Some definitions

For F in CNF:
$\alpha \models F$ if in every clause C there is literal $a \in C$ with $\alpha(a) = 1$.

A clause of width 1 is called a unit clause.

Property

If a is a unit clause in F, then $\alpha(a) = 1$ for every $\alpha \models F$.

Literal a is pure in F if $\bar{a}$ does not occur in F

Property

If $C \in F$ contains a pure literal, then F is satisfiable iff $F \setminus C$ is satisfiable.

Resolution

Proof system for showing unsatisfiability of formulas in CNF.

The only inference rule: Resolution

$$\frac{(C \vee a) \qquad (D \vee \bar{a})}{(C \vee D)}$$

$C \vee D$ is called the resolvent of $C \vee a$ and $D \vee \bar{a}$.

Lemma

For every assignment α:

If $\alpha \models C \vee a$ and $\alpha \models D \vee \bar{a}$, then also $\alpha \models C \vee D$.

Resolution proofs

A resolution derivation of a clause C
from a formula F in CNF is

- a sequence $C_1, \ldots, C_k$ of clauses with:
- $C_k = C$
- for every $i \leq k$ one of the following holds:
  - C_i is a clause $C_i \in F$,
  - there are $j_1, j_2 < i$ s.t. C_i is the resolvent of C_{j_1} and C_{j_2}.

A refutation of F is a derivation of the empty clause $\square$ from F.

Resolution: soundness and completeness

Theorem

Let a resolution derivation of C from F be given.

Then for every assignment α:
If $\alpha \models F$, then also $\alpha \models C$.

Corollary

If there is a refutation of F, then F is unsatisfiable.

Theorem

If F is unsatisfiable, then there is a resolution refutation of F.

Overview

DPLL algorithms
- Simplifications
- Monien-Speckenmeyer algorithm
- Kullmann's method

The basic DPLL algorithm

After *Davis, Putnam, Logemann* and *Loveland*, 1960

The basic DPLL Algorithm:

DPLL(F, α)

  if $F\alpha = 0$ then return UNSAT
  if $F\alpha = 1$ then return α

  pick $x \in V(F\alpha)$

  $\beta :=$ DPLL($F, \alpha \cup [x := 0]$)

  if $\beta \neq$ UNSAT
    then return β
    else return DPLL($F, \alpha \cup [x := 1]$)

A simple analysis

For a k-CNF formula F, iterate the following:

- pick a clause C in F
- branch successively on the k variables in $V(C)$

Of the 2^k assignments to $V(C)$, one sets $\alpha(C) = \alpha(F) = 0$.

Thus: $2^k - 1$ branching tree of height n/k.

Runtime is essentially the tree-size b_k^n for $b_k := \sqrt[k]{2^k - 1} < 2$

k	3	4	5	6	7	8
b_k	1.91294	1.96799	1.98735	1.99477	1.99777	1.99903

Unit propagation

A branching strategy (almost) always employed:

- pick x if x occurs in a unit clause a.

One branch immediately fails

$\rightsquigarrow$ just set $[a := 1]$ instead of branching

Mostly realized as simplification step before branching:

```
UnitProp(F, α)
  while Fα contains unit clause a
     α := α ∪ [a := 1]
  return α
```

Pure literals and subsumption

Other simplification steps in original DPLL:

- elimination of pure literals
- deletion of subsumed clauses

`PureLit(`F`,`α`)`

`while` $F\alpha$ `contains pure literal` a

$\alpha := \alpha \cup [a := 1]$

Clause C subsumes D if $C \subseteq D$

`Subs(`F`)`

`while` F `contains clauses` $C \subseteq D$

$F := F \setminus D$

The general DPLL algorithm

```
DPLL(F, α)
  simplify(F, α)
  if F = 0 then return UNSAT
  if F = 1 then return α
  pick x ∈ V(F) and ϵ ∈ {0, 1}
  β := DPLL(F[x := ϵ], α ∪ [x := ϵ])
  if β ≠ UNSAT
     then return β
     else return DPLL(F[x := ϵ̄], α ∪ [x := ϵ̄])
```

Monien-Speckenmeyer algorithm - simple version

Branching strategy:

- pick literal from a shortest clause

Equivalently:

simpleMS(F, α)

  if $F\alpha = 0$ then return UNSAT
  if $F\alpha = 1$ then return α

  pick shortest clause $C = a_1 \vee \ldots \vee a_r$

  for $i := 1$ to r do

    $\gamma_i := [a_1 := 0, \ldots, a_{i-1} := 0, a_i := 1]$
    $\beta :=$ simpleMS($F, \alpha \cup \gamma_i$)
    if $\beta \neq$ UNSAT then return β

  return UNSAT

Analysis of `simpleMS`

At each node: r-fold branching for some $r \leq k$,

in ith branch $n - i$ variables.

Recursion for the tree-size:

$$T(n) := T(n-1) + \ldots + T(n-k)$$

Set $T(n) := b^n$, thus the basis $b_k = b$ satisfies

$$b^k := b^{k-1} + \ldots + b + 1$$

The solutions for small k are:

k	3	4	5	6	7	8
b_k	1.83929	1.92757	1.96595	1.98359	1.99197	1.99603