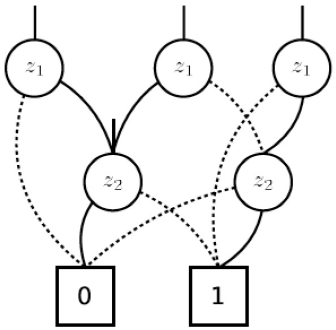
Overview

Synthesis of ROBDDs

- Construct a $\wp$-ROBDD for $f_1 \; op \; f_2$ given $\wp$-ROBDDs for f_1 and f_2
  where op is a Boolean connective such as disjunction, implication, etc.
- Idea: use a single ROBDD with (global) variable ordering $\wp$ to represent several switching functions
- This yields a shared OBDD (SOBDD, for short), which is:
  - a multi-rooted ROBDD
  - a combination of several ROBDDs with variable ordering $\wp$
  - by sharing nodes for common $\wp$-consistent co-factors
- The size of $\wp$-SOBDD $\overline{\mathfrak{B}}$ for functions $f_1, \ldots, f_k$ is at most $N_{f_1} + \ldots + N_{f_k}$ where N_f is the size of the $\wp$-ROBDD for f

Shared OBDDs

- Idea: combine several OBDDs with same variable ordering.
- This enables sharing of common $\wp$-consistent co-factors.
- A shared $\wp$-OBDD is an OBDD with multiple roots.
- It represents multiple switching functions.



Shared OBDD representing $\underbrace{z_1 \wedge \neg z_2}_{f_1}$, $\underbrace{\neg z_2}_{f_2}$, $\underbrace{z_1 \oplus z_2}_{f_3}$ and $\underbrace{\neg z_1 \vee z_2}_{f_4}$

Using Shared OBDDs for CTL Model Checking

Use a single SOBDD to represent for model checking Φ:

- $\Delta(\overline{x}, \overline{x}')$ for the transition relation
  - In practice, often the interleaved variable order for Δ is used.
- $f_a(\overline{x})$, $a \in AP$, for the satisfaction sets of the atomic propositions
- The satisfaction sets $Sat(\Psi)$ for every state sub-formula Ψ of Φ

Synthesizing Shared Reduced OBDDs

Relies on the use of two tables

- The unique table
  - keeps track of ROBDD nodes that already have been created
  - table entry $\langle var(v), succ_1(v), succ_0(v)\rangle$ for each inner node v
  - main operation: *find_or_add*(z, v_1, v_0) with $v_1 \neq v_0$
    - return v if there exists a node $v = \langle z, v_1, v_0\rangle$ in the ROBDD
    - if not, create a new z-node v with $succ_0(v) = v_0$ and $succ_1(v) = v_1$
  - implemented using hash functions (expected access time is $O(1)$)

- The computed table
  - keeps track of tuples for which ITE has been executed (memoisation)
  - $\Rightarrow$ realises a kind of dynamic programming

The ITE Normal Form

The ITE (if-then-else) operator: $ITE(g, f_1, f_2) = (g \wedge f_1) \vee (\neg g \wedge f_2)$.
The representation of the SOBDD nodes in the unique table:

$$f_v = ITE\big(z, f_{succ_1(v)}, f_{succ_0(v)}\big)$$

Then:

$$\begin{aligned} \neg f &= ITE(f, 0, 1) \\ f_1 \vee f_2 &= ITE(f_1, 1, f_2) \\ f_1 \wedge f_2 &= ITE(f_1, f_2, 0) \\ f_1 \oplus f_2 &= ITE(f_1, \neg f_2, f_2) = ITE(f_1, ITE(f_2, 0, 1), f_2) \end{aligned}$$

If g, f_1, f_2 are switching functions for Var, $z \in Var$ and $b \in \{0, 1\}$, then

$$ITE(g, f_1, f_2)|_{z=b} = ITE(g|_{z=b}, f_1|_{z=b}, f_2|_{z=b}).$$

ITE Operator on SOBDDs

- A node in a $\wp$-SOBDD for representing $ITE(g, f_1, f_2)$ is a node w with $info\langle z, w_1, w_0\rangle$ where:
  - z is the minimal (wrt. $\wp$) essential variable of $ITE(g, f_1, f_2)$
  - w_b is an SOBDD-node with $f_{w_b} = ITE(g|_{z=b}, f_1|_{z=b}, f_2|_{z=b})$

- This suggests a recursive algorithm:
  - determine z
  - recursively compute the nodes for ITE for the cofactors of g, f_1 and f_2

$ITE(u, v_1, v_2)$ on SOBDDs (Initial Version)

```
if u is terminal then
  if val(u) = 1 then
    w := v1                                   (* ITE(1, f_v1, f_v2) = f_v1 *)
  else
    w := v2                                   (* ITE(0, f_v1, f_v2) = f_v2 *)
  fi
else
  z := min{var(u), var(v1), var(v2)};         (* minimal essential variable *)
  w1 := ITE(u|z=1, v1|z=1, v2|z=1);
  w0 := ITE(u|z=0, v1|z=0, v2|z=0);
  if w0 = w1 then
    w := w1;                                  (* elimination rule *)
  else
    w := find_or_add(z, w1, w0);              (* isomorphism rule *)
  fi
fi
return w
```

ROBDD Size

The size of the $\wp$-ROBDD for $ITE(g, f_1, f_2)$ is bounded from above by $N_g \cdot N_{f_1} \cdot N_{f_2}$ where N_f denotes the size of the $\wp$-ROBDD for f.

for some ITE-functions optimisations are possible, e.g., $f \oplus g$

Main Deficiency

Problem: for multiple paths from (u, v_1, v_2) to (u', v_1', v_2')

multiple invocations of ITE(u', v_1', v_2') occur.

$\Rightarrow$ Store triples (u, v_1, v_2) for which ITE already has been computed

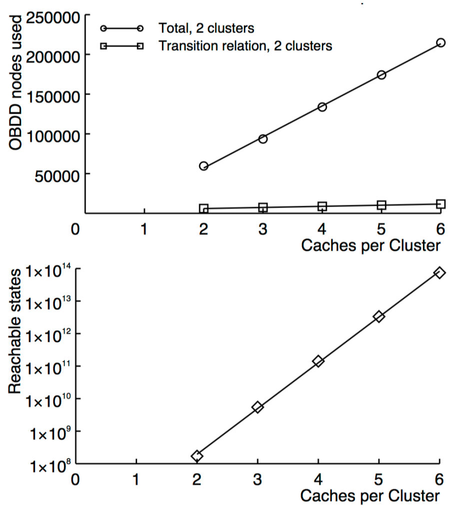
This is similar as in dynamic programming.

$ITE(u, v_1, v_2)$ on SOBDDs Revisited

if there is an entry for (u, v_1, v_2, w) in the computed table **then**
  return node w
else
  if u is terminal **then**
    if $val(u) = 1$ **then** $w := v_1$ **else** $w := v_2$ **fi**
  else
    $z := \min\{var(u), var(v_1), var(v_2)\}$;
    $w_1 := ITE(u|_{z=1}, v_1|_{z=1}, v_2|_{z=1})$;
    $w_0 := ITE(u|_{z=0}, v_1|_{z=0}, v_2|_{z=0})$;
    if $w_0 = w_1$ **then** $w := w_1$ **else** $w := find\_or\_add(z, w_1, w_0)$ **fi**;
    insert (u, v_1, v_2, w) in the computed table;
    return node w
  fi
fi

The number of recursive calls for nodes u, v_1, v_2 equals the $\wp$-ROBDD size of $ITE(f_u, f_{v_1}, f_{v_2})$, which is bounded by $N_u \cdot N_{v_1} \cdot N_{v_2}$

Experimental Results



ROBDD size and state space size for cache coherence protocol [McMillan 1993]

BDD-Based Bisimulation Minimisation

Overview

Summary

- ROBDDs are a succinct data structure for many switching functions
- Crucial factor: the variable ordering
- Transition systems can be easily represented by switching functions
- Symbolic CTL model checking = fixed-point computation with switching functions
  - it is all about using ROBDD representations and manipulating them
- If ROBDD representation is compact, CTL model checking scales well
- Several large companies have in-house symbolic model checkers
  - IBM, Lucent, Intel, Motorola, SGI, Fujitsu, Siemens, . . .

Next —and Final— Lecture

Friday January 17, 14:30