Binary Decision Diagrams
and
Symbolic Model Checking

Randy Bryant    CMU
Ed Clarke       CMU
Ken McMillan    Cadence
Allen Emerson   U Texas

http://www.cs.cmu.edu/~bryant
Binary Decision Diagrams

Restricted Form of Branching Program
- Graph representation of Boolean function
- Canonical form
- Simple algorithms to construct & manipulate

Application Niche
- Problems expressed as Quantified Boolean Formulas
- A lot of interesting problems are in PSPACE

Symbolic Model Checking
- Prove properties about large-scale, finite-state system
- Successfully used to verify hardware systems
View $n$-variable Boolean function as language $\subseteq \{0,1\}^n$.

Reduced DFA is canonical representation.
From DFA to OBDD

Canonical representation of Boolean function

- Two functions equivalent if and only if graphs isomorphic
- Desirable property: *simplest form is canonical.*
Representing Circuit Functions

Functions

- All outputs of 4-bit adder
- Functions of data inputs

Shared Representation

- Graph with multiple roots
- 31 nodes for 4-bit adder
- 571 nodes for 64-bit adder

*Linear growth*
Effect of Variable Ordering

\[(a_1 \land b_1) \lor (a_2 \land b_2) \lor (a_3 \land b_3)\]

Good Ordering

Bad Ordering

Linear Growth

Exponential Growth
Sample Function Classes

<table>
<thead>
<tr>
<th>Function Class</th>
<th>Best</th>
<th>Worst</th>
<th>Ordering Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU (Add/Sub)</td>
<td>linear</td>
<td>exponential</td>
<td>High</td>
</tr>
<tr>
<td>Symmetric</td>
<td>linear</td>
<td>quadratic</td>
<td>None</td>
</tr>
<tr>
<td>Multiplication</td>
<td>exponential</td>
<td>exponential</td>
<td>Low</td>
</tr>
</tbody>
</table>

General Experience

- Many tasks have reasonable OBDD representations
- Algorithms remain practical for up to 500,000 node OBDDs
- Heuristic ordering methods generally satisfactory
Symbolic Manipulation with OBDDs

Strategy

- Represent data as set of OBDDs
  - Identical variable orderings
- Express solution method as sequence of symbolic operations
  - Sequence of constructor & query operations
  - Similar style to on-line algorithm
- Implement each operation by OBDD manipulation
  - Do all the work in the constructor operations

Key Algorithmic Properties

- Arguments are OBDDs with identical variable orderings
- Result is OBDD with same ordering
- Each step polynomial complexity
If-Then-Else Operation

**Concept**
- Basic technique for building OBDD from logic network or formula.

**Arguments** $I, T, E$
- Functions over variables $X$
- Represented as OBDDs

**Result**
- OBDD representing composite function
  - $(I \land T) \lor (\neg I \land E)$
If-Then-Else Execution Example

Optimizations
- Dynamic programming
- Early termination rules
If-Then-Else Result Generation

- Recursive calling structure implicitly defines unreduced BDD
- Apply reduction rules bottom-up as return from recursive calls
Restriction Operation

Concept

- Effect of setting function argument $x_i$ to constant $k$ (0 or 1).
- Also called Cofactor operation (UCB)

$$F_X \text{ equivalent to } F [x = 1]$$
$$F_X^c \text{ equivalent to } F [x = 0]$$

![Diagram of Restriction Operation](image)
Restriction Execution Example

Argument $F$

Restriction $F[b=1]$

Reduced Result

![Diagram of Argument $F$]

![Diagram of Restriction $F[b=1]$]

![Diagram of Reduced Result]
Other operations can be expressed in terms of If-Then-Else

\[
\text{And}(F, G) = \begin{cases} 
F & \text{if } X = 1 \\
G & \text{if } X = 0 
\end{cases}
\]

\[
\text{If-Then-Else}(F, G, 0) = \begin{cases} 
1 & \text{if } X = 1 \\
0 & \text{if } X = 0 
\end{cases}
\]

\[
\text{Or}(F, G) = \begin{cases} 
F & \text{if } X = 1 \\
G \rightarrow 1 & \text{if } X = 0 
\end{cases}
\]

\[
\text{If-Then-Else}(F, 1, G) = \begin{cases} 
1 & \text{if } X = 1 \\
0 & \text{if } X = 0 
\end{cases}
\]
Generating OBDD from Network

Task: Represent output functions of gate network as OBDDs.

```plaintext
Network
A ─── T1
B ─── T2
C ─── Out

Evaluation
A ← new_var ("a");
B ← new_var ("b");
C ← new_var ("c");
T1 ← And (A, 0, B);
T2 ← And (B, C);
Out ← Or (T1, T2);
```

Resulting Graphs

---

- A: 0 1 0
- B: 1 0 1
- C: 1 0 0
- T1: 1 0 1
- T2: 0 1 0
- Out: 1

---
Create new function by composing functions F and G.
Useful for composing hierarchical modules.
Variable Quantification

- Eliminate dependency on some argument through quantification
- Combine with AND for universal quantification.
Finite State System Analysis

**Systems Represented as Finite State Machines**
- Sequential circuits
- Communication protocols
- Synchronization programs

**Analysis Tasks**
- State reachability
- State machine comparison
- Temporal logic model checking

**Traditional Methods Impractical for Large Machines**
- Polynomial in number of states
- Number of states exponential in number of state variables.
- Example: single 32-bit register has 4,294,967,296 states!
Temporal Logic Model Checking

Verify Reactive Systems

- Construct state machine representation of reactive system
  - Nondeterminism expresses range of possible behaviors
  - “Product” of component state machines
- Express desired behavior as formula in temporal logic
- Determine whether or not property holds

Traffic Light Controller Design

“It is never possible to have a green light for both N-S and E-W.”

Model Checker

True

False + Counterexample
Characteristic Functions

Concept

- $A \subseteq \{0,1\}^n$
  - Set of bit vectors of length $n$
- **Represent set $A$ as Boolean function $A$ of $n$ variables**
  - $X \in A$ if and only if $A(X) = 1$

Set Operations

- **Union**

- **Intersection**
Symbolic FSM Representation

Nondeterministic FSM

Symbolic Representation

- Represent set of transitions as function \( \delta(\text{Old, New}) \)
  - Yields 1 if can have transition from state \( \text{Old} \) to state \( \text{New} \)

- Represent as Boolean function
  - Over variables encoding states

\( o_1, o_2 \) encoded
\( n_1, n_2 \) encoded
old state
new state
Reachability Analysis

Task
- Compute set of states reachable from initial state $Q_0$
- Represent as Boolean function $R(S)$
- Never enumerate states explicitly

Given

Compute

Initial
Breadth-First Reachability Analysis

- $R_i$ – set of states that can be reached in $i$ transitions
- Reach fixed point when $R_n = R_{n+1}$
  - Guaranteed since finite state
Iterative Computation

- $R_{i+1}$ – set of states that can be reached $i+1$ transitions
  - Either in $R_i$
  - or single transition away from some element of $R_i$
Symbolic FSM Analysis Example

- K. McMillan, E. Clarke (CMU)  J. Schwalbe (Encore Computer)

**Encore Gigamax Cache System**
- Distributed memory multiprocessor
- Cache system to improve access time
- Complex hardware and synchronization protocol.

**Verification**
- Create “simplified” finite state model of system ($10^9$ states!)
- Verify properties about set of reachable states

**Bug Detected**
- Sequence of 13 bus events leading to deadlock
- With random simulations, would require $\approx 2$ years to generate failing case.
- In real system, would yield MTBF < 1 day.
System Modeling Example

Gigamax Memory System

Global Bus

Interface

Cluster #2 Abstraction

Cluster #3 Abstraction

Simplifying Abstractions

- Single word cache
- Single bit/word
- Abstract other clusters
- Imprecise timing

Mem.

Cache Control.

Cache Control.

Proc.

Proc.

Arbitrary reads & writes
Commercial Applications of Symbolic Model Checking

Several Commercial Tools
- Difficult training and customer support

Most Large Companies Have In-House Versions
- IBM, Lucent, Intel, Motorola, SGI, Fujitsu, Siemens, …
- Many based on McMillan’s SMV program

Requires Sophistication
- Beyond that of mainstream designers
Application Challenge

Challenging Systems to Design

Model checking Capacity

Degree of Concurrency

Cannot Apply Directly to Full Scale Design

- Verify smaller subsystems
- Verify abstracted versions of full system
  - Must understand system & tool to do effectively
Real World Issues

Still Too Volatile

- Fail by running out of space
- Useless once exceed physical memory capacity

Ongoing Research to Improve Memory Performance

- Dynamic variable ordering
- Exploiting modularity of system model
  - Partitioned transition relations
- Exploiting parallelism
  - Map onto multiple machines
  - Difficult program for parallel computation
    - Dynamic, irregular data structures
Dynamic Variable Reordering

- Richard Rudell, Synopsys

**Periodically Attempt to Improve Ordering for All BDDs**
- Part of garbage collection
- Move each variable through ordering to find its best location

**Has Proved Very Successful**
- Time consuming but effective
- Especially for sequential circuit analysis
Dynamic Reordering By Sifting

- Choose candidate variable
- Try all positions in variable ordering
  - Repeatedly swap with adjacent variable
- Move to best position found

Best Choices
Localized Effect

- Add / delete / alter only nodes labeled by swapping variables
- Do not change any incoming pointers
Tuning of BDD Packages

Cooperative Effort

- Bwolen Yang, in cooperation with researchers from Colorado, Synopsys, CMU, and T.U. Eindhoven
- Measure & improve performance of BDDs for symbolic model checking

Methodology

- Generated set of benchmark traces
- Run 6 different packages on same machine
- Compare results and share findings
  - Cooperative competition
Effect of Optimizations

Compare pre- vs. post-optimized results for 96 runs

- 6 different BDD packages
- 16 benchmark traces each
- Limit each run to maximum of 8 CPU hours and 900 MB
- Measure speedup = $T_{old} / T_{new}$ or:
  - New: Failed before but now succeeds
  - Fail: Fail both times
  - Bad: Succeeded before, but now fails
Optimization Results Summary

Cumulative Speedup Histogram

# of cases

speedups

>100 10 5 2 0.95 0 >5 >1 >0 new failed bad

6 22 33 61 75 76 76 13 6 1
What’s Good about OBDDs

**Powerful Operations**
- Creating, manipulating, testing
- Each step polynomial complexity
  - Graceful degradation

**Generally Stay Small Enough**
- Especially for digital circuit applications
- Given good choice of variable ordering

**Weak Competition**
- No other method comes close in overall strength
- Especially with quantification operations
Thoughts on Algorithms Research

Need to be Willing to Attack Intractable Problems

- Many real-world problems NP-hard
- No approximations for verification

Who Works on These?

- Mostly people in application domain
  - Most work on BDDs in computer-aided design conferences
- Not by people with greatest talent in algorithms
  - No papers in STOC/FOCS/SODA
  - Probably many ways they could improve things
- Fundamental dilemma
  - Can only make weak formal statements about efficiency
  - Utility demonstrated empirically