Contents

1 Resolution Exercises

SAT via Resolution

function sat_resolution(ϕ: formula)
{
cnf F := transform_to_cnf(ϕ);
cnf $Fold$;
while(C_1 and C_2 resolveable on A exist in F
and have not been resolved yet)
{
F := resolve(F,C_1,C_2,A);
if($\Box \in F$)
return false;
}
return true;
}

Resolution in propositional logic: Examples

Example 1. 1 Is $(A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee B) \wedge (\neg A \vee \neg B)$ satisfiable?

Example 2. 2 Does A follow from $(A \vee B \vee C) \wedge (\neg C \vee B) \wedge (A \vee \neg B)$?

Example 3. 3 Does $\neg A$ follow from $(A \vee B \vee C) \wedge (\neg C \vee B) \wedge (A \vee \neg B)$?

Example 4. 4 Does $P = A \wedge B$ follow from $(\neg A \rightarrow B) \wedge (A \rightarrow B) \wedge (\neg A \rightarrow \neg B)$?

Optimizations to resolution algorithm

Problem
The algorithm can generate many irrelevant or redundant clauses

Example 5. S = $\{\{A, B\}, \{\neg A, B\}, \{A, \neg B\}, \{\neg A, \neg B\}\}$

Solution
At each step

- Delete tautological clauses
- Delete clauses already generated
- Delete "subsumed" clauses ($\{B\}\ vs\ \{A, B\}$)

Linear resolution

Definition
A resolution proof for R from a set S of clauses is *linear* if it is a sequence $C_1, \dots, C_n$ s.t. $C_1 \in S, C_n = R$ and for each $i = 2, \dots, n$, C_i is the resolvent of C_{i-1} and B_{i-1}, with $B_{i-1} \in S$ or $B_{i-1} = C_j$, with $j < i$.

Intuition...
In a proof for linear resolution, at each step the resolvent obtained in the previous step is used.

Example 6. S = $\{A \vee B, A \vee \neg B, \neg A \vee B, \neg A \vee \neg B\}$

2 Translation to CNF

Problems with Distributivity
Example:

$$
\begin{aligned}
&(A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \cdots \vee (A_n \wedge B_n) \equiv \\
&(A_1 \vee [(A_2 \wedge B_2) \vee \cdots \vee (A_n \wedge B_n)]) \wedge \\
&(B_1 \vee [(A_2 \wedge B_2) \vee \cdots \vee (A_n \wedge B_n)]) \equiv \\
&(A_1 \vee A_2 \vee [(A_3 \wedge B_3) \vee \cdots \vee (A_n \wedge B_n)]) \wedge \\
&(A_1 \vee B_2 \vee [(A_3 \wedge B_3) \vee \cdots \vee (A_n \wedge B_n)]) \wedge \\
&(B_1 \vee A_2 \vee [(A_3 \wedge B_3) \vee \cdots \vee (A_n \wedge B_n)]) \wedge \\
&(B_1 \vee B_2 \vee [(A_3 \wedge B_3) \vee \cdots \vee (A_n \wedge B_n)]) \equiv \\
&\ldots \equiv \\
&(A_1 \vee \cdots \vee A_n) \wedge (A_1 \vee \cdots \vee A_{n-1} \vee B_n) \wedge \cdots \wedge (B_1 \vee \cdots \vee B_n)
\end{aligned}
$$

Problems with Distributivity
In the example, we went from a formula with $2 \times n$ variable occurrences and $2 \times n - 1$ connectives to a formula with 2^n variable occurrences and $2^n - 1$ connectives!

Theorem 7. *Transforming a formula into an equivalent formula in CNF may cause an exponential enlargement.*

But one can do better.

Structure-Preserving Transformation (S-PT)

Algorithm

1. Replace each "sub-formula" with a newly introduced variable

2. Convert into CNF all the resulting formulas

Pros

- linear time transformation
- no exponential blow up in space

Cons

- add new variables (even if linear in the size of the input)

Structure-Preserving Transformation (S-PT)

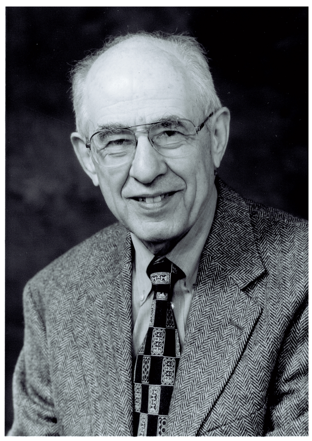
Example:

$$\overbrace{(A_1 \wedge B_1)}^{C_1} \vee \overbrace{(A_2 \wedge B_2)}^{C_2} \vee \cdots \vee \overbrace{(A_n \wedge B_n)}^{C_n} \text{ satisfiable iff}$$
$$(C_1 \vee \cdots \vee C_n) \wedge (C_1 \leftrightarrow (A_1 \wedge B_1)) \wedge \cdots \wedge (C_n \leftrightarrow (A_n \wedge B_n)) \equiv$$
$$(C_1 \vee \cdots \vee C_n) \wedge$$
$$(C_1 \vee \neg(A_1 \wedge B_1)) \wedge (\neg C_1 \vee (A_1 \wedge B_1))$$
$$\wedge \cdots \wedge$$
$$(C_n \vee \neg(A_n \wedge B_n)) \wedge (\neg C_n \vee (A_n \wedge B_n)) \equiv$$
$$(C_1 \vee \cdots \vee C_n) \wedge$$
$$(C_1 \vee \neg A_1 \vee \neg B_1) \wedge (\neg C_1 \vee A_1) \wedge (\neg C_1 \vee B_1)$$
$$\wedge \cdots \wedge$$
$$(C_n \vee \neg A_n \vee \neg B_n) \wedge (\neg C_n \vee A_n) \wedge (\neg C_n \vee B_n)$$

The CNF contains $8 \times n$ variable occurrences and $12 \times n - 1$ connectives. In general, this transformation is more involved.

3 DPLL

Davis, Putnam, Logemann, Loveland



Martin Davis, Hilary Putnam "A Computing Procedure for Quantification Theory" Journal of the ACM, 1960

Davis, Putnam, Logemann, Loveland

Martin Davis, George Logemann, Donald Loveland "A Machine Program for Theorem-Proving" Communications of the ACM, 1962

DPLL algorithm

- Γ is a set of clauses (CNF formula)
- U is the set of literals representing a partial truth assignment (initialized with $\emptyset$)

DPLL(Γ, U)
1 **if** $\{l\} \in \Gamma$ **then** SIMPLIFY(Γ,l)
2 **if** $\Gamma = \emptyset$ **then return** TRUE
3 **if** $\emptyset \in \Gamma$ **then return** FALSE
4 $l \leftarrow$ CHOOSE-LITERAL(Γ)
5 **return** DPLL($\Gamma \cup \{l\}$, U) **or**
DPLL($\Gamma \cup \{\neg l\}$, U)

SIMPLIFY

SIMPLIFY(Γ, U)
while $\{l\} \in \Gamma$ **do**
  $U \leftarrow U \cup \{l\}$
  foreach $c \in \Gamma$ **do**
    if $l \in c$ **then**
      $\Gamma \leftarrow \Gamma \setminus \{c\}$
    else if $\neg l \in c$ **then**
      $\Gamma \leftarrow (\Gamma \setminus \{c\}) \cup \{c \setminus \neg l\}$

DPLL properties

1. DPLL(Γ,U) returns TRUE iff Γ is satisfiable, and *False* otherwise
2. DPLL(Γ,U) can be (easily) modified in order to compute all the solutions of Γ (DPLL is *correct* and *complete*)
3. DPLL(Γ,U) works in polynomial-space

Features in the DPLL method

- *Simplification*: the input set of clauses is simplified at each branch using (at least) unit clause propagation
- *Branching*: when no further simplification is possible, a literal is selected using some heuristic criterion and assumed as a unit clause in the current set of clauses
- *Backtracking*: when a contradiction (empty clause) arises, then the search resumes from some previous assumption l by assuming $\neg l$ instead

Examples with DPLL (I)

Example 8. $\Gamma = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_4 \vee \neg x_3)$

1. x_2 is assigned to 0, Γ is simplified and results in $(x_1 \vee \neg x_3) \wedge (x_4 \vee \neg x_3)$, $U = \{\neg x_2\}$
2. we choose $\neg x_3$ for branching, the formula is SAT, $\Gamma = \emptyset$, $U = \{\neg x_2, \neg x_3\}$

Example 9. $\Gamma = (x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_2) \wedge (x_4 \vee \neg x_3)$

1. x_2 is assigned to 0, Γ is simplied and results in $(x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_4 \vee \neg x_3)$, $U = \{\neg x_2\}$
2. if we choose now $\neg x_3$ for branching, the process is the same as before; otherwise, if x_1 is chosen, another choice has to be made

Branching matters!

Examples with DPLL (II)

Example 10. $\{x_1 \vee x_2 \vee x_3, x_1 \vee x_2 \vee \neg x_3, x_1 \vee \neg x_2 \vee x_3, x_1 \vee \neg x_2 \vee \neg x_3, \neg x_1 \vee x_4, x_1 \vee \neg x_4 \vee \neg x_5 \vee x_6, \neg x_1 \vee x_7\}$

(Some of) The major players (I)

Before 2K

Böhm, Tableau Inspired early work on SAT solvers and started gaining popularity for applications

POSIT, SATZ Effective proof-of-concept implementations

Grasp, SATO, RelSAT First application-targeted solvers

- Intelligent backtracking techniques
- Efficient data structures (SATO)
- Learning techniques (Grasp, RelSAT)

(Stålmarck) Patented method, first industrial SAT solver

(Some of) The major players (II)

After 2K

Chaff (mChaff, zChaff) turning point in the applicability of SAT

- borrows from the tradition of SATO and Grasp
- very effective on "structured" instances
- first SAT solver to conquer "hard" instances from model checking and planning domains
- winner of the SAT 2002 competition (industrial category)

SatEliteGTI/MiniSAT (winner of SAT2005, industrial category)

- mostly "Chaff-based" ...

Key technologies in today's DPLL implementations (II)

Current state-of-the-art (SOTA) solvers can be divided in two categories:

- "look-ahead" solvers, with a powerful simplification procedure, a simple look-back (essentially backtracking) and a heuristic based on the information gleaned during the look-ahead phase. Best for "small but relatively difficult" instances, typically randomly generated.
- "look-back" solvers, with a simple but efficient look-ahead, a sophisticated look-back based on "learning", and a constant time heuristic based on the information gleaned during the look-back phase. Best for "large but relatively easy" instances, typically encoding real-world problems.

SAT-solvers input DIMACS format

"Official" input format for SAT-solvers. Each solver has to comply with it for enter in the competition. Ex: $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2) \wedge (x_4 \vee \neg x_3)$

Example 11.

$$\begin{array}{l} c\, This\; is\; a\; CNF\; in\; DIMACS \\ c \\ p\; cnf\; 4\; 3 \\ 1\; 2\; \text{-}3\; 0 \\ \text{-}2\; 0 \\ 4\; \text{-}3\; 0 \end{array}$$

DIMACS format in BNF grammar

BNF grammar

$$\begin{array}{l}
< input > ::= < preamble > < formula > EOF \\
\\
< preamble > ::= \; [< commentlines >] < problemline > \\
< commentlines > ::= < commentline > < commentlines > \mid < commentline > \\
< commentline > ::= \; c \; < text > EOL \\
< problemline > ::= \; p\, cnf < pnum > < pnum > EOL \\
\\
< formula > ::= < clauselist > \\
< clauselist > ::= < clause > < clauselist > \mid < clause > \\
< clause > ::= < literal > < clause > \mid < literal > 0 \\
< literal > ::= < num > \\
\\
< text > ::= \; A\; sequence\; of\; non-special\; ASCII\; characters \\
< pnum > ::= \; A\; signed\; integer\; greater\; than\; 0 \\
< num > ::= \; A\; signed\; integer\; different\; from\; 0
\end{array}$$

Challenges and ongoing work

Hot topics

- Incremental SAT
- Non clausal SAT
- SAT-based decision procedures

More on:

- `http://www.satlive.org/`
- `http://www.satisfiability.org/`

In preparazione all'esercitazione...

1. Scrivere su google "Minisat solver"
2. Cliccare sul primo link
3. Sulla barra di sinistra cliccare su "MiniSat"
4. Scaricare "$MiniSat\_v1.14\_linux$" in fondo alla pagina (frame principale)
5. Provare ad eseguire "$./MiniSat\_v1.14\_linux -h$"(ove necessario, modificare i diritti sul file, e.g., "$chmod\,755\,MiniSat\_v1.14\_$