2.6 The DPLL Procedure

Goal:
Given a propositional formula in CNF (or alternatively, a finite set N of clauses), check whether it is satisfiable (and optionally: output *one* solution, if it is satisfiable).

Preliminaries

Recall:

$\mathcal{A} \models N$ if and only if $\mathcal{A} \models C$ for all clauses C in N.

$\mathcal{A} \models C$ if and only if $\mathcal{A} \models L$ for some literal $L \in C$.

Assumptions:

Clauses contain neither duplicated literals nor complementary literals.

The order of literals in a clause is irrelevant.

$\Rightarrow$ Clauses behave like *sets* of literals.

Notation:

We use the notation $C \vee L$ to denote a clause with some literal L and a clause rest C. Here L need *not* be the last literal of the clause and C may be empty.

$\overline{L}$ is the complementary literal of L, i. e., $\overline{P} = \neg P$ and $\overline{\neg P} = P$.

Partial Valuations

Since we will construct satisfying valuations incrementally, we consider *partial valuations* (that is, partial mappings $\mathcal{A} : \Pi \to \{0, 1\}$).

Every partial valuation $\mathcal{A}$ corresponds to a set M of literals that does not contain complementary literals, and vice versa:

$\mathcal{A}(L)$ is true, if $L \in M$.

$\mathcal{A}(L)$ is false, if $\overline{L} \in M$.

$\mathcal{A}(L)$ is undefined, if neither $L \in M$ nor $\overline{L} \in M$.

We will use $\mathcal{A}$ and M interchangeably.

A clause is true in a partial valuation $\mathcal{A}$ (or in a set M of literals) if one of its literals is true; it is false (or *"conflicting"*) if all its literals are false; otherwise it is undefined (or *"unresolved"*).

Unit Clauses

Observation:
Let $\mathcal{A}$ be a partial valuation. If the set N contains a clause C, such that all literals but one in C are false in $\mathcal{A}$, then the following properties are equivalent:

- there is a valuation that is a model of N and extends $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ and makes the remaining literal L of C true.

C is called a *unit clause;* L is called a *unit literal.*

Pure Literals

One more observation:
Let $\mathcal{A}$ be a partial valuation and P a variable that is undefined in $\mathcal{A}$. If P occurs only positively (or only negatively) in the unresolved clauses in N, then the following properties are equivalent:

- there is a valuation that is a model of N and extends $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ and assigns 1 (0) to P.

P is called a *pure literal.*

The Davis-Putnam-Logemann-Loveland Procedure

```
boolean DPLL(literal set $M$, clause set $N$) {
    if (all clauses in $N$ are true in $M$) return true;
    elsif (some clause in $N$ is false in $M$) return false;
    elsif ($N$ contains unit literal $P$) return DPLL($M \cup \{P\}$, $N$);
    elsif ($N$ contains unit literal $\neg P$) return DPLL($M \cup \{\neg P\}$, $N$);
    elsif ($N$ contains pure literal $P$) return DPLL($M \cup \{P\}$, $N$);
    elsif ($N$ contains pure literal $\neg P$) return DPLL($M \cup \{\neg P\}$, $N$);
    else {
        let $P$ be some undefined variable in $N$;
        if (DPLL($M \cup \{\neg P\}$, $N$)) return true;
        else return DPLL($M \cup \{P\}$, $N$);
    }
}
```

Initially, DPLL is called with an empty literal set and the clause set N.

2.7 From DPLL to CDCL[3]

The DPLL procedure can be improved significantly:

The pure literal check is only done while preprocessing (otherwise is too expensive).

If a conflict is detected, information is reused by conflict analysis and learning.

The algorithm is implemented iteratively $\Rightarrow$ the backtrack stack is managed explicitly (it may be possible and useful to backtrack more than one level).

The branching variable is not chosen randomly.

Under certain circumstances, the procedure is restarted.

Literature:

Lintao Zhang and Sharad Malik: The Quest for Efficient Boolean Satisfiability Solvers, Proc. CADE-18, LNAI 2392, pp. 295–312, Springer, 2002.

Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli: Solving SAT and SAT Modulo Theories – From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T), pp. 937–977, Journal of the ACM, 53(6), 2006.

Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh (eds.): Handbook of Satisfiability, IOS Press, 2009

Daniel Le Berre's slides at VTSA'09: `http://www.mpi-inf.mpg.de/vtsa09/`.

Conflict Analysis and Learning

Conflict analysis serves a dual purpose:

Backjumping (non-chronological backtracking): If we detect that a conflict is independent of some earlier branch, we can skip over that backtrack level.

Learning: By deriving a new clause from a conflict that is added to the current set of clauses, we can reuse information that is obtained in one branch in further branches. (Note: This may produce a large number of new clauses; therefore it may become necessary to delete some of them afterwards to save space.)

These ideas are implemented in all modern SAT solvers.

Because of the importance of clause learning the algorithm is now called CDCL: Conflict Driven Clause Learning.

[3]The presentation in this subsection differs significantly from the 2021/2022 lecture. Keep that in mind when you use online lecture recordings or read exercises or exam questions from previous years.

Formalizing CDCL

We model the improved DPLL procedure by a transition relation $\Rightarrow_{\text{CDCL}}$ on a set of states.

States:

- *fail*
- $M \parallel N$,

where M is a *list of annotated literals (“trail”)* and N is a set of clauses.

Annotated literals:

- L^C: deduced literal, due to unit propagation using clause C.
- L^{d}: decision literal (guessed literal).

Alternative notation: $L^{C_i} \quad \equiv \quad \begin{matrix} L \\ C_i \end{matrix} \quad \equiv \quad \begin{matrix} L \\ i \end{matrix}$

Unit Propagate:

$M \parallel N \cup \{C \vee L\} \;\Rightarrow_{\text{CDCL}}\; M\ L^{C \vee L} \parallel N \cup \{C \vee L\}$

if C is false in M and L is undefined in M.

Decide:

$M \parallel N \;\Rightarrow_{\text{CDCL}}\; M\ L^{\text{d}} \parallel N$

if L is undefined in M and contained in N.

Fail:

$M \parallel N \cup \{C\} \;\Rightarrow_{\text{CDCL}}\; \textit{fail}$

if C is false in M and M contains no decision literals.

Backjump:

$M'\ K^{\text{d}}\ M'' \parallel N \;\Rightarrow_{\text{CDCL}}\; M'\ L^{C \vee L} \parallel N$

if some clause $D \in N$ is false in $M'\ K^{\text{d}}\ M''$
and if there is some “backjump clause” $C \vee L$ such that
$N \models C \vee L$,
C is false in M', and
L is undefined in M'.

We will see later that the Backjump rule is always applicable, if the list of literals M contains at least one decision literal and some clause in N is false in M.

There are many possible backjump clauses. One candidate is $\overline{L_1} \vee \ldots \vee \overline{L_n}$, where the L_i are all the decision literals in $M' \; L^{\mathrm{d}} \; M''$. With this backjump clause, CDCL simulates DPLL. (But usually there are better choices.)

Reasonable strategy:

- Use “Fail”, if applicable.
- Otherwise use “Backjump”, if applicable.
  Choose M' as short as possible. ($\Rightarrow$ Go back to the earliest position where guessing can be replaced by knowledge.)
- Otherwise use “Unit Propagate”, if applicable.
- Otherwise use “Decide”.

A trail M defines an ordering $\succ_M$ on literals: $L \succ_M K$ if L or $\overline{L}$ occurs in M after (i. e., right of) K or $\overline{K}$ (with any annotation d or C).

Lemma 2.16 *If $\varepsilon \parallel N \;\Rightarrow^*_{\mathrm{CDCL}}\; M \parallel N$, then:*

(1) M contains neither duplicated nor complementary literals (regardless of their annotations).

(2) If L^C is a deduced literal in M, then C has the form $C' \vee L$.

(3) If $L^{C' \vee L}$ is a deduced literal in M, then $L \succ_M K$ for every literal K of C'.

(4) If $L^{C' \vee L}$ is a deduced literal in M, then C' is false in M.

(5) Every literal L in M follows from N and decision literals in M that are smaller than or equal to L w. r. t. $\succ_M$.

Proof. By induction on the length of the derivation. □

Lemma 2.17 *Every derivation starting from $\varepsilon \parallel N$ terminates.*

Proof. Let $n \in \mathbb{N}$ be the number of propositional variables in N. We define a function ϕ that maps every CDCL state $M \parallel N$ in the derivation to a list of annotated literals S such that S is obtained from M by deleting all the C-superscripts of deduced literals in M. (The d-superscripts of decision literals in M are kept.) Since each of the n propositional variables can occur at most once in M, each of them can occur at most once in S (positively or negatively, with or without a d-superscript), therefore the number of possible lists that can be returned by ϕ is bounded by $n! \cdot 5^n$.

Let $M \parallel N$ and $M' \parallel N'$ be two CDCL states, such that

$$\phi(M \parallel N) = S = S_0 L_1^{\mathrm{d}} S_1 \dots L_k^{\mathrm{d}} S_k$$

and

$$\phi(M' \parallel N') = S' = S'_0 {L'_1}^{\mathrm{d}} S'_1 \dots {L'_{k'}}^{\mathrm{d}} S'_{k'},$$

where the lists S_i and S'_i contain no decision literals. We define a relation $\succ$ on lists of annotated literals by $S \succ S'$ if and only if

(i) there is some j such that $0 \leq j \leq \min(k, k')$, $|S_i| = |S'_i|$ for all $0 \leq i < j$, and $|S_j| < |S'_j|$, or

(ii) $|S_i| = |S'_i|$ for all $0 < i \leq k < k'$ and $|S| < |S'|$.

It is routine to check that $\succ$ is irreflexive and transitive, hence a strict partial ordering, and that for every CDCL step $M \parallel N \Rightarrow_{\mathrm{CDCL}} M' \parallel N'$ we have $\phi(M \parallel N) \succ \phi(M' \parallel N')$. Now assume that there is an infinite CDCL derivation

$$\varepsilon \parallel N_0 \Rightarrow_{\mathrm{CDCL}} M_1 \parallel N_1 \Rightarrow_{\mathrm{CDCL}} M_2 \parallel N_2 \Rightarrow_{\mathrm{CDCL}} \dots$$

Since ϕ can return only finitely many lists, there must exist indices $i < j$ such that

$$\phi(M_i \parallel N_i) \succ^+ \phi(M_j \parallel N_j) = \phi(M_i \parallel N_i).$$

By transitivity, this implies $\phi(M_i \parallel N_i) \succ \phi(M_i \parallel N_i)$, but that contradicts the irreflexivity of $\succ$.

Lemma 2.18 *Suppose that we reach a state $M \parallel N$ starting from $\varepsilon \parallel N$ such that some clause $D \in N$ is false in M. Then:*

(1) If M does not contain any decision literal, then "Fail" is applicable.

(2) Otherwise, "Backjump" is applicable.

Proof. (1) Obvious.

(2) Let $L_1, \dots, L_n$ be the decision literals occurring in M (in this order). By part (5) of Lemma 2.16, every literal in M follows from $N \cup \{L_1, \dots, L_n\}$, and since $M \models \neg D$, we obtain $N \cup \{L_1, \dots, L_n\} \models \neg D$. By Prop. 2.6, this is equivalent to $N \cup \{L_1, \dots, L_n\} \cup D \models \bot$, and since $D \in N$, it is equivalent to $N \cup \{L_1, \dots, L_n\} \models \bot$. Consequently, $N \models \overline{L_1} \vee \dots \vee \overline{L_n}$. Now let $C = \overline{L_1} \vee \dots \vee \overline{L_{n-1}}$, $L' = \overline{L_n}$, $L = L_n$, and let M' be the list of all literals of M occurring before L_n, then the condition of "Backjump" is satisfied. $\square$

Theorem 2.19 *Suppose that we reach a final state starting from $\varepsilon \parallel N$.*

(1) If the final state is $M \parallel N$, then N is satisfiable and M is a model of N.

(2) If the final state is fail, then N is unsatisfiable.

Proof. (1) Observe that the "Decide" rule is applicable as long as literals in N are undefined in M. Hence, in a final state, all literals must be defined. Furthermore, in a final state, no clause in N can be false in M, otherwise "Fail" or "Backjump" would be applicable. Hence M is a model of every clause in N.

(2) If we reach *fail*, then in the previous step we must have reached a state $M \parallel N$ such that some $C \in N$ is false in M (i.e., $M \models \neg C$), and M contains no decision literals. By part (5) of Lemma 2.16, every literal in M follows from N, therefore $N \models \neg C$, and therefore $N \cup \{C\} \models \bot$. On the other hand, $C \in N$, so $N \models \bot$. □

Getting Better Backjump Clauses

Lemma 2.20 *Suppose that $\varepsilon \parallel N \Rightarrow^*_{\text{CDCL}} M \parallel N$. Let D is a clause such that $N \models D$ and D is false in M. Let $\overline{L}$ be the largest literal of D w.r.t. $\succ_M$ and let $D = D' \vee \overline{L}$. Suppose that $L^{C \vee L}$ is a deduced literal in M. Let $D_0 = C \vee D'$. Then $N \models D_0$; D_0 is false in M; and all literals of D_0 are smaller than $\overline{L}$.*

Proof. If D is false in M, then every literal of D is the complement of a literal in M. Let $\overline{L}$ be the largest literal of D w.r.t. $\succ_M$ and let $D = D' \vee \overline{L}$. Clearly, D' is false in M. If L is a deduced literal, then M contains $L^{C \vee L}$ and all literals of C are smaller than L and $\overline{L}$. Moreover, C is false in M, and by construction of M, we have $N \models C \vee L$. Therefore, $D_0 = C \vee D'$ is false in M; all literals of D_0 are smaller than $\overline{L}$; and $N \models D_0$ by (generalized) resolution. □

The clause $C \vee D'$ is called a *resolvent* of $C \vee L$ and $D' \vee \overline{L}$; this is denoted by

$$\frac{C \vee L \qquad D' \vee \overline{L}}{C \vee D'}$$

Note that the resolvent $C \vee D'$ is again entailed by N and false in M. If its largest literal is the complement of a deduced literal, we can therefore repeat the process with $C \vee D'$.

Lemma 2.21 *Suppose that $\varepsilon \parallel N \Rightarrow^*_{\text{CDCL}} M \parallel N$. Let D is a clause such that $N \models D$ and D is false in M. If D has the form $D = D' \vee \overline{L}$, where L is larger than or equal to the largest decision literal of M and all literals in D' are smaller than the largest decision literal of M, then M can be written as $M' \, K^{\text{d}} \, M''$ such that D' is false in M' and L is undefined in M' (that is, D is a backjump clause).*

Proof. By assumption, there exists a decision literal of M that is larger than every literal of D'. Choose K as the smallest decision literal with this property. □

Suppose that $\varepsilon \parallel N \Rightarrow^*_{\text{CDCL}} M \parallel N$, where L^d is the largest decision literal in M and some $D \in N$ is false in M.

If we have used a reasonable strategy, then D must contain two literals whose complements are larger than or equal to L.

By repeated resolution steps as described in Lemma 2.20, we must eventually reach a clause that satisfies Lemma 2.21.

This clause is a backjump clause
$\Rightarrow$ *1UIP (first unique implication point) strategy.*

Learning Clauses

Backjump clauses are good candidates for learning.

To model learning, the CDCL system is extended by the following two rules:

Learn:

$M \parallel N \Rightarrow_{\text{CDCL}} M \parallel N \cup \{C\}$

if $N \models C$.

Forget:

$M \parallel N \cup \{C\} \Rightarrow_{\text{CDCL}} M \parallel N$

if $N \models C$.

If we ensure that no clause is learned infinitely often, then termination is guaranteed.

The other properties of the basic CDCL system hold also for the extended system.

Restart

Runtimes of CDCL-style procedures depend extremely on the choice of branching variables.

If no solution is found within a certain time limit, it can be useful to *restart* from scratch with an adapted variable selection heuristics. Learned clauses, however, are kept.

In addition, it is useful to restart after a unit clause has been learned.

The restart rule is typically applied after a certain number of clauses have been learned or a unit is derived:

Restart:

$$M \parallel N \;\Rightarrow_{\text{CDCL}}\; \varepsilon \parallel N$$

If Restart is only applied finitely often, termination is guaranteed.

2.8 Implementing CDCL

The formalization of CDCL that we have seen so far leaves many aspects unspecified.

To get a fast solver, we must use good heuristics, for instance to choose the next undefined variable, and we must implement basic operations efficiently.

Variable Order Heuristic

Choosing the right undefined variable to branch is important for efficiency, but the branching heuristics may be expensive itself.

State of the art: Use branching heuristics that need not be recomputed too frequently.

In general: Choose variables that occur frequently; after a restart prefer variables from recent conflicts.

The VSIDS (Variable State Independent Decaying Sum) heuristic:

- We associate a positive *score* to every propositional variable P_i. At the start, k_i is the number of occurrences of P_i in N.
- The variable order is then the descending ordering of the P_i according to the k_i.

The scores k_i are adjusted during a CDCL run.

- Every time a learned clause is computed after a conflict, the propositional variables in the learned clause obtain a bonus b, i.e., $k_i := k_i + b$.
- Periodically, the scores are leveled: $k_i := k_i/l$ for some l.
- After each restart, the variable order is recomputed, using the new scores.

The purpose of these mechanisms is to keep the search focused. The parameter b directs the search around the conflict,

Further refinements:

- Add the bonus to all literals in the clauses that occur in the resolution steps to generate a backjump clause.
- If the score of a variable reaches a certain limit, all scores are rescaled by a constant.
- Occasionally (with low probability) choose a variable at random, otherwise choose the undefined variable with the highest score.

Implementing Unit Propagation Efficiently

For applying the unit rule, we need to know the number of literals in a clause that are not false.

Maintaining this number is expensive, however.

Better approach: *"Two watched literals":*

In each clause, select two (currently undefined) "watched" literals.

For each variable P, keep a list of all clauses in which P is watched and a list of all clauses in which $\neg P$ is watched.

If an undefined variable is set to 0 (or to 1), check all clauses in which P (or $\neg P$) is watched and watch another literal (that is true or undefined) in this clause if possible.

Watched literal information need not be restored upon backtracking.

2.9 Preprocessing and Inprocessing

Some operations are only needed once at the beginning of the CDCL run.

(i) Deletion of tautologies

(ii) Deletion of duplicated literals

Some operations are useful, but expensive. They are performed only initially and after restarts (before computation of the variable order heuristics), possibly with time limits.

Note: Some of these operations are only satisfiability-preserving; they do not yield equivalent clause sets.

Literature:

Matti Järvisalo, Marijn J. H. Heule, and Armin Biere: Inprocessing Rules, Proc. IJCAR 2012, LNAI 7364, pp. 355–370, Springer, 2012

Examples:

(i) *Subsumption*

$$N \cup \{C\} \cup \{D\} \;\Rightarrow\; N \cup \{C\}$$

if $C \subseteq D$ considering C, D as multisets of literals.

(ii) *Purity deletion*

Delete all clauses containing a literal L where $\overline{L}$ does not occur in the clause set.

(iii) *Merging replacement resolution*

$$N \cup \{C \vee L\} \cup \{D \vee \overline{L}\} \;\Rightarrow\; N \cup \{C \vee L\} \cup \{D\}$$

if $C \subseteq D$ considering C, D as multisets of literals.

(iv) *Bounded variable elimination*

Compute all possible resolution steps

$$\frac{C \vee L \qquad D \vee \overline{L}}{C \vee D}$$

on a literal L with premises in N; add all non-tautological conclusions to N; then throw away all clauses containing L or $\overline{L}$; repeat this as long as $|N|$ does not grow.

(v) *RAT ("Resolution asymmetric tautologies")*

C is called an *asymmetric tautology* w. r. t. N, if its negation can be refuted by unit propagation using clauses in N.

C has the *RAT property* w. r. t. N, if it is an asymmetric tautology w. r. t. N, or if there is a literal L in C such that $C = C' \vee L$ and all clauses $D' \vee C'$ for $D' \vee \overline{L} \in N$ are asymmetric tautologies w. r. t. N.

RAT elimination:

$$N \cup \{C\} \;\Rightarrow\; N$$

if C has the RAT property w. r. t. N.

2.10 OBDDs

Goal:

Efficient manipulation of (equivalence classes of) propositional formulas.

Method: Minimized graph representation of decision trees, based on a fixed ordering on propositional variables.

$\Rightarrow$ Canonical representation of formulas.

$\Rightarrow$ Satisfiability checking as a side effect.

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.

BDDs

BDD (Binary decision diagram):

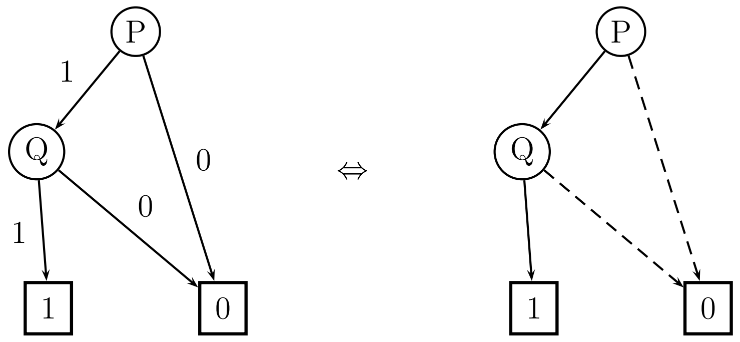
Labelled DAG (directed acyclic graph).

Leaf nodes:

labelled with a truth value (0 or 1).

Non-leaf nodes (interior nodes):

labelled with a propositional variable,
exactly two outgoing edges, labelled with 0 ($\dashrightarrow$) and 1 ($\longrightarrow$)



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

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

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:

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

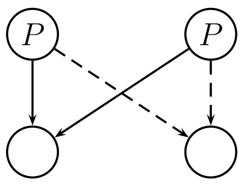
(*Shannon expansion* of F, originally due to Boole).

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

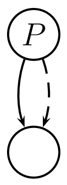
Reduced OBDDs

An OBDD is called *reduced*, if it has

- no duplicated leaf nodes $\boxed{0}$ $\boxed{0}$
- no duplicated interior nodes



- no redundant tests



Theorem 2.22 (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 2.23 (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, $F_0 \not\models\!\mid F_1$. 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 $F \models\!\mid (P \wedge F_1) \vee (\neg P \wedge F_0)$, 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 $F \not\models\!\mid F'$.

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. $\square$

Corollary 2.24 *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.

$$\begin{aligned} F \circ G &\models\mid (P \land (F \circ G)\{P \mapsto \top\}) \lor (\neg P \land (F \circ G)\{P \mapsto \bot\}) \\ &\models\mid (P \land (F\{P \mapsto \top\} \circ G\{P \mapsto \top\}) \\ &\qquad \lor (\neg P \land (F\{P \mapsto \bot\} \circ G\{P \mapsto \bot\}))) \end{aligned}$$

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).

$\Rightarrow$ 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*

$$\forall P.\, F \models\mid (F\{P \mapsto \top\}) \land (F\{P \mapsto \bot\})$$

$$\exists P.\, F \models\mid (F\{P \mapsto \top\}) \lor (F\{P \mapsto \bot\})$$

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 $F = (P_1 \land P_2) \lor (P_3 \land P_4) \lor \cdots \lor (P_{2n-1} \land P_{2n})$.

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

2.11 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.

2.12 Other Calculi

Ordered resolution
Tableau calculus
Hilbert calculus
Sequent calculus
Natural deduction

see next chapter