Boolean Satisfiability

Gianpiero Cabodi *Stefano Quer*

Politecnico di Torino

Torino, Italy

{gianpiero.cabodi,stefano.quer}@polito.it

http://staff.polito.it/{gianpiero.cabodi,stefano.quer}/

References

- Resolution
  - Davis & Putnam, JACM'60
- Backtrack Search
  - Davis et. al, CACM'62
  - Non-chronological backtracking and clause recording
    - Marques-Silva & Sakallah, ICCAD'96
    - Bayardo & Schrag, AAAI'97; Zhang, CADE'97
  - Relevance-based learning
    - Bayardo & Schrag, AAAI'97
  - Conflict-induced necessary assignments
    - Marques-Silva & Sakallah, ICCAD'96
  - Randomization and restarts
    - Gomes & Selman, AAAI'98
    - Baptista & Marques-Silva, CP'2000
  - Formula simplification
    - Li, AAAI'2000; Marques-Silva, CP'2000
- Stalmarck's Method
  - Stalmarck, Patent'89
  - Groote & Warners, CWI TechRep'1999
- Recursive Learning
  - Kunz & Pradhan, ITC'92
  - Marques-Silva & Glass, DATE'99
- Local Search
  - Selman & Kautz, IJCAI'93

Outline

- Boolean Satisfiability (SAT)
- Basic Algorithms
  - DP
  - DPLL
- Circuit SAT
  - Gate-to-CNF
  - Circuit-to-CNF
  - SAT on Circuit
  - DIMACS files

Boolean Satisfiability (SAT)

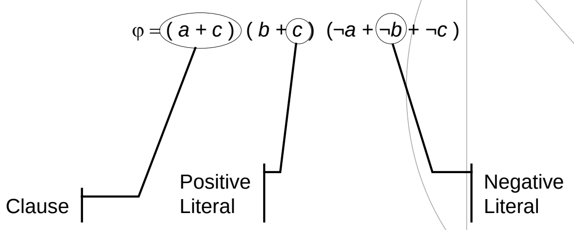
- Given a suitable representation for a Boolean function *f* (*X*)
  - Find an assignment *X** such that *f* (*X**) = 1

  OR
  - *Prove that such an assignment does not exist*, i.e., *f*(*X*) = 0 for all possible assignments
- In the "classical" SAT problem, *f* (*X*) is represented as
  - Product-of-sums (POS)

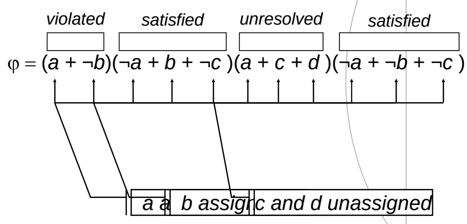
  OR
  - Conjunctive normal form (CNF)

- SAT belongs to NP
  - There is a non-deterministic Touring Machine deciding SAT in polinomial time
  - On a real – deterministic computer this would require exponential time
- Many decision (yes/no) problems can be formulated either directly or indirectly in terms of Boolean Satisfiability

Conjunctive Normal Form (CNF)



Literal & Clause Classification



Davis-Putnam (DP) Procedure

- ❖ **Search for consistent assignment to entire cone of requested vertex by systematically trying all combinations (may be partial!!!)**
- ❖ **Keep a queue of vertices that remain to be justified**
  - ◆ **Pick decision vertex from the queue and case split on possible assignments**
  - ◆ **For each case**
    - ✧ **Propagate as many implications as possible**
      - – generate more vertices to be justified
      - – if conflicting assignment encountered undo all implications and backtrack
    - ✧ **Recur to next vertex from queue**

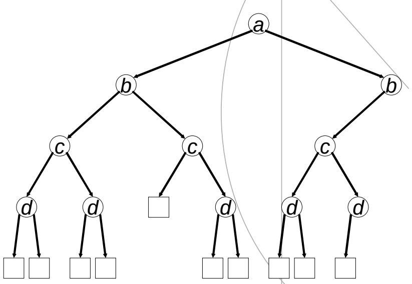
Davis Putnam Logemann Loveland (DPLL) Procedure

```
Algorithm DPLL ()
  while (ChooseNextAssignment ())
     while (Deduce() == CONFLICT)
        bLevel = AnalyzeConflict ()
        if (bLevel < 0) return (UNSATISFIABLE)
        else Backtrack (bLevel)
  return (SATISFIABLE)
```

- ❖ ChooseNextAssignment
  - ◆ picks next decision variable and assignment
- ❖ Deduce
  - ◆ does Boolean Constraint Propagation (BCP - implications)
- ❖ AnalyzeConflict
  - ◆ back-processes from conflict and learns clauses
- ❖ Backtrack
  - ◆ un-does assignments

Basic Case Splitting (Backtrack Search)

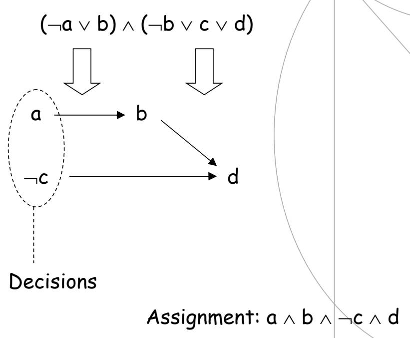
1 (*a* + *b* + *c*)
2 (*a* + *b* + ¬*c*)
3 (¬*a* + *b* + ¬*c*)
4 (*a* + *c* + *d*)
5 (¬*a* + *c* + *d*)
6 (¬*a* + *c* + ¬*d*)
7 (¬*b* + ¬*c* + ¬*d*)
8 (¬*b* + ¬*c* + *d*)



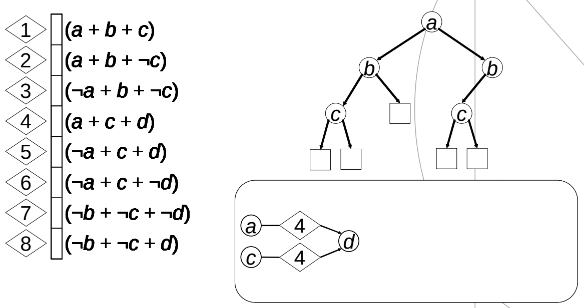
Implications: Unit Clause Rule

- ❖ **An unresolved clause is *unit* if it has exactly one unassigned literal**
  - ◆ **A unit clause has exactly one option for being satisfied**
  - ◆ **The value of that clause can be implied immediately**
  - ◆ **Example**
    - ✧ **(*a* + *c*)(*b* + *c*)(¬a + ¬b + ¬c)**
    - ✧ ***a b* ⇒ ¬*c***
    - ✧ **i.e., a=1 and b=1 imply that c must be set to 0.**

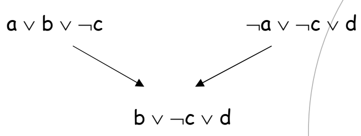
The Implication Graph (BCP)



Basic Case Splitting with Implications

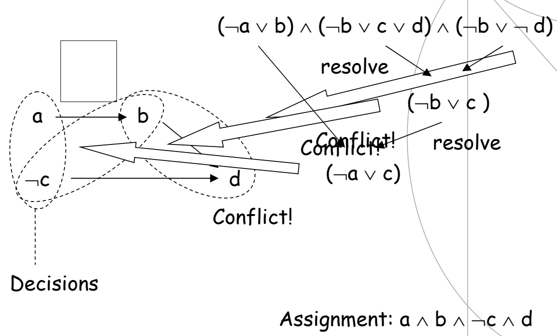


Resolution



When a conflict occurs, the implication graph is used to guide the resolution of clauses, so that the same conflict will not occur again.

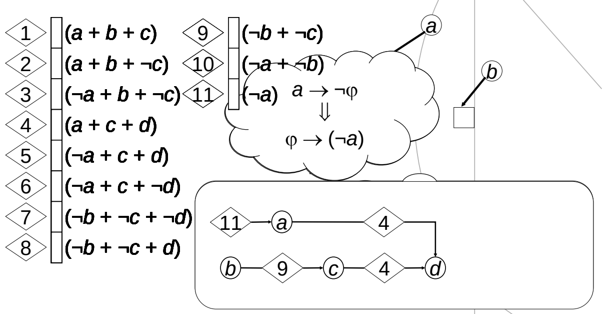
Conflict Clauses



- **Conflict clauses**
  - **Are generated by resolution**
  - **Are implied by existing clauses**
  - **Are in conflict in the current assignment**
  - **Are safely added to the clause set**

Many heuristics are available for determining when to terminate the resolution process

Conflict-based Learning



A SAT Example: Optimization of if-then-else chain

Original C code

```
if (!a && !b) h();
else if (!a) g();
else f();
```

```
if (!a) {
  if (!b) h();
  else g();
} else f();
```

Optimized C code

```
if (a) f();
else {
  if (!b) h();
  else g();}
```

```
if (a) f();
else if (b) g();
else h();
```

How to check if these are equivalent?

- **Represent procedures as independent Boolean variables**

```
Original = if (¬a ∧ ¬ b) h();
           else if (¬a) g();
           else f();
Optimized = if (a) f();
            else {
              if (¬b) h();
              else g();}
```

- **Compile the if-then-else chains into Boolean formulae**

```
if x then y else z = ITE (x,y,z)
                   =(x ∧ y) (¬x ∧ z)
```

- **Check equivalence of Boolean formulae**

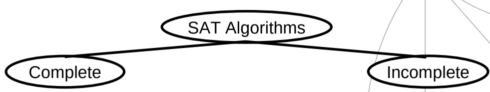
```
Compile (Original) ≡ Compile (Optimized)
```

```
Original = if ¬a∧¬b then h else if ¬a then g else f
  = (¬a ∧ ¬b) ∧ h ∨ ¬(¬a ∧ ¬b) ∧
      if ¬a then g else h
  = (¬a ∧ ¬b) ∧ h ∨ ¬(¬a ∧ ¬b) ∧ (¬a ∧ g ∨ a ∧ f)
Optimized = if a then f else if b then g else h
  = (a ∧ f) ∨ ¬a ∧ if b then g else h
  = a ∧ f ∨ ¬a ∧ (b ∧ g ∨ ¬b ∧ h)
```

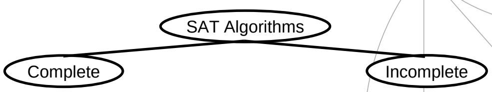
```
(¬a ∧ ¬b) ∧ h ∨ ¬(¬a ∧ ¬b) ∧ (¬a ∧ g ∨ a ∧ f)
                      ≠
          a ∧ f ∨ ¬a ∧ (b ∧ g ∨ ¬b ∧ h)
```

is satisfiable?

A Taxonomy of SAT Algorithms



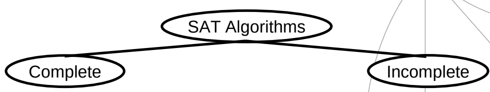
A Taxonomy of SAT Algorithms



Can prove unsatisfiability

Cannot prove unsatisfiability

A Taxonomy of SAT Algorithms



Can prove unsatisfiability

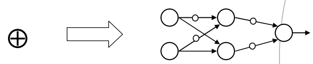
- Backtrack search (DP)
- Resolution (original DP)
- Stalmarck's method (SM)
- Recursive learning (RL)
- BDDs
- ...

Cannot prove unsatisfiability

- Local search (hill climbing)
- Continuous formulations
- Genetic algorithms
- Simulated annealing
- Tabu search
- ...

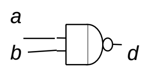
Circuit to CNF

- Naive conversion of circuit to CNF
  - Multiply out expressions of circuit until two level structure
  - Example
    - $y = x_1 \oplus x_2 \oplus x_2 \oplus \dots \oplus x_n$ (Parity function)
    - Circuit size is linear in the number of variables



    - Generated chess-board Karnaugh map
    - CNF (or DNF) formula has 2^{n-1} terms (exponential in the # vars)
- Better approach
  - Introduce one variable per circuit vertex
  - Formulate the circuit as a conjunction of constraints imposed on the vertex values by the gates
  - Uses more variables but size of formula is linear in the size of the circuit

Gate To CNF



$$\varphi_x = [d = \neg(a \wedge b)]$$
$$= \neg[d \oplus \neg(a \wedge b)]$$
$$= \neg[\neg(a \wedge b) \wedge \neg d + a \wedge b \wedge d]$$
$$= \neg[\neg a \wedge \neg d + \neg b \wedge \neg d + a \wedge b \wedge d]$$
$$= (a + d)\wedge(b + d)\wedge(\neg a + \neg b + \neg d)$$

$$\varphi_x = [d = \neg(a \wedge b)]\wedge[\neg d = (a \wedge b)]$$
$$= [d = (\neg a + \neg b)]\wedge[\neg d = (a \wedge b)]$$
$$= (\neg a \rightarrow d)(\neg b \rightarrow d)(a\ b \rightarrow \neg d)$$
$$= (a + d)(b + d)(\neg a + \neg b + \neg d)$$

- y = AND $(x_1, \dots, x_i)$

  $\varphi_x = [\ \prod_i (x_i \vee \neg y)] \wedge [\ \sum_I \neg x_i \vee y\ ]$
- y = NAND $(x_1, \dots, x_i)$

  $\varphi_x = [\ \prod_i (x_i \vee y)] \wedge [\ \sum_I \neg x_i \vee \neg y\ ]$
- y = OR $(x_1, \dots, x_i)$

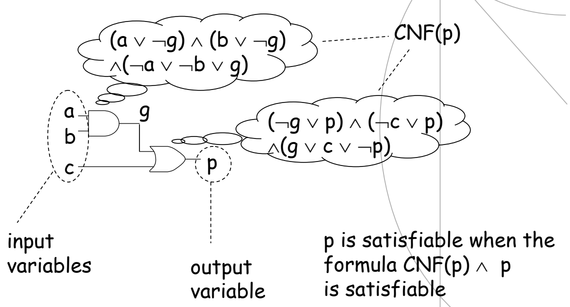
  $\varphi_x = [\ \prod_i (\neg x_i \vee y)] \wedge [\ \sum_i x_i \vee \neg y\ ]$
- y = NOR $(x_1, \dots, x_i)$

  $\varphi_x = [\ \prod_i (\neg x_i \vee \neg y)] \wedge [\ \sum_i x_i \vee y\ ]$
- y = NOT (x)

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

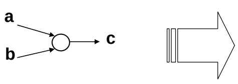
Circuit SAT

Can the circuit output be 1?



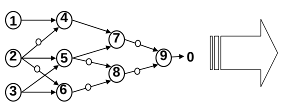
p is satisfiable when the formula CNF(p) ∧ p is satisfiable

Exercise



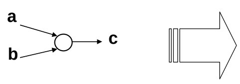
A single gate

$(\neg a \vee \neg b \vee c) \wedge (a \vee \neg c) \wedge (b \vee \neg c)$



?

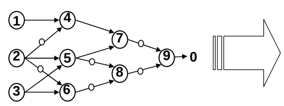
Exercise



A single gate

$(\neg a \vee \neg b \vee c) \wedge (a \vee \neg c) \wedge (b \vee \neg c)$

A circuit

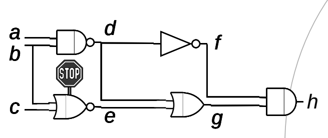


$(\neg 1 \vee 2 \vee 4) \wedge (1 \vee \neg 4) \wedge (\neg 2 \vee \neg 4) \wedge$

$(\neg 2 \vee \neg 3 \vee 5) \wedge (2 \vee \neg 5) \wedge (3 \vee \neg 5) \wedge$

$(2 \vee \neg 3 \vee 6) \wedge (\neg 2 \vee \neg 6) \wedge (3 \vee \neg 6) \wedge$

An Example



$\varphi = h\ [d{=}\neg(ab)]\ [e{=}\neg(b{+}c)]\ [f{=}\neg d]\ [g{=}d{+}e]\ [h{=}fg]$

$= h$

$(a + d)(b + d)(\neg a + \neg b + \neg d)$

$(\neg b + \neg e)(\neg c + \neg e)(b + c + e)$

$(\neg d + \neg f)(d + f)$

$(\neg d + g)(\neg e + g)(d + e + \neg g)$

$(f + \neg h)(g + \neg h)(\neg f + \neg g + h)$

From CNF to DIMACS File

- **CNF**

  `(a ∨ d) ∧ (¬b ∨ d) ∧ (¬a ∨ b ∨ ¬c)`

- **Variable coding**

  `a → 1, b → 2, c → 3, d → 4`

- **DIMACS format for CNF formulae**

```
p cnf 4 3
1 4 0
-2 4 0
-1 2 -3 0
```