Symbolic Encoding using Decision Diagrams

16.412J/6.834J Cognitive Robotics

Martin Sachenbacher
(using material from Randal Bryant, Alan Mishchenko, and Geert Janssen)

The State Explosion Problem

- Many problems suffer from state space explosion: the number of states is exponential in the number of variables in the system.
- Decision diagrams (DDs): graph-based, canonical representation of functions that avoids space explosion in many practical cases.

CiteSeer Database

Overview

- Binary Decision Diagrams (BDDs)
- Extensions: ADDs, MDDs, ZDDs, SBDDs
- Decision Diagram Packages
- Combining Symbolic Encoding and Search

Boolean Functions

- Binary Variables $x_1, ..., x_n$
- Function $f: x_1, ..., x_n \rightarrow \{0,1\}$

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$f(x,y,z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sum of Products (DNF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f = (\neg x\land y\land z) \lor (x\land \neg y\land \neg z) \lor (x\land y\land \neg z)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product of Sums (CNF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f = (x\lor y\lor z) \land (x\lor \neg y\lor \neg z) \land (x\lor \neg y\lor z) \land (\neg x\lor \neg y\lor \neg z)$</td>
</tr>
</tbody>
</table>

Shannon Expansion

- Consider boolean function $f(x_1, x_2, ..., x_n)$. Then:

$$f = (\neg x_1 \land f|_{x_1=0}) \lor (x_1 \land f|_{x_1=1})$$

Cofactor $x_1=0$ Cofactor $x_1=1$
**Binary Decision Trees**
- Recursive Shannon expansion

```
  x
 / \0  1
 y  z
 0  1 0 1
 z  y  y
 0  1 0 1
 z  y  y
 0  1 0 1
 z  y  y
 0  1 0 1
```

**Variable Ordering**
- Impose arbitrary total ordering on variables
- Variables must obey ordering along all paths
- Property: No conflicting assignments along path

```
x  x  x
  y  z  x
```

**Ordered Binary Decision Tree**
- Order $x < y < z$

```
  x
 / \0  1
 y  z
 0  1 0 1
 z  y  y
 0  1 0 1
 z  y  y
 0  1 0 1
 z  y  y
 0  1 0 1
```

**Avoiding Blow-Up**
- Decision tree no more space-efficient than truth table
  - Still explicitly enumerates all the $2^n$ possible valuations
- Idea: Transform to **directed acyclic graph (DAG)**
  - Promotes sharing of sub-expressions

**Rule 1: Collapse Leaf Nodes**

```
  x
 / \0  1
 y  z
 0  1 0 1
 z  y  y
 0  1 0 1
 z  y  y
 0  1 0 1
 z  y  y
 0  1 0 1
```

No longer a tree.
Rule 2: Remove Redundant Tests

Rule 3: Isomorphic Subgraphs

Rule may become applicable again

Final Representation

Reduced, Ordered Binary Decision Diagram (ROBDD)
**OBDT to ROBDD Summary**

An ROBDD can be obtained from an OBDT by repeatedly applying the following reduction rules (until none of the them can be applied anymore):
- Remove duplicate terminal (leaf) nodes
- Remove duplicate non-terminal (internal) nodes
- Remove nodes with redundant tests

**Canoncity of ROBDDs**

- Is there a unique DD for each boolean function?
- Theorem: DD canonical, iff reduced and ordered.
  - Reduced: None of the reductions applicable
  - Ordered: There is a total variable ordering
- Enables checking equivalence of boolean functions by checking equivalence of corresponding ROBDDs.

**Equivalence Checking Example**

- Do two circuits compute an identical function?
  - Basic task in formal hardware verification
  - Compare new design to known “good” design

**Solution by Combinatorial Search**

- Prove all assignments fail to check equivalence
- Typically, must explore significant fraction of inputs
- Exponential time complexity

**Solution using ROBDDs**

- Functions equal iff ROBDDs identical
  - Never enumerate explicit function values
  - Exploit structure and regularity of functions

**Example ROBDDs**

- 4 bit adder
  - 31 nodes
- 64 bit adder
  - 571 nodes
- ...
Influence of Variable Ordering

- Function $f = (a_1 \land b_1) \lor (a_2 \land b_2) \lor (a_3 \land b_3)$

Good ordering: Linear growth

Bad ordering: Exponential growth

Variable Ordering

- ROBDD size depends strongly on variable ordering
- Finding variable ordering that produces minimal ROBDD size is intractable
- But, heuristics exist for finding good variable orderings
- Functions exist whose ROBDD has exponential size for any variable ordering
- But, such functions are rarely encountered

Manipulating ROBDDs

- Procedure $\text{Apply}(f,g)$ for implementing all the $2^4 = 16$ two-argument logical operations on boolean functions

$$\text{Apply}(f,g) =$$

$$(\neg x \land \text{Apply}(f_{x=0}, g_{x=0})) \lor (x \land \text{Apply}(f_{x=1}, g_{x=1}))$$

Recursive descent into subtrees $x=0$

Recursive descent into subtrees $x=1$

Apply Operator

- Three possible cases

Apply Operator Pseudocode

```pseudocode
Function Apply(F, G)
    if (AlreadyComputed(F, G)) return the result
    elsif (F ∈ {0,1} and G ∈ {0,1}) return oper(F, G)
    elsif (Var(F) = Var(G))
        u ← CreateNode(Var(F), Apply(Fx', Gx'), Apply(Fx, Gx));
    elsif (Var(F) < Var(G))
        u ← CreateNode(Var(F), Apply(Fx', G), Apply(Fx, G));
    else /* (Var(F) > Var(G)) */
        u ← CreateNode(Var(G), Apply(F, Gx'), Apply(F, Gx));
    InsertComputed(F, G, u);
    return u;

Complexity: O(|F|\ast|G|)
```

Apply Operator Example
Apply Operator Example

```
A1, B1
A2, B2
A3, B3
A4, B4
```

Generating ROBDDs Incrementally

```
A ← new_var("a")
B ← new_var("b")
C ← new_var("c")
T1 ← And(A, B)
T2 ← And(B, C)
O1 ← Or(T1, T2)
```

Decision Diagrams

ROBDDs Summary

- Exploit structure and regularities of functions
- Exponential number of discrete states can be captured in polynomial-size ROBDD
- Satisfiability, Tautology, Complement constant
- Apply (and, or, etc.) polynomial in ROBDD size
- Variable ordering important, but difficult to find

Overview

- Binary Decision Diagrams (BDDs)
- Extensions: ADDs, MDDs, ZDDs, SBDDs
- Decision Diagram Packages
- Combining Symbolic Encoding and Search

Algebraic DDs (ADDs) [Bahar 93]

- Extension to functions with non-binary values
- Canonicity of representation (as for BDDs)
- Applications in combinatorial optimization

```
f(x,y,z)
0 0 0 3
0 1 0 2
0 0 1 2
0 1 1 1
```

Multi-Valued DDs (MDDs) [Kam 90]

- Extension to functions with non-binary variables
- Canonicity of representation (as for ROBDDs)
- However, binary encoding of domains often better
Zero-suppressed DDs (ZDDs) [Minato 93]
- Adapted to sparse functions (many 0's in on-set)
- Modified rule 2: Remove node if 1-edge points to 0

Shared BDDs (SBDDs) [Minato 90]
- Global table storing unique nodes
- Equivalence check for functions becomes constant

Overview
- Binary Decision Diagrams (BDDs)
- Extensions: ADDs, MDDs, ZDDs, SBDDs
- Decision Diagram Packages
- Combining Symbolic Encoding and Search

Decision Diagram Packages

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Author(s)</th>
<th>Affiliation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>ABCD 0.3</td>
<td>Armin Biere</td>
<td>ETH Zurich</td>
</tr>
<tr>
<td>BUD</td>
<td>BuDDy 1.9</td>
<td>Jørn Lind-Nielsen</td>
<td>ITU, Denmark</td>
</tr>
<tr>
<td>CAL</td>
<td>CAL</td>
<td>Rajeev Ranjan</td>
<td>UC Berkeley</td>
</tr>
<tr>
<td>CMU</td>
<td>CMU</td>
<td>David Long</td>
<td>CMU/ATT</td>
</tr>
<tr>
<td>CUD</td>
<td>CUDD 2.3.1</td>
<td>Fabio Somerzi</td>
<td>Boulder, CO</td>
</tr>
<tr>
<td>EST</td>
<td>EST 1ed</td>
<td>Robert Medlic</td>
<td>Maribor, Slovenia</td>
</tr>
<tr>
<td>IBM</td>
<td>IBM</td>
<td>Geert Janssen</td>
<td>IBM Watson</td>
</tr>
<tr>
<td>MON</td>
<td>MONA</td>
<td>Anders Müller</td>
<td>ATT/BRICS</td>
</tr>
<tr>
<td>PDT</td>
<td>PDT</td>
<td>Cusnato/Corno</td>
<td>Politecnico di Torino</td>
</tr>
<tr>
<td>STA</td>
<td>StaticBdd 1.0</td>
<td>Stefan Edelkamp</td>
<td>Freiburg, Germany</td>
</tr>
<tr>
<td>TGR</td>
<td>TGeF 3.0?</td>
<td>Coudert/Madre/Touati</td>
<td>Bull/DEC/Xorix</td>
</tr>
<tr>
<td>TUD</td>
<td>TUDD 0.8.3a</td>
<td>Stefan Höreth</td>
<td>Darmstadt, Germany</td>
</tr>
</tbody>
</table>

Package Characteristics

<table>
<thead>
<tr>
<th>Code</th>
<th>Name of the package</th>
<th>Vmax</th>
<th>P</th>
<th>GC</th>
<th>CT</th>
<th>DV</th>
<th>Ext</th>
<th>Year</th>
<th>Pro</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Name of the package</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Name of the package</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: [Janssen 2002]
Comparing Packages

<table>
<thead>
<tr>
<th>Code</th>
<th>Lang</th>
<th>TR</th>
<th>M</th>
<th>Mmax</th>
<th>Siz</th>
<th>Vmax</th>
<th>P</th>
<th>GC</th>
<th>CT</th>
<th>DV</th>
<th>Ext</th>
<th>Year</th>
<th>Pro</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>C</td>
<td>I</td>
<td>Y</td>
<td>Y</td>
<td>2^25</td>
<td>8</td>
<td>2^10</td>
<td>N</td>
<td>MS</td>
<td>S</td>
<td>N</td>
<td>44</td>
<td>1997</td>
</tr>
<tr>
<td>BUD</td>
<td>C/C++</td>
<td>I</td>
<td>N</td>
<td>N</td>
<td>2^32</td>
<td>20</td>
<td>2^21</td>
<td>N</td>
<td>MS</td>
<td>M</td>
<td>Y</td>
<td>116</td>
<td>1996</td>
</tr>
<tr>
<td>CAL</td>
<td>C</td>
<td>P</td>
<td>B</td>
<td>Y</td>
<td>2^28</td>
<td>16</td>
<td>2^16</td>
<td>Y</td>
<td>RC</td>
<td>N</td>
<td>Y</td>
<td>237</td>
<td>1994</td>
</tr>
<tr>
<td>CMU</td>
<td>C</td>
<td>P</td>
<td>Y</td>
<td>Y</td>
<td>2^29</td>
<td>16</td>
<td>2^16</td>
<td>Y</td>
<td>RC</td>
<td>Y</td>
<td>Y</td>
<td>75</td>
<td>1993</td>
</tr>
<tr>
<td>CUD</td>
<td>C/C++</td>
<td>P</td>
<td>Y</td>
<td>Y</td>
<td>2^28</td>
<td>16</td>
<td>2^16</td>
<td>N</td>
<td>RC</td>
<td>Y</td>
<td>Y</td>
<td>287</td>
<td>1996</td>
</tr>
<tr>
<td>EST</td>
<td>C</td>
<td>P</td>
<td>D</td>
<td>Y</td>
<td>2^32</td>
<td>36</td>
<td>2^32</td>
<td>N</td>
<td>RC</td>
<td>M</td>
<td>N</td>
<td>72</td>
<td>1999</td>
</tr>
<tr>
<td>BM</td>
<td>C</td>
<td>I</td>
<td>Y</td>
<td>Y</td>
<td>2^30</td>
<td>16</td>
<td>2^24</td>
<td>N</td>
<td>MS</td>
<td>Y</td>
<td>Y</td>
<td>169</td>
<td>1999</td>
</tr>
<tr>
<td>MON</td>
<td>C</td>
<td>I</td>
<td>D</td>
<td>Y</td>
<td>2^24</td>
<td>16</td>
<td>2^16</td>
<td>N</td>
<td>MS</td>
<td>M</td>
<td>N</td>
<td>40</td>
<td>1997</td>
</tr>
<tr>
<td>PRT</td>
<td>C</td>
<td>I</td>
<td>N</td>
<td>N</td>
<td>2^31</td>
<td>24</td>
<td>2^16</td>
<td>N</td>
<td>MS</td>
<td>M</td>
<td>N</td>
<td>34</td>
<td>1993</td>
</tr>
<tr>
<td>STA</td>
<td>C++</td>
<td>I</td>
<td>N</td>
<td>N</td>
<td>2^23</td>
<td>8</td>
<td>2^9</td>
<td>N</td>
<td>MS</td>
<td>S</td>
<td>N</td>
<td>66</td>
<td>1999</td>
</tr>
<tr>
<td>TGR</td>
<td>C</td>
<td>P</td>
<td>Y</td>
<td>Y</td>
<td>2^30</td>
<td>16</td>
<td>2^16</td>
<td>Y</td>
<td>RC</td>
<td>Y</td>
<td>Y</td>
<td>90</td>
<td>1994</td>
</tr>
<tr>
<td>TUD</td>
<td>C</td>
<td>P</td>
<td>Y</td>
<td>Y</td>
<td>2^30</td>
<td>20</td>
<td>2^16</td>
<td>Y</td>
<td>RC</td>
<td>Y</td>
<td>Y</td>
<td>44</td>
<td>1998</td>
</tr>
</tbody>
</table>

Source: [Janssen 2002]

Overview

- Binary Decision Diagrams (BDDs)
- Extensions: ADDs, MDDs, ZDDs, SBDDs
- Decision Diagram Packages
- Combining Symbolic Encoding and Search

Branch-and-Bound Search

- Each search node is a soft constraint subproblem
- Lower Bound (lb): Optimistic estimate of best solution in subtree
- Upper Bound (ub): Best solution found so far
- Prune, if lb ≥ ub.

Symbolic Branch-and-Bound

- Each search node is a set of soft constraint subproblems
- Lower Bound Function (f_{lb}): Optimistic estimates of best solutions in subtree
- Upper Bound (ub): Best solution found so far
- Prune, if f_{lb} ≥ ub.

Domain Splitting

Generalize to search over sets:
- Partition domains into sets \( P_i \), \( \cup_{i=1} P_i = d_i \)
- Choose subset \( p \in P_i \) for unassigned variable \( x_i \)
- Combine all completely assigned constraints

Example: 4-Queens

- Variables: Rows \( x_1, x_2, x_3, x_4 \)
- Domains: Columns \( 1, 2, 3, 4 \)
- Constraints: \( c_{12}(x_1, x_2), c_{13}(x_1, x_3), c_{14}(x_1, x_4), \)
  \( c_{23}(x_2, x_3), c_{24}(x_2, x_4), c_{34}(x_3, x_4) \)

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & Q & \_ & \_ \\
2 & 4 & x_3 & Q \\
3 & 1 & \_ & \_ \\
4 & 1 & x_4 & Q \\
\end{array}
\]
Example

- Search Tree

Example

- Domain splitting with partitions \( P_i = \{1, 2\}, \{3, 4\} \)

Example

- Domain splitting with partitions \( P_i = \{1, 2\}, \{3, 4\} \)

Sinking Operation

- sink(\( c_{ij}, \alpha \)) is a new constraint where all values of tuples \( \geq \alpha \) have been replaced by \( \top \)
- Generalizes the test \( \not{b} < u_b \) to functions

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Constraint sink(( c_{ij}, 0.05 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G ) 0</td>
<td>( G ) 0 0.96</td>
</tr>
<tr>
<td>( B ) 0</td>
<td>( B ) 0 0.7</td>
</tr>
<tr>
<td>( N ) 1</td>
<td>( N ) 1 0.2</td>
</tr>
</tbody>
</table>

Symbolic Branch-and-Bound

- Function \( \text{SDFBB}(f_i; \text{assignments}, u_b : \text{value}); \text{value} \)
  
  \( f_b \leftarrow \text{lb}(f_i) \)
  \( f_t \leftarrow \text{sink}(f_b, u_b) \)
  
  Distance lower bound:
  \( \text{lb} = \text{identity}. \)

  if \( f_t \neq \top \) then
  
  if \( \text{var}(f_i) = u \) then return \( f_t \) \( \frac{1}{2} \)

  let \( x_i \) be an unassigned variable

  for each \( p \in \mathcal{P} \) do
    \( u_b \leftarrow \min(u_b, \text{SDFBB}(f_i \oplus (x_i \in p), u_b)) \)

  return \( u_b \)

  Encode functions \( f_i, f_b \)
  as decision diagrams.