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

1. Motivation
2. The Variable Ordering Problem
3. Symbolic CTL Model Checking
4. Implementation Details
5. Summary

Inventors of BDD-Based Model Checking

Randy Bryant (USA)
Edmund Clarke Jr. (USA)
Ken McMillan (USA)
David Dill (USA)
The Variable Ordering Problem

Overview

1 Motivation
2 The Variable Ordering Problem
3 Symbolic CTL Model Checking
4 Implementation Details
5 Summary

Joost-Pieter Katoen
Lecture#19 5/40

Variable Ordering

- ROBDDs are canonical for a fixed variable ordering
  - the size of the ROBDD crucially depends on the variable ordering
  - $\# \text{ nodes in } \varphi \text{- ROBDD } \mathcal{B} = \# \text{ of } \varphi \text{-consistent co-factors of } f$

- 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 $f(\ldots) = 1 \text{ iff } \geq k \text{ variables } x_i \text{ are true}$

- Some switching functions only have exponential ROBDDs
  - this holds, e.g., for the middle bit of the multiplication function

Joost-Pieter Katoen
Lecture#19 6/40

Function Stable with Exponential ROBDD

The ROBDD of $f_{\text{stab}}(x, y) = (x_1 \leftrightarrow y_1) \land \ldots \land (x_n \leftrightarrow y_n)$
has $3 \cdot 2^n - 1$ vertices under ordering $x_1 < \ldots < x_n < y_1 < \ldots < y_n$

Joost-Pieter Katoen
Lecture#19 7/40

Function Stable with Linear ROBDD

The ROBDD of $f_{\text{stab}}(x, y) = (x_1 \leftrightarrow y_1) \land \ldots \land (x_n \leftrightarrow y_n)$
has $3 \cdot n + 2$ vertices under ordering $x_1 < y_1 < \ldots < x_n < y_n$
The Variable Ordering Problem

Another Exponential Example

The Variable Ordering Problem

Another Exponential Example

The Variable Ordering Problem

And An Optimal Linear ROBDD

The Variable Ordering Problem

The Even Parity Function

The Variable Ordering Problem

Symmetric Functions

Definition: symmetric function

Switching function \( f \in \text{Eval}(z_1, \ldots, z_m) \) 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: \( z_1 \lor z_2 \lor \ldots \lor z_m \), \( z_1 \land z_2 \land \ldots \land z_m \), the parity function, and the majority function.

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

Definition: the even parity function

The switching function \( f_{\text{even}} \in \text{Eval}(x_1, \ldots, x_n) \) defined by
\[
 f_{\text{even}}(x_1, \ldots, x_n) = 1 \text{ iff the number of variables } x_i \text{ with value 1 is even}
\]
is called the even parity function.

\( f_{\text{even}} \) has exponential size truth table or propositional formula
but admits an ROBDD of linear size.
The Variable Ordering Problem

The Multiplication Function

- Consider two $n$-bit integers
  - let $b_{n-1}b_{n-2}\ldots b_0$ and $c_{n-1}c_{n-2}\ldots c_0$
  - where $b_{n-1}$ is the most significant bit, and $b_0$ the least significant bit

- Multiplication yields a $2n$-bit integer
  - the ROBDD $B_{f_{n-1}}$ 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 $\Omega(2^n/n)$
  - 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

Variable swapping is a local operation only involving two adjacent levels (courtesy: Bryant)
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 $\text{size}(\mathcal{B})$ at any position for $x_i$
3. Shift $x_i$ to position for which $\text{size}(\mathcal{B})$ 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

Experimental Results

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Good</th>
<th>Bad</th>
<th>Bad+Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#nodes</td>
<td>secs</td>
<td>#nodes</td>
</tr>
<tr>
<td>16-bit rotator</td>
<td>81</td>
<td>&lt;1</td>
<td>108132</td>
</tr>
<tr>
<td>8-bit adder</td>
<td>36</td>
<td>&lt;1</td>
<td>751</td>
</tr>
<tr>
<td>16-bit adder</td>
<td>76</td>
<td>&lt;1</td>
<td>196575</td>
</tr>
<tr>
<td>32-bit adder</td>
<td>155</td>
<td>&lt;1</td>
<td>&gt;1000000</td>
</tr>
<tr>
<td>32-bit alu</td>
<td>17829</td>
<td>&lt;1</td>
<td>&gt;1000000</td>
</tr>
<tr>
<td>64-bit alu</td>
<td>35749</td>
<td>1.9</td>
<td>&gt;1000000</td>
</tr>
<tr>
<td>128-bit alu</td>
<td>71598</td>
<td>4.0</td>
<td>&gt;1000000</td>
</tr>
<tr>
<td>256-bit alu</td>
<td>890</td>
<td>&lt;1</td>
<td>79007</td>
</tr>
<tr>
<td>8-bit Min_Max</td>
<td>3310</td>
<td>&lt;1</td>
<td>&gt;1000000</td>
</tr>
<tr>
<td>16-bit Min_Max</td>
<td>12566</td>
<td>2</td>
<td>&gt;1000000</td>
</tr>
<tr>
<td>32-bit Min_Max</td>
<td>605883</td>
<td>255</td>
<td>1324674</td>
</tr>
</tbody>
</table>

[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
Symbolic CTL Model Checking

Overview

1 Motivation
2 The Variable Ordering Problem
3 Symbolic CTL Model Checking
4 Implementation Details
5 Summary

Joost-Pieter Katoen
Lecture#19 21/40

Symbolic CTL Model Checking

Idea

▶ Take a symbolic representation of a transition system (Δ and χ_B)

▶ Backward reachability Pre^*(B) = \{ s \in S \mid s \not\models \exists B \} 

▶ Initially: f_0 = χ_B characterizes the set T_0 = B

▶ Then, successively compute the functions f_{j+1} = χ_{T_j+1} for:

\[ T_{j+1} = T_j \cup \{ s \in S \mid \exists s' \in S. s' \in \text{Post}(s) \land s' \in T_j \} \]

▶ Second set is symbolically given by: \( \exists \overline{x}'. (\Delta(\overline{x}, \overline{x}') \land f_j(\overline{x}) \) 

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

Symbolic Computation of \( \text{Sat}(\exists(C \cup B)) \)

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

Symbolic Computation of \( \text{Sat}(\exists \Box B) \)

Compute the largest set \( T \subseteq B \) with \( \text{Post}(t) \cap T \neq \emptyset \) for all \( t \in T \)

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

Symbolically this amounts to:

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

This can be efficiently done by ROBDD representations of switching functions
Overview

1 Motivation
2 The Variable Ordering Problem
3 Symbolic CTL Model Checking
4 Implementation Details
5 Summary

Implementation Details

Synthesis of ROBDDs

- Construct a \( \varphi \)-ROBDD for \( f_1 \ op f_2 \) given \( \varphi \)-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 \( \varphi \) 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 \( \varphi \)
  - by sharing nodes for common \( \varphi \)-consistent co-factors

- The size of \( \varphi \)-SOBDD \( \overline{B} \) 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 \( \varphi \)-ROBDD for \( f \)

Shared OBDDs

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

Using Shared OBDDs for CTL Model Checking

Use a single SOBDD to represent for model checking \( \Phi \):

- \( \Delta(x, x') \) for the transition relation
  - In practice, often the interleaved variable order for \( \Delta \) is used.
- \( f_a(x) \), \( a \in AP \), for the satisfaction sets of the atomic propositions
- The satisfaction sets \( \text{Sat}(\Psi) \) for every state sub-formula \( \Psi \) of \( \Phi \)
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 \((\text{var}(v), \text{succ}_1(v), \text{succ}_0(v))\) for each inner node \(v\)
  - main operation: \(\text{find_or_add}(z, v_1, v_0)\) with \(v_1 \neq v_0\)
  - return \(v\) if there exists a node \(v = (z, v_1, v_0)\) in the ROBDD
  - if not, create a new \(z\)-node \(v\) with \(\text{succ}_0(v) = v_0\) and \(\text{succ}_1(v) = v_1\)
  - implemented using hash functions (expected access time is \(O(1)\))

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

The ITE Normal Form

The ITE (if-then-else) operator: 
\[
\text{ITE}(g, f_1, f_2) = (g \land f_1) \lor (\neg g \land f_2).
\]

The representation of the SOBDD nodes in the unique table:
\[
f_v = \text{ITE}(z, f_{\text{succ}_1(v)}, f_{\text{succ}_0(v)}).
\]

Then:
\[
\neg f = \text{ITE}(f_0, 0, 1)
\]
\[
f_1 \lor f_2 = \text{ITE}(f_1, 1, f_2)
\]
\[
f_1 \land f_2 = \text{ITE}(f_1, f_2, 0)
\]
\[
f_1 \oplus f_2 = \text{ITE}(f_1, \neg f_2, f_2) = \text{ITE}(f_1, \text{ITE}(f_2, 0, 1), f_2)
\]

If \(g, f_1, f_2\) are switching functions for \(\text{Var}\), \(z \in \text{Var}\) and \(b \in \{0, 1\}\), then
\[
\text{ITE}(g, f_1, f_2)|_{z=b} = \text{ITE}(g|_{z=b}, f_1|_{z=b}, f_2|_{z=b}).
\]

ITE Operator on SOBDDs

- A node in a \(\wp\)-SOBDD for representing \(\text{ITE}(g, f_1, f_2)\)
  - is a node \(w\) with \(\text{info}(z, w_1, w_0)\) where:
    - \(z\) is the minimal (wrt. \(\wp\)) essential variable of \(\text{ITE}(g, f_1, f_2)\)
    - \(w_0\) is an SOBDD-node with \(f_{w_0} = \text{ITE}(g|_{z=b}, f_1|_{z=b}, f_2|_{z=b})\)
  - This suggests a recursive algorithm:
    - determine \(z\)
    - recursively compute the nodes for ITE for the cofactors of \(g, f_1\) and \(f_2\)
ROBDD Size

The size of the \( \mathcal{P} \)-ROBDD for \( \text{ITE}(g, f_1, f_2) \) is bounded from above by \( N_g \cdot N_{f_1} \cdot N_{f_2} \) where \( N_f \) denotes the size of the \( \mathcal{P} \)-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 \( \text{ITE}(u', v'_1, v'_2) \) occur.

\[ \Rightarrow \] Store triples \((u, v_1, v_2)\) for which \( \text{ITE} \) 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 \(\text{val}(u) = 1\) then \(w := v_1\) else \(w := v_2\)
  - else
    - \(z := \min\{\text{var}(u), \text{var}(v_1), \text{var}(v_2)\}\)
    - \(w_1 := \text{ITE}(u|_{z=1}, v_1|_{z=1}, v_2|_{z=1})\)
    - \(w_0 := \text{ITE}(u|_{z=0}, v_1|_{z=0}, v_2|_{z=0})\)
    - if \(w_0 = w_1\) then \(w := w_1\) else \(w := \text{find_or_add}(z, w_1, w_0)\)
    - insert \((u, v_1, v_2, w)\) in the computed table;
    - return node \(w\)

The number of recursive calls for nodes \(u, v_1, v_2\) equals the \( \mathcal{P} \)-ROBDD size of \( \text{ITE}(f_1, f_2, f_3) \), 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

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, ...