Symbolic Encoding using Decision Diagrams

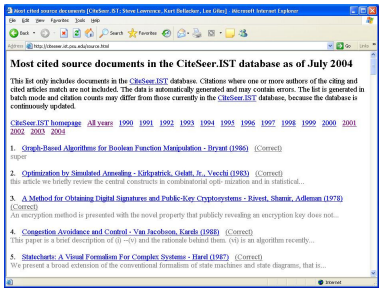
16.412J/6.834J Cognitive Robotics

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 $x_1, \ldots, x_n$
- Function f: $x_1, \ldots, x_n \rightarrow \{0,1\}$

Sum of Products (DNF)

$f = (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge z)$

Product of Sums (CNF)

$f = (x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z)$

Truth Table

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

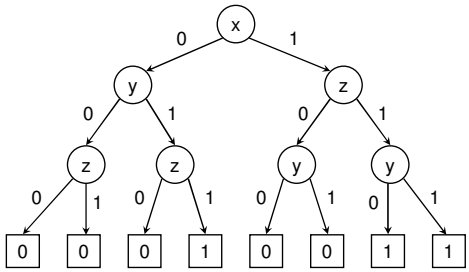
Shannon Expansion

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

$$f = (\neg x_1 \wedge \underbrace{f|_{x1=0}}_{\text{Cofactor } x_1=0}) \vee (x_1 \wedge \underbrace{f|_{x1=1}}_{\text{Cofactor } x_1=1})$$

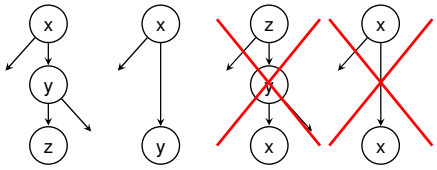
Binary Decision Trees

- Recursive Shannon expansion



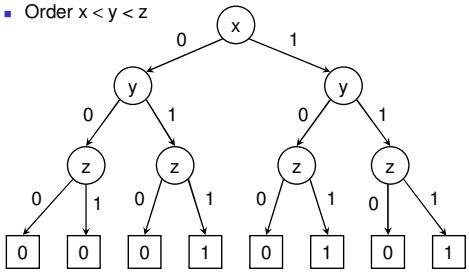
Variable Ordering

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



Ordered Binary Decision Tree

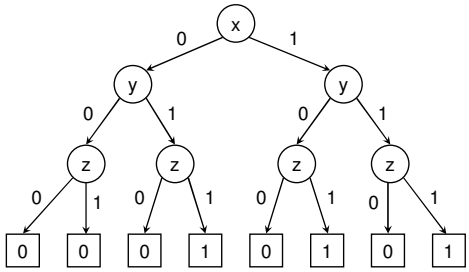
- Order x < y < z



Avoiding Blow-Up

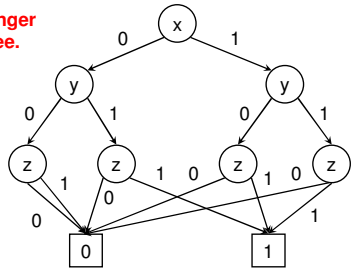
- Decision tree no more space-efficient than truth table
  - Still explicitly enumerates all the 2^n possible valuations
- Idea: Transform to **directed acyclic graph** (DAG)
  - Promotes sharing of sub-expressions

Ordered Binary Decision Tree

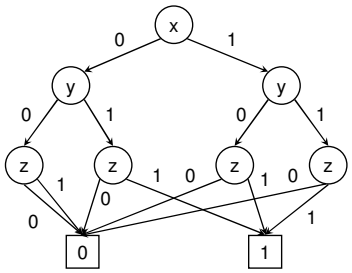


Rule 1: Collapse Leaf Nodes

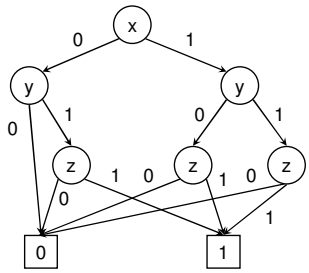
No longer a tree.



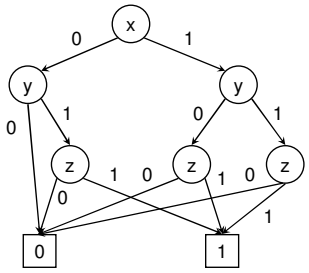
Rule 2: Remove Redundant Tests



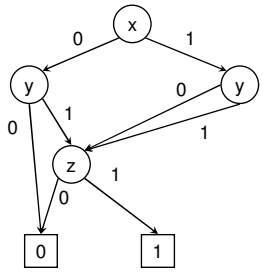
Rule 2: Remove Redundant Tests



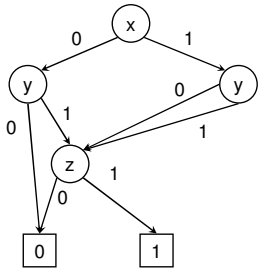
Rule 3: Isomorphic Subgraphs



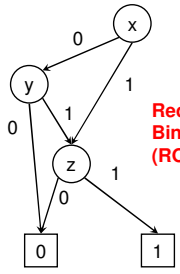
Rule 3: Isomorphic Subgraphs



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):

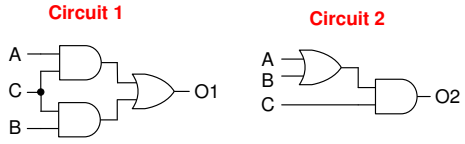
- Remove **duplicate terminal** (leaf) nodes
- Remove **duplicate non-terminal** (internal) nodes
- Remove nodes with **redundant tests**

Canonicity 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.

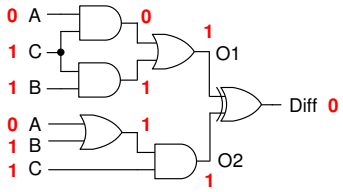
Equivalence Checking Example

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



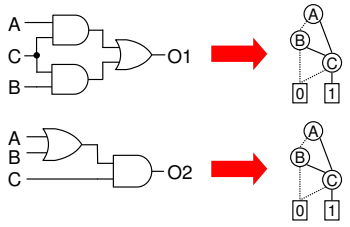
Solution by Combinatorial Search

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



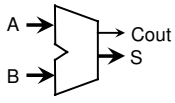
Solution using ROBDDs

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

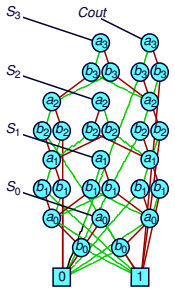


Example ROBDDs

- 4 bit adder
  - 31 nodes
- 64 bit adder
  - 571 nodes
- …

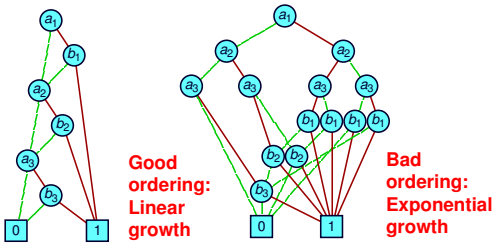


Linear growth



Influence of Variable Ordering

- Function f = (a1 ∧ b1) ∨ (a2 ∧ b2) ∨ (a3 ∧ b3)



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 ☺

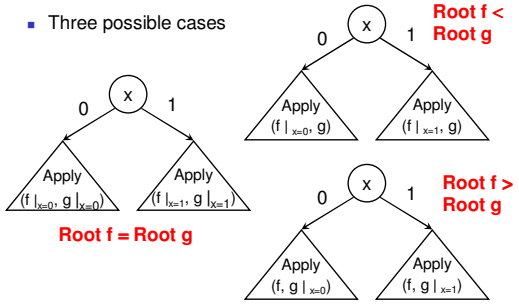
Manipulating ROBDDs

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

$$\text{Apply}(f,g) = (\neg x \wedge \underbrace{\text{Apply}(f|_{x=0}, g|_{x=0})}_{\text{Recursive descent into subtrees x=0}}) \vee (x \wedge \underbrace{\text{Apply}(f|_{x=1}, g|_{x=1})}_{\text{Recursive descent into subtrees x=1}})$$

Apply Operator

- Three possible cases

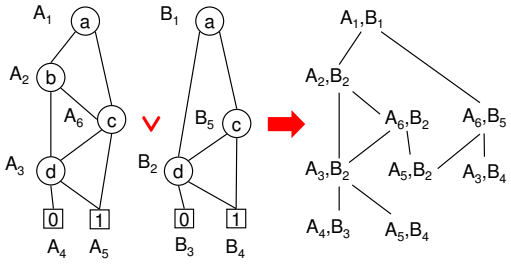


Apply Operator Pseudocode

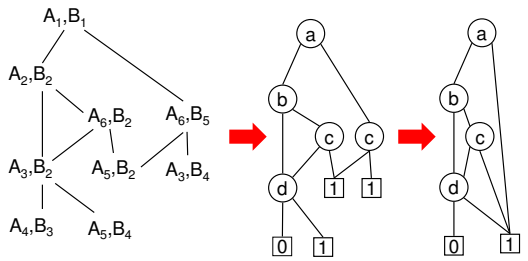
```
Function Apply( F, G )
  if ( AlreadyComputed( F, G ) ) return the result
  elsif ( F∈{0,1} and G∈{0,1} ) return oper( F, G )
  elsif ( Var( F ) = Var( G ) )
    u ← CreateNode( Var(F), Apply(Fx',Gx'), Apply(Fx,Gx));
  elsif ( Var( F ) < Var( G ) )
    u ← CreateNode( Var(F) , Apply(Fx',G ), Apply(Fx,G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

Complexity: O(|F|∗|G|)

Apply Operator Example

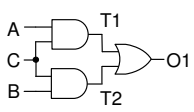


Apply Operator Example



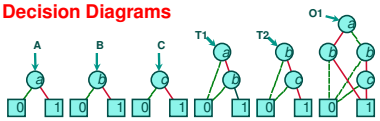
Generating ROBDDs Incrementally

- Network of components as ROBDD



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

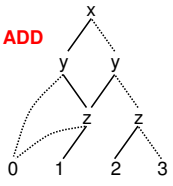
- Binary Decision Diagrams (BDDs)
- **Extensions: ADDs, MDDs, ZDDs, SBDDs**
- Decision Diagram Packages
- Combining Symbolic Encoding and Search

Algebraic DDs (ADDs) [Bahar 93]

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

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

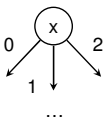
ADD



Multi-Valued DDs (MDDs) [Kam 90]

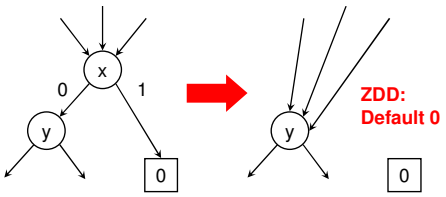
- Extension to functions with **non-binary variables**
- Canonicity of representation (as for ROBDDs)
- However, **binary encoding** of domains often better

MDD



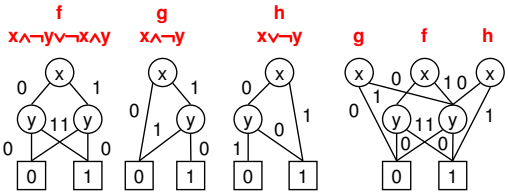
Zero-suppressed DDs (ZDDs) [Minato 93]

- Adapted to **sparse functions** (many 0's in on-set)
- Modified rule 2: Remove node if 1-edge points to 0



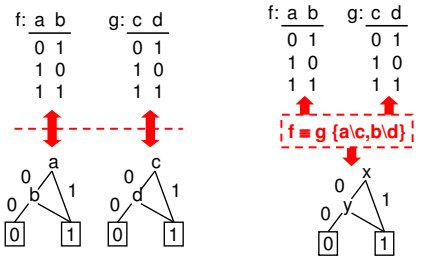
Shared BDDs (SBDDs) [Minato 90]

- **Global table** storing unique nodes
- Equivalence check for functions becomes constant



DD Templates [Goel Hasteer Bryant 03]

- Share not only functions, but also **function types**:



Overview

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

Decision Diagram Packages

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

Package Characteristics

Code	Name of the package	**Vmax**	Maximum # of variables
Lang	Programming language	**P**	Unique table per variable
T	Type (Pointers/Indices)	**GC**	Ref. Count or Mark-Sweep
R	Traversal (Depth/Breadth)	**CT**	Computed table shared
M	Supports Managers	**DV**	Dynamic Variable ordering
Nmax	Maximum # of nodes	**Ext**	# of functions in API
Siz	Size of a node in bytes	**Year**	First year when available
64	Size increases on 64-bit	**Pro**	Quality of code

Source: [Janssen 2002]

Comparing Packages

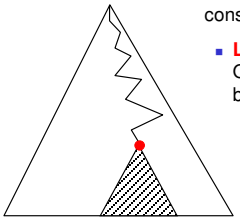
Code	Lang	T	R	M	Nmax	Siz	64	Vmax	P	GC	CT	DV	Ext	Year	Pro
ABC	C	I	D	Y	2^25	8	-	2^10	N	MS	S	N	44	1997	B
BUD	C/C++	I	D	N	2^32	20	N	2^21	N	MS	M	Y	116	1996	B
CAL	C	P	B	Y	2^28	16	Y	2^16	Y	RC 8	S	Y	237	1994	C
CMU	C	P	D	Y	2^29	16	Y	2^16	Y	RC 8	S	Y	75	1993	C
CUD	C/C++	P	D	Y	2^28	16	Y	2^16	N	RC16	S	Y	287	1995	B
EST	C	P	D	N	2^32	36	Y	2^32	N	RC32	M	N	72	2000	A
IBM	C	I	D	Y	2^30	16	N	2^24	N	MS	S	Y	169	1999	A
MON	C	I	D	Y	2^24	16	N	2^16	N	MS	M	N	40	1997	D
PDT	C	I	D	N	2^31	24	N	2^16	N	MS	M	N	34	1993	C
STA	C++	I	D	N	2^23	8	N	2^ 9	N	MS	S	N	66	1999	C
TGR	C	P	D	Y	2^30	16	Y	2^16	Y	RC16	M	Y	90	1994	?
TUD	C	I	D	Y	2^30	20	N	2^16	Y	RC16	S	Y	44	1998	B

Source: [Janssen 2002]

Overview

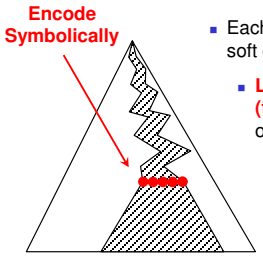
- Binary Decision Diagrams (BDDs)
- Extensions: ADDs, MDDs, ZDDs, SBDDs
- Decision Diagram Packages
- **Combining Symbolic Encoding and Search**

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

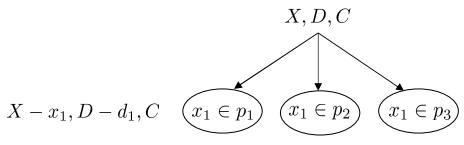


- Each search node is a **set of** soft constraint subproblems
  - **Lower Bound Function (f_{lb}):** Optimistic estimates of best solutions in subtree
    - **Upper Bound (ub):** Best solution found so far
      - Prune, if **f_{lb} ≥ ub**.

Domain Splitting

Generalize to search over **sets**:

- Partition domains into sets $P_i, \cup_{p \in P_i} = d_i$
- 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: $c_{12}(x_1, x_2)$, $c_{13}(x_1, x_3)$, $c_{14}(x_1, x_4)$, $c_{23}(x_2, x_3)$, $c_{24}(x_2, x_4)$, $c_{34}(x_3, x_4)$

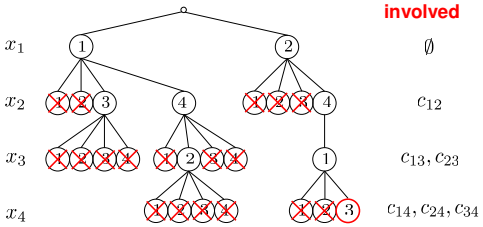
c_{12} : $x_1\ x_2$

1 3
1 4
2 4
3 1
4 1
4 2

	1	2	3	4
x_1		Q		
x_2				Q
x_3	Q			
x_4			Q	

Example

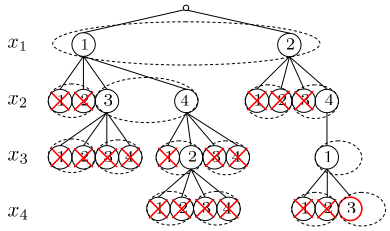
- Search Tree



Solution

Example

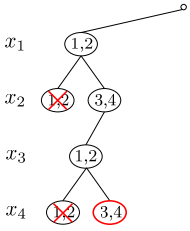
- Domain splitting with partitions $P_i = \{\{1,2\},\{3,4\}\}$



Solution

Example

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



Solution

Sinking Operation

- $\mathrm{sink}(c_j, \alpha)$ is a new constraint where all values of tuples $\succeq \alpha$ have been replaced by $\top$
- Generalizes the test $lb < ub$ to functions

Constraint

c_{e2} :

e_2	u	
G	0	.95
B	0	.05
B	1	.05

Constraint sink(c_{e2},0.05)

e_2	u	
G	0	.95
B	0	0
B	1	0

Symbolic Branch-and-Bound

- **Function** SDFBB(f_t: assignments, ub : value): value

```
f_lb ← lb(f_t)
f_t ← sink(f_lb, ub)
if f_t ≢ ⊤ then
    if var(f_t) = n then return f_t ⇓_∅
    let x_i be an unassigned variable
    for each p ∈ P_i do
        ub ← min(ub, SDFBB(f_t ⊕ (x_i ∈ p), ub))
    return ub
return ⊤
```

Distance lower bound: lb = identity.

Encode functions f_t, f_{lb} as decision diagrams.