DPLL Algorithm

Anthony W. Lin

What it is

- A complete algorithm for SAT based on search (depth-first search) and deduction (unit resolution)

- Developed by Davis, Putnam, Logemann, and Loveland over 50 years ago.

- Modern SAT-solvers enhance DPLL algorithms with numerous heuristics (e.g. clause learning, non-chronological backtracking, branching strategies, restart strategies, better data structures, …)

A simple DFS algorithm

```
def SAT(Formula):
    if Formula == true:
        return true
    elif Formula == false:
        return false
    else:
        x = CHOOSE_VAR(Formula)
        return Formula[true/x] or Formula[false/x]
```

pick a new
decision variable

boolean or

Example

Apply DFS on the following formula

$$(\neg x \vee y) \wedge x \wedge \neg y$$

Unit Resolution

DPLL applies a restricted version of the resolution inference rule

$$\frac{\ell \qquad C[\neg \ell]}{C[\bot / \neg \ell]}$$

unit clause

Example: $$\frac{\{x\} \qquad \{\neg x, y, \neg z\}}{\{y, \neg z\}}$$

Boolean Constraint Propagation (BCP)

Keep applying unit resolution

$$\{P\} \qquad \{\neg P, Q\} \qquad \{R, \neg Q, S\}$$

$$\{Q\}$$

$$\{R, S\}$$

DPLL Algorithm

```
def SAT(Formula):
    Formula = BCP(Formula)
    if Formula == true:
        return true
    elif Formula == false:
        return false
    else:
        x = CHOOSE_VAR(Formula)
        return Formula[true/x] or Formula[false/x]
```

Example

$$\{\neg P, Q, R\} \qquad \{\neg Q, R\} \qquad \{\neg Q, \neg R\} \qquad \{P, \neg Q, \neg R\}$$

A simple improvement

We call a literal ℓ <u>pure</u> if it appears only positively or negatively in Formula

What is so special about pure literals?

We can make them true before choosing a decision variable to branch!

DPLL Algorithm

keep applying
BCP and
SET_PURE_TRUE
till not possible

```
def SAT(Formula):
    while true:
        Formula1 = BCP(Formula)
        Formula2 = SET_PURE_TRUE(Formula1)
        if Formula2 == Formula1:
            break
    if Formula == true:
        return true
    elif Formula == false:
        return false
    else:
        x = CHOOSE_VAR(Formula)
        return Formula[true/x] or Formula[false/x]
```

Example

$\{\neg P, Q, R\}$ $\{\neg Q, R\}$ $\{\neg Q, \neg R\}$ $\{P, \neg Q, \neg R\}$

Example

$$\{A, \neg B, \neg C\} \quad \{\neg A, \neg B, \neg C\} \quad \{B, C\} \quad \{C, D\} \quad \{C, \neg D\}$$

Example

$$\{A, \neg B, D\}\ \{A, \neg B, E\}\ \{\neg B, \neg D, \neg E\}\ \{A, B, C, D\}$$
$$\{A, B, C, \neg D\}\{A, B, \neg C, E\}\{A, B, \neg C, \neg E\}$$

Correctness of DPLL

Termination: Can argue by induction on the size of the formula. This uses the fact that both BCP and SET_PURE_TRUE terminate and never output a bigger formula.

Partial Correctness (i.e. if terminates output the right thing): Can argue by induction on the size of the formula using that BCP produces an equivalent formula, and SET_PURE_TRUE outputs an equisatisfiable formula.

Implementation Matter

Naive implementation of BCP takes quadratic time

We can make it a linear time algorithm by using <u>three hash maps</u> for the formula.

$$\{P\} \qquad \{\neg P, Q\} \qquad \{R, \neg Q, S\}$$

Formula $$F = \{1 : \{\neg P\}, 2 : \{\neg P, Q\}, 3 : \{R, \neg Q, S\}\}$$

Var->Clause $$VC = \{P : \{1, 2\}, Q : \{2, 3\}, R : \{3\}, S : \{3\}\}$$

Set of keys for F for unit clauses $$U = \{1\}$$

Conflict Driven Clause Learning

Background

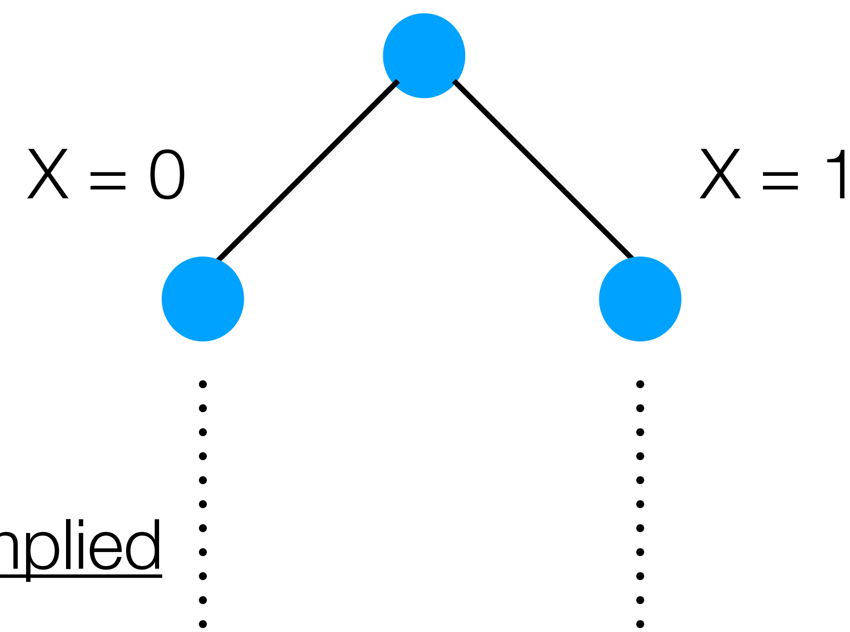
- CDCL enhances DPLL with: (1) clause learning, and (2) non-chronological backtracking (a.k.a. backjumping)
- CDCL was proposed by Marques-Silva and Sakallah, and Bayardo and Schrag in mid-late 1990s
- They were awarded CAV'2009 Test-of-Time Award for their "*fundamental contributions to the development of high-performance boolean satisfiability solvers*"
- Most modern SAT-solvers improve CDCL (e.g. Minisat, Zchaff, Z3, Glucose, Lingeling, …) by other heuristics

Clause Learning

If conflict occurs because of BCP, learn a new clause that explains and prevents the same conflict.

Some terminology

x is a decision variable



X = 0 is a decision assignment

BCP might generate an implied assignment, e.g., Y = 1
[e.g. if Y is a unit clause]

Conflict clause is a clause that is violated in a branch of this search tree

More notation

Label an implied assignment by the unit clause

$$\underset{\text{C1}}{\{P, Q\}} \quad \underset{\text{C2}}{\{\neg P\}}$$

(P=0,C2)

Conflict Analysis

Decision assignment

<X1=1,X3=1,X2=0>

C1 $\{\neg X_1, X2, \neg X_3, X_4\}$

C2 $\{\neg X_1, \neg X2, \neg X_3, X_4\}$

C3 $\{\neg X_1, \neg X_4, X_5\}$

C4 $\{\neg X_1, \neg X_5, X_6\}$

C5 $\{\neg X_1, \neg X_4, \neg X_6\}$

C6 $\{X_1, X_3, X_5\}$

Implied assignment

<(X4=1,C1),(X5=1,C3),(X6=1,C4)>

Conflict clause

$\{\neg X_1, \neg X_4, \neg X_6\}$

Characteristic of a *good conflict clause*

It’s a conflict clause

Adding it to the formula preserves equivalence

It contains only decision variables

$\{\neg X_1, \neg X_4, \neg X_6\}$ NOT yet a good conflict clause

Learning a good conflict clause

$\mathcal{A} = \langle p_1 \mapsto b_1, \ldots, p_k \mapsto b_k \rangle$ – assignment leading to conflict

Use backward induction (via resolution) and compute the learned clause A_1

1. A_{k+1} any conflict clause

2. p_i is a decision variable or not appear in A_{i+1} ==>

$$A_i = A_{i+1}$$

3. $p_i \mapsto b_i$ is an implied assignment (implied by C_i) that appears in A_{i+1}

A_i = a resolvent of A_{i+1} and C_i

Example

C1 $\{\neg X_1, X2, \neg X_3, X_4\}$

C2 $\{\neg X_1, \neg X2, \neg X_3, X_4\}$

C3 $\{\neg X_1, \neg X_4, X_5\}$

C4 $\{\neg X_1, \neg X_5, X_6\}$

C5 $\{\neg X_1, \neg X_4, \neg X_6\}$

C6 $\{X_1, X_3, X_5\}$

<X1=1,X3=1,X2=0>

<(X4=1,C1),(X5=1,C3),(X6=1,C4)>

$$A_7 = \{\neg X_1, \neg X_4, \neg X_6\}$$

$$A_6 = ?$$

$$\cdots$$

$$A_1 = ?$$

Where to backtrack to

The highest level where the learned clause is a unit clause

$$\{\neg X_1, X2, \neg X_3, X_4\}$$
$$\{\neg X_1, \neg X2, \neg X_3, X_4\}$$
$$\{\neg X_1, \neg X_4, X_5\}$$
$$\{\neg X_1, \neg X_5, X_6\}$$
$$\{\neg X_1, \neg X_4, \neg X_6\}$$
$$\{X_1, X_3, X_5\}$$

<X1=1,X3=1,X2=0>

<(X4=1,C1),(X5=1,C3),(X6=1,C4)>