Il problema SAT

Prof. Marco Gavanelli

The Boolean Satisfiability problem (SAT)

- A boolean function (or formula) F

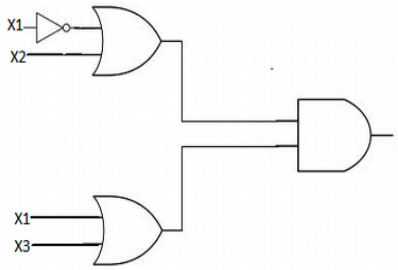
  $(X1 \vee X2 \vee \neg X3) \wedge (\neg X1 \vee X2 \vee \neg X3)$

- Sometimes printed as

  $(X1 + X2 + X3') * (X1' + X2 + X3')$

- Are there assignments to X1, X2, and X3 that make the function TRUE?

- If there are, the function is Satisfiable.



SAT

- Le funzioni booleane vengono date in Conjunctive Normal Form (CNF)
- Una congiunzione di disgiunzioni
- Ogni disgiunzione viene chiamata clausola
- Ogni clausola è l'OR di un insieme di letterali
- Ogni letterale può essere una variabile o una variabile negata

Importanza

- Il problema SAT venne studiato nell'ambito della logica / automatic theorem proving
- Ha una particolare importanza nell'ambito della complessità computazionale
- Lo si può usare per codificare altri tipi di problemi
- In ambito SAT sono stati sviluppati algoritmi efficienti, che poi sono stati portati anche in ambito CSP

Complexity

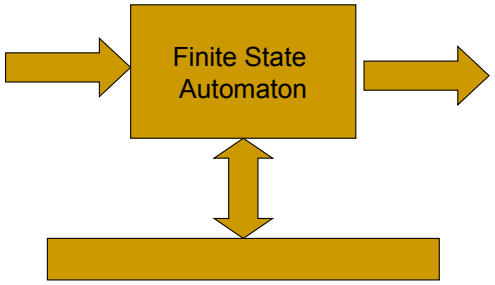
- *We will study (asymptotic) complexity in the worst-case: O(f(n)). Costs:*
  - *Constant*
  - *Log*
  - ***Polynomial*** *(tractable)*
  - ***Exponential****, …* *(intractable)*
- *Which resource do we choose to measure complexity?*
  - *Time,*
  - *Space,*
  - *…*
- *Which machine do we choose?*
  - *Turing machine, RAM, PRAM, …*
- *Which representation of the input? s(s(s(0))) or 011 ?*

Decision Problems

- *The basic issue is solving decision problems, i.e., problems that can have answers "Yes" or "No".*
- *E.g.: is there a TSP with cost lower than X?*

Machines: Turing Machine

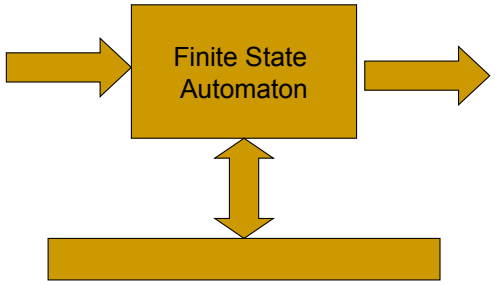
- *3 tapes:*
  - *Input (read only)*
  - *Output (write only)*
  - *Data (R/W)*
- *The 3 tapes are important for SPACE complexity:*
  - *if the input is on the Data tape, then sublinear space complexity is meaningless*



Machines: Turing Machine

The TM can

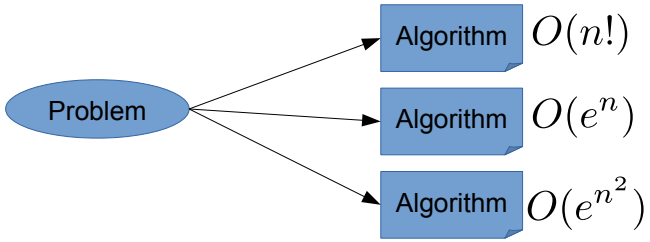
- *Loop*
- *Terminate, executing a HALT instruction*
  - *With ACCEPT*
  - *With NOT ACCEPT*



(The term "Accept" comes from the fact that formally each decision problem is encoded with a string, and it is considered solved if the string belongs to some language)

Complexity of a problem

The complexity of a problem is the complexity of the best algorithm that can solve the problem



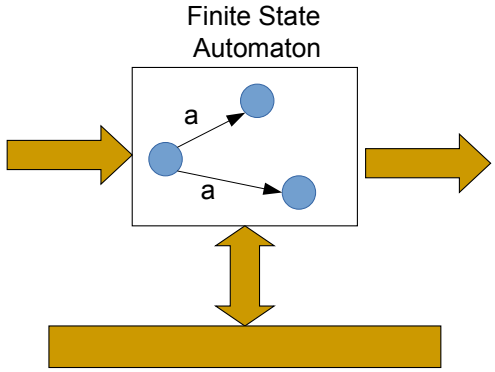
… but what about future discoveries?

What if we don't know?

- *What if we cannot prove that a problem is in a class?*
  - *E.g., nobody found a Polynomial algorithm, but nobody proved that it is impossible to solve it with a Polynomial algorithm?*

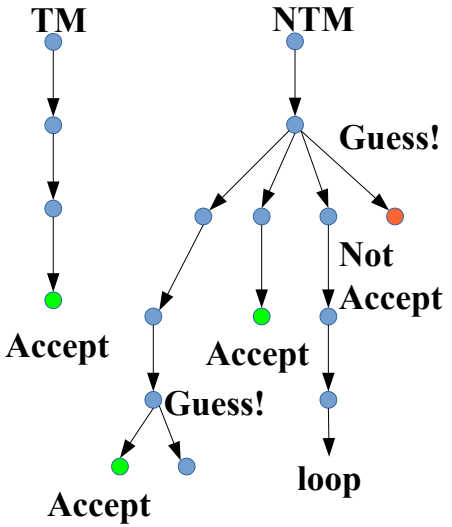
Nondeterministic Turing Machine

- *A nondeterministic Turing Machine is a TM in which the next state is not "unique".*



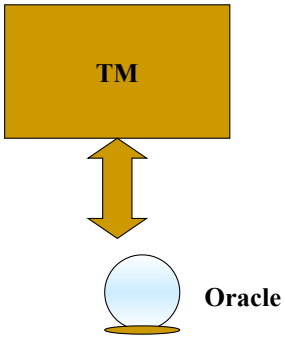
Det. & Non-Det. TM

- *While the TM has a unique flow, the NTM has a tree of flows*
- *The NTM accepts iff one of the branches is labelled Accept, and rejects if all reject*
- ***<u>In case of success</u>**, the computing time is the time required by the shortest branch*



Machines: Nondeterministic Turing Machine

- A nondeterministic Turing machine can also be thought as a TM equipped with an oracle, which is able to "guess"
  - E.g., it can guess values taken from a finite or infinite set



Nondet. Turing Machine: example

- CSP
- Algorithm:
  - Guess an assignment to all variables
  - If all constraint are satisfied, return ACCEPT
  - Otherwise, return NOT ACCEPT
- Thus, CSP is polynomially solvable by a nondeterministic Turing machine

O(n) if n is the number of variables

We assume each constraint can be checked in polynomial time

First notation

- To start with, complexity classes are defined as
  [Machine] [Cost] [Resource]
- Machine:
  - N for nondeterministic TM
  - (nothing) for deterministic TM
- Cost:
  - LOG: O(log)
  - P: O(Polynomial)
  - EXP: O(Exponential)
- Resource:
  - (nothing): Time
  - SPACE: Space

- Eg, SAT is in NP, NPSPACE
- Of course,

$$P \subseteq NP$$

NP

- *NP is the set of all problems in which it is easy (polynomial) to check if an assignment is indeed a solution*
- *Eg.:*
  - *SAT*
  - *Hamiltonian path*
  - *Bounded TSP: Is there a solution that costs less than B?*
  - Graph coloring
  - N-queens, …
- The query is usually existential:

$$\exists X_1, X_2, \ldots \quad \text{s.t.} \quad P(X_1, X_2, \ldots)$$

Switching to determinism?

- *We can, of course, say that we can build the oracle through a backtracking algorithm, so*

$$SAT \in EXP$$

- *Space complexity?*
- *The backtracking algorithm takes $O(n^2)$ space*

$$SAT \in PSPACE$$

- *but this is not very precise, …*
- *One day, we could find an algorithm for solving SAT polynomially (?!?), so we are not sure if*
  $SAT \in EXP \setminus P$ or $SAT \in P$

Teorema di Cook [1971]

- *Cook [1971] dimostra che qualunque problema in NP può essere polinomialmente ridotto al problema SAT*
- *Quindi, se si trova un algoritmo polinomiale per risolvere il problema SAT, è possibile risolvere in tempo polinomiale qualunque problema in NP*
- *Si dice che SAT è NP-difficile (NP-Hard).*
- *Inoltre, SAT è in NP. Si dice che SAT è NP-completo*

Davis Putnam (DP) 1960

- Elimina clausole con solo un letterale (unit propagation)
  - Se *F* contiene le clausole {*p*} e {*p'*}, allora è *FALSA*
  - Se *F* contiene la clausola {*p*}, allora
    - Elimina tutte le clausole che contengono *p*
    - elimina *p'* da tutte le altre clausole
  - Se la formula è vuota, allora è SAT
- Letterale puro (valida per soddisfacibilità, non per equivalenza)
  - Se una formula contiene *p* e non *p'*, allora assegna *p=true*
- Eliminazione di variabili tramite risoluzione
  - Se *F* contiene *n* clausole con *p* e *m* clausole con *p'*, elimina le (*m+n*) clausole e produci nuove *mn* clausole:
    per ogni coppia $p \vee A$ e $p' \vee B$, produci una clausola $A \vee B$

Problemi di DP

- DP è stato il primo algoritmo per risolvere problemi SAT
- richiede spazio esponenziale, che lo rende in pratica inutilizzabile, se non per formule piccole
- L'evoluzione è un algoritmo di search

Davis, Logemann, Loveland (DLL) 1962

DPLL(F, U)

Unit-propagate(F, U);

if F contains the empty clause then return FALSE;

if $F = \top$ then exit with a model of U ;

$L \leftarrow$ a literal containing an atom from F ;

DPLL($F|_L$, $U \cup \{L\}$);

DPLL($F|_{L'}$, $U \cup \{L'\}$)

DPLL

- L'algoritmo DPLL è un algoritmo di search:
  - Si seleziona un letterale e lo si rende vero (in backtracking si cambia il suo valore "flip")
  - Si propagano le conseguenze di questo assegnamento
  - Terminata la propagazione, si può avere un conflitto o successo
  - Se si ha conflitto, si fa backtracking
  - Se si ha successo, si seleziona un altro letterale non assegnato (se non ne esistono → successo)

Unit Propagation (UP)

- La propagazione principale effettuata dall'algoritmo DPLL è la unit propagation
- Può essere implementata modificando il database delle clausole
  - Quando un letterale p = vero
    - Elimino tutte le clausole in cui compare p
    - Elimino -p da tutte le clausole in cui compare
  - In tal caso, quando una clausola contiene un solo letterale, si impone che quel letterale sia vero
- Oppure non si modificano le clausole nel database, ma si considera un assegnamento corrente
  - Se nell'assegnamento corrente una clausola ha tutti i letterali falsi eccetto uno → si impone che quel letterale sia vero

Note

- *Si può immaginare il SAT come CSP, dove le clausole sono implementate come vincoli*
- *L'algoritmo DPLL è equivalente a fare arc-consistency + search*

Conflict Directed Clause Learning (CDCL)

- *I risolutori moderni cercano di apprendere nuove informazioni quando arrivano ad un fallimento*
- *Le nuove informazioni sono nuovamente codificate in forma a clausole e aggiunte al database*
- *In questo modo, nel resto della ricerca si potranno evitare rami inutili*
- *Inoltre, il backtracking non è più cronologico: non si torna all'ultima scelta fatta, ma si all'ultima scelta che è responsabile del conflitto (backjumping)*
- *Tesi di dottorato di Joao Marques-Silva*

CDCL solver

```
status = preprocess();
if (status!=UNKNOWN) return status;
while (Status!= UNKNOWN)
  { decide_next_branch();
    status = deduce();
     if (status == CONFLICT)
     {  blevel = analyze_conflict();
        if (blevel == 0)
           return UNSATISFIABLE;
           else backtrack(blevel);
      }
     else if (status == SATISFIABLE)
                return SATISFIABLE;
  }
```


Implication graphs and learning

Current truth assignment: $\{x_9=0@1, x_{10}=0@3, x_{11}=0@4, x_{12}=1@2, x_{13}=1@5\}$

Current decision assignment: $\{x_1=1@9\}$

$\omega_1 = (\neg x_1 \vee x_2)$

$\omega_2 = (\neg x_1 \vee x_3 \vee x_9)$

$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$\omega_4 = (\neg x_4 \vee x_5 \vee x_{10})$

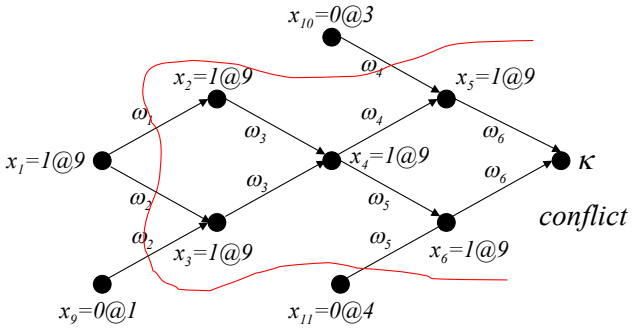
$\omega_5 = (\neg x_4 \vee x_6 \vee x_{11})$

$\omega_6 = (\neg x_5 \vee \neg x_6)$

$\omega_7 = (x_1 \vee x_7 \vee \neg x_{12})$

$\omega_8 = (x_1 \vee x_8)$

$\omega_9 = (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$



We learn the *conflict clause* ω_{10} : $(-x_1 \vee x_9 \vee x_{11} \vee x_{10})$

and backtrack to the highest (deepest) dec. level in this clause (9).

Implication graph, flipped assignment

Current truth assignment: $\{x_9=0@1, x_{10}=0@3, x_{11}=0@4, x_{12}=1@2, x_{13}=1@5\}$

Current decision assignment: $\{x_1=1@9\}$

$\omega_1 = (\neg x_1 \vee x_2)$

$\omega_2 = (\neg x_1 \vee x_3 \vee x_9)$

$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$

$\omega_4 = (\neg x_4 \vee x_5 \vee x_{10})$

$\omega_5 = (\neg x_4 \vee x_6 \vee x_{11})$

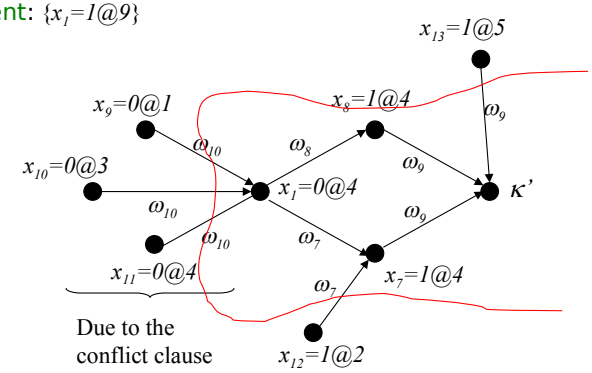
$\omega_6 = (\neg x_5 \vee x_6)$

$\omega_7 = (x_1 \vee x_7 \vee \neg x_{12})$

$\omega_8 = (x_1 \vee x_8)$

$\omega_9 = (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$

ω_{10} $(\neg x_1 \vee x_9 \vee x_{11} \vee x_{10})$



We learn the *conflict clause* ω_{11} : $(-x_{13} \vee x_9 \vee x_{10} \vee x_{11} \vee -x_{12})$

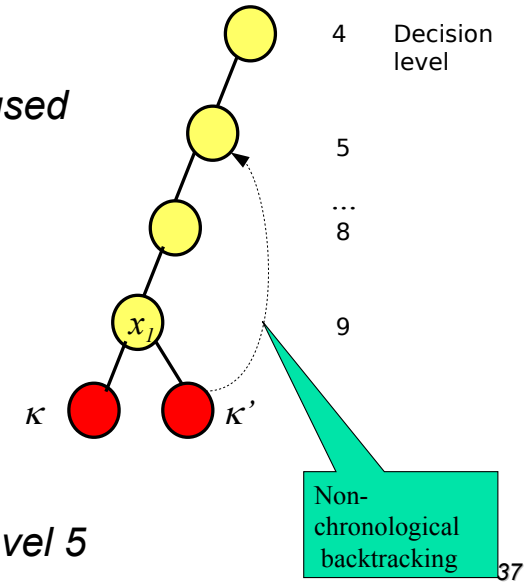
and backtrack to the highest (deepest) dec. level in this clause (5).

Non-chronological backtracking

Which assignments caused the conflicts?

x_9= 0@1
x_{10}= 0@3
x_{11}= 0@4
x_{12}= 1@2
x_{13}= 1@5

These assignments Are sufficient for Causing a conflict.



Backtrack to decision level 5

Conflict graph

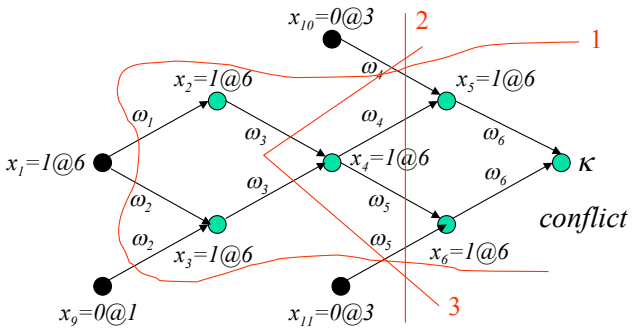
- Nel conflict graph si distinguono i tipi di vertici:
  - Branching vertices: sono quelli stabiliti dalla search
  - Deduced vertices: sono variabili fissate dalla unit propagation
    - Fra i deduced vertices ce n'è uno speciale *k* che rappresenta il conflitto
- Qualunque taglio che separa i branching vertices dal conflitto rappresenta una clausola di conflitto valida
  - Il lato dei branching vertices si chiama *reason side*, quello del conflitto si chiama *conflict side*
  - I vertici di confine dal reason side si chiamano *conflict set*
  - La clausola ottenuta negando i letterali del conflict set è la clausola appresa

Non-chronological backtracking (option #1)

- *So the rule is: backtrack to the largest decision level in the conflict clause.*
- *Q: What if the flipped assignment works ?*
  A: continue to the next decision level, leaving the current one without a decision variable.
  - *Backtracking back to this level will lead to another conflict and further backtracking.*

More Conflict Clauses

- *Def: A Conflict Clause is any clause implied by the formula*
- *Let L be a set of literals labeling nodes that form a cut in the implication graph, separating the conflict node from the roots.*
- *Claim: $\bigvee_{l \in L} \neg l$ is a Conflict Clause.*



1. $(x_{10} \vee \text{-}x_1 \vee x_9 \vee x_{11})$
2. $(x_{10} \vee \text{-}x_4 \vee x_{11})$
3. $(x_{10} \vee \text{-}x_2 \vee \text{-}x_3 \vee x_{11})$

Conflict clauses

- *How many clauses should we add ?*
- *If not all, then which ones ?*
  - *Shorter ones ?*
  - *Check their influence on the backtracking level?*
  - *The most "influential"?*
- *The answer requires two definitions:*
  - *Asserting clauses*
  - *Unique Implication points (UIP's)*

Asserting clauses

- *Def: An Asserting Clause is a Conflict Clause with a single literal from the current decision level. Backtracking (to the right level) makes it a Unit clause.*
- *Modern solvers only consider Asserting Clauses.*

Unique Implication Points (UIP's)

- *Def: A node N* in the implication graph is a *Unique Implication Point (UIP) if all paths from current decision node to the conflict node must go through N.*
- *The First-UIP is the closest UIP to the conflict.*
- *The method of choice: an asserting clause that includes the first UIP. In this case* $(x_{10} \vee \text{-}x_4 \vee x_{11})$.