Every BDD node can be interpreted as a mapping from valuations to truth values: Traverse the BDD from the given node to a leaf node; for any node labelled with P take the 0-edge or 1-edge depending on whether $\mathcal{A}(P)$ is 0 or 1.

$\Rightarrow$ Compact representation of truth tables.

OBDDs

OBDD (Ordered BDD):

Let $<$ be a total ordering of the propositional variables.

An OBDD w. r. t. $<$ is a BDD where every edge from a non-leaf node leads either to a leaf node or to a non-leaf node with a strictly larger label w. r. t. $<$.

OBDDs and formulas:

A leaf node $\boxed{0}$ represents $\bot$ (or any unsatisfiable formula).

A leaf node $\boxed{1}$ represents $\top$ (or any valid formula).

If a non-leaf node v has the label P, and its 0-edge leads to a node representing the formula F_0, and its 1-edge leads to a node representing the formula F_1, then v represents the formula

$$\begin{aligned} F \;&\models\!\!\!\!\!\!\;\;\dashv\; \text{if } P \text{ then } F_1 \text{ else } F_0 \\ &\models\!\!\!\!\!\!\;\;\dashv\; (P \wedge F_1) \vee (\neg P \wedge F_0) \\ &\models\!\!\!\!\!\!\;\;\dashv\; (P \rightarrow F_1) \wedge (\neg P \rightarrow F_0) \end{aligned}$$

Conversely:

Define $F\{P \mapsto H\}$ as the formula obtained from F by replacing every occurrence of P in F by H.

For every formula F and propositional variable P:

$$F \;\models\!\!\!\!\!\!\;\;\dashv\; (P \wedge F\{P \mapsto \top\}) \vee (\neg P \wedge F\{P \mapsto \bot\})$$

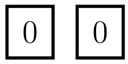
(*Shannon expansion* of F, originally due to Boole).

Consequence: Every formula F can be represented by an OBDD.

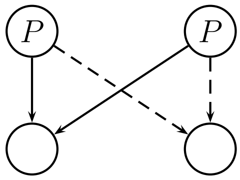
Reduced OBDDs

An OBDD is called *reduced*, if it has

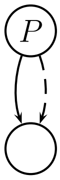
- no duplicated leaf nodes



- no duplicated interior nodes



- no redundant tests



Theorem 2.20 (Bryant 1986) *Every OBDD can be converted into an equivalent reduced OBDD.*

Assumptions from now on:

One fixed ordering $>$.

We consider only reduced OBDDs.

All OBDDs are sub-OBDDs of a single OBDD.

Implementation:

Bottom-up construction of reduced OBDDs is possible using a hash table.

Keys and values are triples $(\mathit{PropVar}, \mathit{Ptr}_0, \mathit{Ptr}_1)$,

where Ptr_0 and Ptr_1 are pointers to the 0-successor and 1-successor hash table entry.

Theorem 2.21 (Bryant 1986) *If v and v' are two different nodes in a reduced OBDD, then they represent non-equivalent formulas.*

Proof. We use induction over the maximum of the numbers of nodes reachable from v and v', respectively. Let F and F' be the formulas represented by v and v'.

Case 1: v and v' are non-leaf nodes labelled by different propositional variables P and P'. Without loss of generality, $P < P'$.

Let v_0 and v_1 be the 0-successor and the 1-successor of v, and let F_0 and F_1 be formulas represented by v_0 and v_1. We may assume without loss of generality that all propositional variables occurring in F', F_0, and F_1 are larger than P. By reducedness, $v_0 \neq v_1$, so by induction, $F_0 \not\models\!\!\!\!\!\!| \;\; F_1$. Hence there must be a valuation $\mathcal{A}$ such that $\mathcal{A}(F_0) \neq \mathcal{A}(F_1)$. Define valuations $\mathcal{A}_0$ and $\mathcal{A}_1$ by

$$\begin{array}{lll} \mathcal{A}_0(P) = 0 & \mathcal{A}_1(P) = 1 & \\ \mathcal{A}_0(Q) = \mathcal{A}(Q) & \mathcal{A}_1(Q) = \mathcal{A}(Q) & \text{for all } Q \neq P \end{array}$$

We know that the node v represents $F \models\!\!\!| \; (P \wedge F_1) \vee (\neg P \wedge F_0)$, so $\mathcal{A}_0(F) = \mathcal{A}_0(F_0) = \mathcal{A}(F_0)$ and $\mathcal{A}_1(F) = \mathcal{A}_1(F_1) = \mathcal{A}(F_1)$, and therefore $\mathcal{A}_0(F) \neq \mathcal{A}_1(F)$. On the other hand, P does not occur in F', therefore $\mathcal{A}_0(F') = \mathcal{A}_1(F')$. So we must have $\mathcal{A}_0(F) \neq \mathcal{A}_0(F')$ or $\mathcal{A}_1(F) \neq \mathcal{A}_1(F')$, which implies $F \not\models\!\!\!\!\!\!| \;\; F'$.

Case 2: v and v' are non-leaf nodes labelled by the same propositional variable.
Case 3: v is a non-leaf node, v' is a non-leaf node, or vice versa.
Case 4: v and v' are different leaf nodes.

Analogously. □

Corollary 2.22 *F is valid, if and only if it is represented by* $\boxed{1}$. *F is unsatisfiable, if and only if it is represented by* $\boxed{0}$.

Operations on OBDDs

Example:

Let $\circ$ be a binary connective.

Let P be the smallest propositional variable that occurs in F or G or both.

$$\begin{aligned} F \circ G \;\models\!\!\!|\;\; & (P \wedge (F \circ G)\{P \mapsto \top\}) \vee (\neg P \wedge (F \circ G)\{P \mapsto \bot\}) \\ \models\!\!\!|\;\; & (P \wedge (F\{P \mapsto \top\} \circ G\{P \mapsto \top\}) \\ & \quad \vee (\neg P \wedge (F\{P \mapsto \bot\} \circ G\{P \mapsto \bot\}))) \end{aligned}$$

Note: $F\{P \mapsto \top\}$ is either represented by the same node as F (if P does not occur in F), or by its 1-successor (otherwise).

⇒ Obvious recursive function on OBDD nodes
(needs memoizing for efficient implementation).

OBDD operations are not restricted to the connectives of propositional logic.

We can also compute operations of *quantified boolean formulas*

$$\forall P.\, F \;\models\!\!\!\!\models\; (F\{P \mapsto \top\}) \wedge (F\{P \mapsto \bot\})$$

$$\exists P.\, F \;\models\!\!\!\!\models\; (F\{P \mapsto \top\}) \vee (F\{P \mapsto \bot\})$$

and images or preimages of propositional formulas w. r. t. boolean relations (needed for typical verification tasks).

The size of the OBDD for $F \circ G$ is bounded by mn, where F has size m and G has size n. (Size = number of nodes)

With memoization, the time for computing $F \circ G$ is also at most $O(mn)$.

The size of an OBDD for a given formula depends crucially on the chosen ordering of the propositional variables:

Let $F = (P_1 \wedge P_2) \vee (P_3 \wedge P_4) \vee \cdots \vee (P_{2n-1} \wedge P_{2n})$.

$P_1 < P_2 < P_3 < P_4 < \cdots < P_{2n-1} < P_{2n}$: $2n + 2$ nodes.

$P_1 < P_3 < \cdots < P_{2n-1} < P_2 < P_4 < \cdots < P_{2n}$: 2^{n+1} nodes.

Even worse: There are (practically relevant!) formulas for which the OBDD has exponential size *for every ordering* of the propositional variables.

Example: middle bit of binary multiplication.

Literature

Randal E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation; IEEE Transactions on Computers, 35(8):677-691, 1986.

Randal E. Bryant: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams; ACM Computing Surveys, 24(3), September 1992, pp. 293-318.

Michael Huth and Mark Ryan: *Logic in Computer Science: Modelling and Reasoning about Systems*, Chapter 6.1/6.2; Cambridge Univ. Press, 2000.

2.10 FRAIGs

Goal:

Efficient manipulation of (equivalence classes of) propositional formulas.

Smaller representation than OBDDs.

Method: Minimized graph representation of boolean circuits.

FRAIG (Functionally Reduced And-Inverter Graph):

Labelled DAG (directed acyclic graph).

Leaf nodes:

labelled with propositional variables.

Non-leaf nodes (interior nodes):

labelled with $\wedge$ (two outgoing edges) or $\neg$ (one outgoing edge).

Reducedness (i. e., no two different nodes represent equivalent formulas) must be established explicitly, using

structural hashing,
simulation vectors,
CDCL,
OBDDs.

$\Rightarrow$ Semi-canonical representation of formulas.

Literature

A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton: FRAIGs: A unifying representation for logic synthesis and verification; ERL Technical Report, EECS Dept., UC Berkeley, March 2005.

2.11 Other Calculi

Ordered resolution
Tableau calculus
Hilbert calculus
Sequent calculus
Natural deduction

see next chapter

3 First-Order Logic

First-order logic

- formalizes fundamental mathematical concepts
- is expressive (Turing-complete)
- is not too expressive (e. g. not axiomatizable: natural numbers, uncountable sets)
- has a rich structure of decidable fragments
- has a rich model and proof theory

First-order logic is also called (first-order) *predicate logic*.

3.1 Syntax

Syntax:

- non-logical symbols (domain-specific)
  ⇒ terms, atomic formulas
- logical connectives (domain-independent)
  ⇒ Boolean combinations, quantifiers

Signatures

A signature $\Sigma = (\Omega, \Pi)$ fixes an alphabet of non-logical symbols, where

- Ω is a set of *function symbols* f with *arity* $n \geq 0$, written $\text{arity}(f) = n$,
- Π is a set of *predicate symbols* P with arity $m \geq 0$, written $\text{arity}(P) = m$.

Function symbols are also called *operator symbols*.
If $n = 0$ then f is also called a *constant (symbol)*.
If $m = 0$ then P is also called a *propositional variable*.

We will usually use

b, c, d for constant symbols,

f, g, h for non-constant function symbols,

P, Q, R, S for predicate symbols.

Convention: We will usually write $f/n \in \Omega$ instead of $f \in \Omega$, $\text{arity}(f) = n$ (analogously for predicate symbols).

Refined concept for practical applications:
many-sorted signatures (corresponds to simple type systems in programming languages); no big change from a logical point of view.

Variables

Predicate logic admits the formulation of abstract, schematic assertions. (Object) variables are the technical tool for schematization.

We assume that X is a given countably infinite set of symbols which we use to denote *variables*.

Terms

Terms over Σ and X (Σ-terms) are formed according to these syntactic rules:

$$\begin{array}{llll} s, t, u, v & ::= & x & , x \in X & \text{(variable)} \\ & \mid & f(s_1, ..., s_n) & , f/n \in \Omega & \text{(functional term)} \end{array}$$

By $\mathrm{T}_\Sigma(X)$ we denote the set of Σ-terms (over X). A term not containing any variable is called a *ground term*. By T_Σ we denote the set of Σ-ground terms.

In other words, terms are formal expressions with well-balanced parentheses which we may also view as marked, ordered trees. The markings are function symbols or variables. The nodes correspond to the *subterms* of the term. A node v that is marked with a function symbol f of arity n has exactly n subtrees representing the n immediate subterms of v.

Atoms

Atoms (also called atomic formulas) over Σ are formed according to this syntax:

$$\begin{array}{llll} A, B & ::= & P(s_1, \ldots, s_m) \ , P/m \in \Pi & \text{(non-equational atom)} \\ & \Big[\ \mid & (s \approx t) & \text{(equation)} \ \Big] \end{array}$$

Whenever we admit equations as atomic formulas we are in the realm of *first-order logic with equality*. Admitting equality does not really increase the expressiveness of first-order logic (see next chapter). But deductive systems where equality is treated specifically are much more efficient.

Literals

$$
\begin{array}{lcll}
L & ::= & A & \text{(positive literal)} \\
 & \mid & \neg A & \text{(negative literal)}
\end{array}
$$

Clauses

$$
\begin{array}{lcll}
C, D & ::= & \bot & \text{(empty clause)} \\
 & \mid & L_1 \vee \ldots \vee L_k, \;\; k \geq 1 & \text{(non-empty clause)}
\end{array}
$$

General First-Order Formulas

$\mathrm{F}_\Sigma(X)$ is the set of first-order formulas over Σ defined as follows:

$$
\begin{array}{lcll}
F, G, H & ::= & \bot & \text{(falsum)} \\
 & \mid & \top & \text{(verum)} \\
 & \mid & A & \text{(atomic formula)} \\
 & \mid & \neg F & \text{(negation)} \\
 & \mid & (F \wedge G) & \text{(conjunction)} \\
 & \mid & (F \vee G) & \text{(disjunction)} \\
 & \mid & (F \rightarrow G) & \text{(implication)} \\
 & \mid & (F \leftrightarrow G) & \text{(equivalence)} \\
 & \mid & \forall x\, F & \text{(universal quantification)} \\
 & \mid & \exists x\, F & \text{(existential quantification)}
\end{array}
$$

Notational Conventions

We omit parentheses according to the conventions for propositional logic.

$\forall x_1, \ldots, x_n\, F$ and $\exists x_1, \ldots, x_n\, F$ abbreviate $\forall x_1 \ldots \forall x_n\, F$ and $\exists x_1 \ldots \exists x_n\, F$.

We use infix-, prefix-, postfix-, or mixfix-notation with the usual operator precedences.

Examples:

$$
\begin{array}{ccc}
s + t * u & \text{for} & +(s, *(t, u)) \\
s * u \leq t + v & \text{for} & \leq(*(s, u), +(t, v)) \\
-s & \text{for} & -(s) \\
s! & \text{for} & !(s) \\
|s| & \text{for} & |\_|(s) \\
0 & \text{for} & 0()
\end{array}
$$