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):

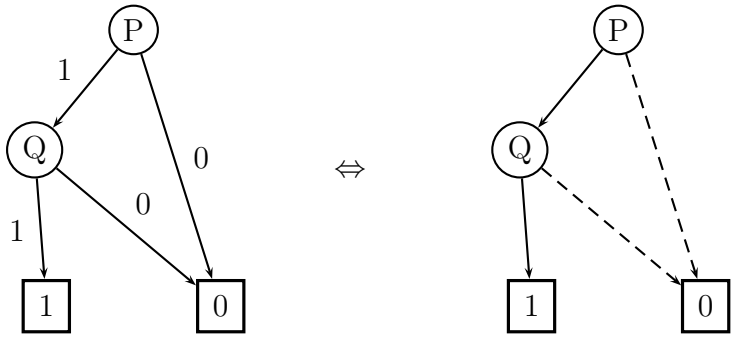
Labelled DAG (directed acyclic graph).

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 (- - →) and 1 (⟶)



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

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

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:

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

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

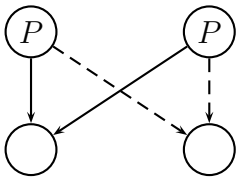
Consequence: Every formula F can be represented by an OBDD.

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 $(PropVar, Ptr_0, Ptr_1)$,

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 $F \models\!\!\models (P \wedge F_1) \vee (\neg P \wedge F_0)$, 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.

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

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).

$\Rightarrow$ 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*

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

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

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.

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