universität innsbruck

SAT and SMT Solving

Sarah Winkler

KRDB
Department of Computer Science
Free University of Bozen-Bolzano

lecture 3
WS 2022

Outline

- Summary of Last Week
- Maximum Satisfiability
- Algorithms for Minimum Unsatisfiability
- Application: Automotive Configuration
- NP-Completeness

Definition (Implication Graph)

for derivation $\| F' \implies^*_{\mathcal{B}} M \| F$ implication graph is constructed as follows:

- add node labelled l for every decision literal l in M
- repeat until there is no change:
  if $\exists$ clause $l_1 \vee \dots l_m \vee l'$ in F such that there are already nodes $l_1^c, \dots, l_m^c$
  - add node l' if not yet present
  - add edges $l_i^c \to l'$ for all $1 \leqslant i \leqslant m$ if not yet present
- if $\exists$ clause $l'_1 \vee \dots \vee l'_k$ in F such that there are nodes $l_1'^c, \dots, l_k'^c$
  - add conflict node labeled C
  - add edges $l_i'^c \to C$

Definitions

- cut separates decision literals from conflict node
- literal l in implication graph is unique implication point (UIP) if all paths from last decision literal to conflict node go through l

Lemma

- *if edges intersected by cut are $l_1 \to l'_1, \dots, l_k \to l'_k$ then $F' \vDash l_1^c \vee \dots \vee l_k^c$*
- *this clause is backjump clause if some l_i is UIP*

Backjump clauses by resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Lemma

every clause C_i corresponds to cut in implication graph:
there is cut intersecting edges $l_{i1} \to l'_{i1}, \dots, l_{ik} \to l'_{ik}$ such that $C_i = l_{i1}^c \vee \dots \vee l_{ik}^c$

Definition (DPLL with Learning and Restarts)

DPLL with learning and restarts $\mathcal{R}$ extends system $\mathcal{B}$ by following three rules:

- learn $\qquad M \parallel F \implies M \parallel F, C$
  if $F \vDash C$ and all atoms of C occur in M or F
- forget $\qquad M \parallel F, C \implies M \parallel F$
  if $F \vDash C$
- restart $\qquad M \parallel F \implies \parallel F$

Theorem (Termination)

any derivation $\parallel F \implies_{\mathcal{R}} S_1 \implies_{\mathcal{R}} S_2 \implies_{\mathcal{R}} \dots$ *is finite if*

- *it contains no infinite subderivation of learn and forget steps, and*
- *restart is applied with increasing periodicity*

Theorem (Correctness)

for $\parallel F \implies_{\mathcal{R}} S_1 \implies_{\mathcal{R}} S_2 \implies_{\mathcal{R}} \dots \implies_{\mathcal{R}} S_n$ *with final state* S_n:

- *if* $S_n = \text{FailState}$ *then* F *is unsatisfiable*
- *if* $S_n = M \parallel F'$ *then* F *is satisfiable and* $M \vDash F$

Two-Watched Literal Scheme

Idea

- maintain two pointers p_1 and p_2 for each clause C
- each pointer points to a literal in the clause that is: unassigned or true if possible, otherwise false
- ensure invariant that $p_1(C) \neq p_2(C)$

Key properties

- clause C enables unit propagation if $p_1(C)$ is false and $p_2(C)$ is unassigned or vice versa $\qquad \mathcal{O}(n)$
- clause C is conflict clause if $p_1(C)$ and $p_2(C)$ are false literals

Setting pointers

- initialization: set p_1 and p_2 to different (unassigned) literals in clause
- decide or unit propagate:
  when assigning literal l true, redirect all pointers to l^c to other literal in their clause if possible
- backjump: no need to change pointers!

Outline

- Summary of Last Week
- Maximum Satisfiability
- Algorithms for Minimum Unsatisfiability
- Application: Automotive Configuration
- NP-Completeness

maxSAT

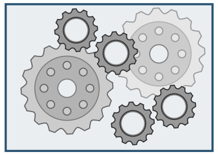
maxSAT Problem

input: propositional formula φ in CNF

output: valuation α such that α satisfies maximal number of clauses in φ

$\varphi \longrightarrow$

$(q \vee \neg r) \wedge (\neg q \vee r) \wedge p \wedge (\neg p \vee \neg q) \wedge (\neg p \vee r) \wedge (\neg p \vee \neg r \vee q)$



maxSAT solver

$\longrightarrow$ unsat

$\alpha(p) = \alpha(q) = \alpha(r) = \mathsf{T}$

Terminology

- optimization problem P asks to find "best" solution among all solutions
- maxSAT encoding transforms optimization problem P into formula φ such that optimal solution to P corresponds to maxSAT solution to φ

Remark
many real world are have optimization problems

Examples

- find shortest path to goal state
  - planning
  - model checking
- find smallest explanation
  - debugging
  - configuration
- find least resource-consuming schedule
  - scheduling
  - logistics
- find most probable explanation
  - probabilistic inference
- ...

Notation

for valuation v let $\overline{v}(\varphi) = \begin{cases} 1 & \text{if } v(\varphi) = \mathsf{T} \\ 0 & \text{if } v(\varphi) = \mathsf{F} \end{cases}$

Maximal Satisfiability

Consider CNF formula φ as set of clauses $C \in \varphi$

Maximal Satisfiability (maxSAT)
instance: CNF formula φ
question: what is maximal $\sum_{C \in \varphi} \overline{v}(C)$ for valuation v?

Partial Maximal Satisfiability (pmaxSAT)
instance: CNF formulas χ and φ
question: what is maximal $\sum_{C \in \varphi} \overline{v}(C)$ for valuation v with $v(\chi) = \mathsf{T}$?

Example

$$\begin{aligned} \varphi = \{\ & 6 \vee 2, && \overline{6} \vee 2, && \overline{2} \vee 1, && \overline{1}, && \overline{6} \vee 8, && 6 \vee \overline{8}, \\ & 2 \vee 4, && \overline{4} \vee 5, && 7 \vee 5, && \overline{7} \vee 5, && \overline{3}, && \overline{5} \vee 3\ \} \\ \chi = \{\ & \overline{1} \vee 2, && \overline{2} \vee \overline{3}, && \overline{5} \vee 1, && 3\ \} \end{aligned}$$

- $\mathsf{maxSAT}(\varphi) = 10$, e.g. for valuation $\overline{1}\ 2\ \overline{3}\ 4\ 5\ 6\ \overline{7}\ 8$
- $\mathsf{pmaxSAT}(\chi, \varphi) = 8$, e.g. for valuation $\overline{1}\ \overline{2}\ 3\ 4\ \overline{5}\ 6\ 7\ 8$

Terminology

- φ are soft constraints

Weighted Maximal Satisfiability (maxSAT_w)
instance: CNF formula φ with weight $w_C \in \mathbb{Z}$ for all $C \in \varphi$
question: what is maximal $\sum_{C \in \varphi} w_C \cdot \overline{v}(C)$ for valuation v?

Weighted Partial Maximal Satisfiability ($\mathsf{pmaxSAT}_w$)
instance: CNF formulas φ and χ, with weight $w_C \in \mathbb{Z}$ for all $C \in \varphi$
question: what is maximal $\sum_{C \in \varphi} w_C \cdot \overline{v}(C)$ for valuation v with $v(\chi) = \mathsf{T}$?

Notation

write $\mathsf{maxSAT}_w(\varphi)$ and $\mathsf{pmaxSAT}_w(\chi, \varphi)$ for solutions to these problems

Example

$$\varphi = \{(\neg x, 2),\quad (y, 4),\quad (\neg x \vee \neg y, 5),\quad (x \vee \neg y, 1)\}$$
$$\chi = \{x\}$$

- $\mathsf{maxSAT}_w(\varphi) = 11$ e.g. for valuation $v(x) = \mathsf{F}$ and $v(y) = \mathsf{T}$
- $\mathsf{pmaxSAT}_w(\chi, \varphi) = 6$, e.g. for valuation $v(x) = \mathsf{T}$ and $v(y) = \mathsf{F}$

Minimum Unsatisfiability (minUNSAT)
instance: CNF formula φ
question: what is minimal $\sum_{C \in \varphi} \overline{v}(\neg C)$ for valuation v?

Notation

write $\mathsf{minUNSAT}(\varphi)$ for solution to minimal unsatisfiability problem for φ

Lemma

$$|\varphi| = \mathsf{minUNSAT}(\varphi) + \mathsf{maxSAT}(\varphi)$$

Example

$$\varphi = \{\neg x,\quad x \vee y,\quad \neg y \vee \neg z,\quad x,\quad y \vee \neg z\}$$

using $v(x) = v(y) = \mathsf{T}$ and $v(z) = \mathsf{F}$ have

- $\mathsf{maxSAT}(\varphi) = 4$
- $\mathsf{minUNSAT}(\varphi) = 1$

Remark
maxSAT and minUNSAT are dual notions

Outline

- Summary of Last Week
- Maximum Satisfiability
- Algorithms for Minimum Unsatisfiability
  - Branch and Bound
  - Binary Search
- Application: Automotive Configuration
- NP-Completeness

Outline

- Summary of Last Week
- Maximum Satisfiability
- Algorithms for Minimum Unsatisfiability
  - Branch and Bound
  - Binary Search
- Application: Automotive Configuration
- NP-Completeness

Branch & Bound

Idea

- gets list of clauses φ as input and returns minUNSAT(φ)
- explores assignments in depth-first search

Ingredients

- UB is minimal number of unsatisfied clauses found so far (upper bound)
- φ_x is formula φ with all occurrences of x replaced by T
- $\varphi_{\overline{x}}$ is formula φ with all occurrences of x replaced by F
- for list of clauses φ, function `simp(`φ`)`
  - replaces ¬T by F and ¬F by T
  - drops all clauses which contain T
  - removes F from all remaining clauses
- $\square$ denotes empty clause and `#empty(`φ`)` number of empty clauses in φ

Example

$$\varphi = y \vee \neg F, \quad x \vee y \vee F, \quad F, \quad x \vee \neg y \vee T, \quad x \vee \neg z$$

$$\texttt{simp}(\varphi) = \qquad x \vee y, \quad \square, \qquad x \vee \neg z$$

Algorithm (Branch & Bound)

```
function BnB(φ, UB)
  φ = simp(φ)
  if φ contains only empty clauses then
      return #empty(φ)
  if #empty(φ) ⩾ UB then
      return UB
  x = selectVariable(φ)
  UB′ = min(UB, BnB(φ_x, UB))
  return min(UB′, BnB(φ_x̄, UB′))
```

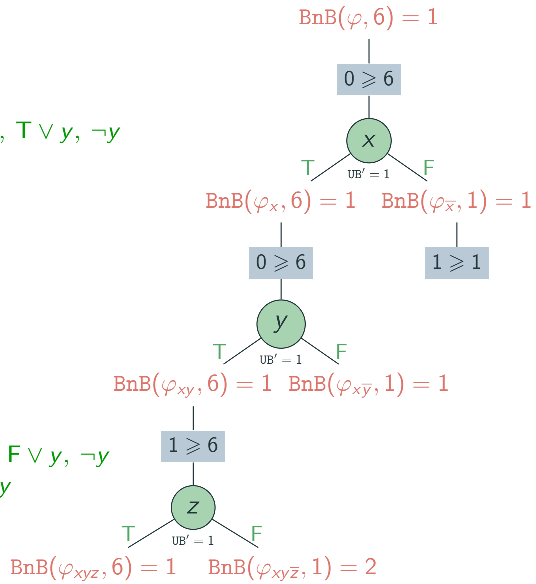
- note that number of clauses falsified by any valuation is $\leqslant |\varphi|$
- start by calling `BnB(`φ, $|\varphi|$`)`
- idea: `#empty(`φ`)` is number of clauses falsified by current valuation

Example

- $\varphi = x,\ \neg x \vee y,\ z \vee \neg y,\ x \vee z,\ x \vee y,\ \neg y$
- call $\texttt{BnB}(\varphi,\ 6)$
- $\texttt{simp}(\varphi) = \varphi$

- $\varphi_x = \mathsf{T},\ \neg\mathsf{T} \vee y,\ z \vee \neg y,\ \mathsf{T} \vee z,\ \mathsf{T} \vee y,\ \neg y$
  $\texttt{simp}(\varphi_x) = y,\ z \vee \neg y,\ \neg y$
  - $\varphi_{xy} = \mathsf{T},\ z \vee \neg\mathsf{T},\ \neg T$
    $\texttt{simp}(\varphi_{xy}) = z, \square$
    - $\varphi_{xyz} = \mathsf{T},\ \square$
      $\texttt{simp}(\varphi_{xyz}) = \square$
    - $\varphi_{xy\overline{z}} = \mathsf{F},\ \square$
      $\texttt{simp}(\varphi_{xy\overline{z}}) = \square,\ \square$
  - $\varphi_{x\overline{y}} = \mathsf{F},\ z \vee \neg\mathsf{F},\ \neg\mathsf{F}$
    $\texttt{simp}(\varphi_{x\overline{y}}) = \square$
- $\varphi_{\overline{x}} = \mathsf{F},\ \neg\mathsf{F} \vee y,\ z \vee \neg y,\ \mathsf{F} \vee z,\ \mathsf{F} \vee y,\ \neg y$
  $\texttt{simp}(\varphi_{\overline{x}}) = \square,\ z \vee \neg y,\ z,\ y,\ \neg y$

- $\mathsf{minUNSAT}(\varphi) = 1$
- e.g. $v(x) = v(y) = v(z) = \mathsf{T}$



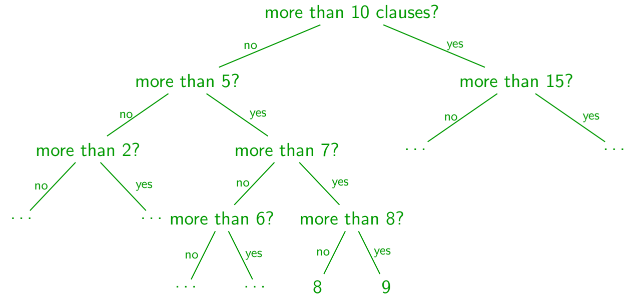
Binary Search

Idea

- gets list of clauses φ as input and returns $\mathsf{minUNSAT}(\varphi)$
- repeatedly call SAT solver in binary search fashion

Example

Suppose given formula with 20 clauses. Can we satisfy ...



Cardinality Constraints

Definitions

- cardinality constraint has form $(\sum_{x \in X} x) \bowtie N$ where $\bowtie$ is $=, <, >, \leqslant$, or $\geqslant$, X is set of propositional variables and $N \in \mathbb{N}$
- valuation v satisfies $(\sum_{x \in X} x) \bowtie N$ iff $k \bowtie N$ where k is number of variables $x \in X$ such that $v(x) = \mathsf{T}$

Remarks

- cardinality constraints are expressible in CNF
  - enumerate all possible subsets $\mathcal{O}(2^{|X|})$
  - BDDs $\mathcal{O}(N \cdot |X|)$
  - sorting networks $\mathcal{O}(|X| \cdot \log^2(|X|))$
- write $\mathtt{CNF}(\sum_{x \in X} x \bowtie N)$ for CNF encoding
- cardinality constraints occur very frequently! (n-queens, Minesweeper, ...)

Example

- $x + y + z = 1$ satisfied by $v(x) = v(y) = \mathsf{F},\ v(z) = \mathsf{T}$
- $x_1 + x_2 + \cdots + x_8 \leqslant 3$ satisfied by $v(x_1) = \cdots = v(x_8) = \mathsf{F}$

Algorithm (Binary Search)

```
function BinarySearch({C_1, ..., C_m})
  φ := {C_1 ∨ b_1, ..., C_m ∨ b_m}
  return search(φ, 0, m)
```

$b_1, \ldots, b_m$ are fresh variables

```
function search(φ, L, U)
  if L ⩾ U then
    return U
  mid := ⌊(U+L)/2⌋
  if SAT(φ ∧ CNF(∑_{i=1}^m b_i ⩽ mid)) then
    return search(φ, L, mid)
  else
    return search(φ, mid + 1, U)
```

Theorem

$\mathtt{BinarySearch}(\psi) = \mathsf{minUNSAT}(\psi)$

Example

$$\varphi = \{\ 6 \vee 2 \vee b_1,\quad \overline{6} \vee 2 \vee b_2,\quad \overline{2} \vee 1 \vee b_3,\quad \overline{1} \vee b_4,\quad \overline{6} \vee 8 \vee b_5,\quad 6 \vee \overline{8} \vee b_6,\quad 2 \vee 4 \vee b_7,\quad \overline{4} \vee 5 \vee b_8,\quad 7 \vee 5 \vee b_9,\quad \overline{7} \vee 5 \vee b_{10},\quad \overline{3} \vee b_{11},\quad \overline{5} \vee 3 \vee b_{12}\ \}$$

- $\mathtt{L} = 0$, $\mathtt{U} = 12$, $\mathtt{mid} = 6$ $\mathtt{SAT}(\varphi \wedge \mathtt{CNF}(\sum_{i=1}^m b_i \leqslant 6))$? ✓
- $\mathtt{L} = 0$, $\mathtt{U} = 6$, $\mathtt{mid} = 3$ $\mathtt{SAT}(\varphi \wedge \mathtt{CNF}(\sum_{i=1}^m b_i \leqslant 3))$? ✓
- $\mathtt{L} = 0$, $\mathtt{U} = 3$, $\mathtt{mid} = 1$ $\mathtt{SAT}(\varphi \wedge \mathtt{CNF}(\sum_{i=1}^m b_i \leqslant 1))$? ✗
- $\mathtt{L} = 2$, $\mathtt{U} = 3$, $\mathtt{mid} = 2$ $\mathtt{SAT}(\varphi \wedge \mathtt{CNF}(\sum_{i=1}^m b_i \leqslant 2))$? ✓
- $\mathtt{L} = 2$, $\mathtt{U} = 2$ `return 2`

Cardinality Constraints in Z3

```python
from z3 import *

xs = [ Bool("x"+str(i)) for i in range (0,10)]
ys = [ Bool("y"+str(i)) for i in range (0,10)]

def card(ps):
  return sum([If(x, 1, 0) for x in ps])

solver = Solver()
solver.add(card(xs) == 5, card(ys) > 2, card(ys) <= 4)

if solver.check() == sat:
  model = solver.model()
  for i in range(0,10):
    print(xs[i], "=", model[xs[i]], ys[i], "=", model[ys[i]])
```


MaxSAT in Z3

```python
from z3 import *

vs = [Bool("v" + str(i)) for i in range(0,5)]
opt = Optimize() # like solver, but can maximize
# add hard constraints directly
opt.add(Or(Not(vs[2]), vs[3], vs[4]))
opt.add(Or(Not(vs[3]), vs[0]))
# now the soft constraints
c0 = Or(vs[2], vs[1])
c1 = Or(Not(vs[2]), vs[1])
c2 = Or(Not(vs[1]), vs[0])
c3 = Not(vs[0])
c4 = Or(Not(vs[3]), vs[1])
# build cost: If(c0,1,0) + If(c1, 1, 0) + If(c2, 1, 0) + ...
cost = sum([ If(c, 1, 0) for c in [c0, c1, c2, c3, c4] ])
opt.maximize(cost)
res = opt.check()
if res == z3.sat:
  model = opt.model() # get valuation
  print(model.eval(cost)) # number of satisfied clauses
  print(model) # assignment
```


Application: Automotive Configuration (1)

Manufacturer constraints on components

component family	components	limit
engine	E_1, E_2, E_3	$= 1$
gearbox	G_1, G_2, G_3	$= 1$
control unit	$C_1, \dots, C_5$	$= 1$
dashboard	$D_1, \dots, D_4$	$= 1$
navigation system	N_1, N_2, N_3	$\leqslant 1$
air conditioner	AC_1, AC_2, AC_3	$\leqslant 1$
alarm system	AS_1, AS_2	$\leqslant 1$
radio	$R_1, \dots, R_5$	$\leqslant 1$

Component families with limitations

G_1	$\rightarrow$	$E_1 \vee E_2$
$N_1 \vee N_2$	$\rightarrow$	D_1
N_3	$\rightarrow$	$D_2 \vee D_3$
$AC_1 \vee AC_3$	$\rightarrow$	$D_1 \vee D_2$
AS_1	$\rightarrow$	$D_2 \vee D_3$
$R_1 \vee R_2 \vee R_5$	$\rightarrow$	$D_1 \vee D_4$

Component dependencies

Encoding

- for every component c use variable x_c which is assigned T iff c is used
- require limitations and dependencies φ_{car} by adding respective clauses

Problem 1: Validity of configuration

- is desired configuration valid?   SAT encoding

  e.g. $E_1 \wedge G_1 \wedge C_5 \wedge (D_2 \vee D_3)$ ✓ $E_3 \wedge G_1 \wedge C_5 \wedge D_2 \vee AC_1$ ✗

Application: Automotive Configuration (2)

Problem 2: Maximize number of desired components

- find maximal valid subset of configuration $c_1, \ldots, c_n$ — partial maxSAT
- possibly with priorities p_i for component c_i — weighted partial maxSAT

$$\underbrace{\varphi_{\text{car}}}_{\text{hard clauses}} \wedge \underbrace{x_{c_1} \wedge \cdots \wedge x_{c_n}}_{\text{soft clauses}}$$

Problem 3: Minimization of cost

- given cost q_i for each component c_i, find cheapest valid configuration
  weighted partial maxSAT

$$\underbrace{\varphi_{\text{car}}}_{\text{hard clauses}} \wedge \underbrace{(c_1, -q_1) \wedge \cdots \wedge (c_n, -q_n)}_{\text{soft clauses}}$$

Result

collaboration with BMW: evaluated on configuration formulas of 2013 product line

Complexity

Remark

maxSAT is not a decision problem

Definition

FP^{NP} is class of functions computable in polynomial time with access to NP oracle

Theorem

maxSAT *is* FP^{NP}*-complete*

Remarks

- FP^{NP} allows polynomial number of oracle calls (which is e.g. SAT solver)
- other members of FP^{NP}:
  optimization versions of travelling salesperson and Knapsack

Outline

- Summary of Last Week
- Maximum Satisfiability
- Algorithms for Minimum Unsatisfiability
- Application: Automotive Configuration
- NP-Completeness

NP-Completeness

Theorem (Cook 1971, Levin 1973)

SAT is NP-complete.

Proof.

- SAT is in NP — easy
  - given φ, guess nondeterministically an assignment v
  - can check whether v satisfies φ (in time linear in size of φ)
- SAT is NP-hard — hard
  - show that any problem in NP can be reduced to a SAT problem
  - more precisely:
    - given nondeterministic Turing machine $\mathcal{N}$ and input w such that $\mathcal{N}$ runs in polynomial time
    - construct formula φ such that

$$\mathcal{N} \text{ accepts } w \iff \varphi \text{ is satisfiable}$$

∎

Reminder: Turing Machines

Definition

Turing machine (TM) is 8-tuple $\mathcal{N} = (Q, \Sigma, \Gamma, \vdash, \textvisiblespace, \delta, s, t)$ with

- Q: finite set of states
- Σ: input alphabet
- $\Gamma \supseteq \Sigma$: tape alphabet
- $\vdash \in \Gamma - \Sigma$: left endmarker
- $\textvisiblespace \in \Gamma - \Sigma$: blank symbol
- $\delta \colon Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$: transition function
- $s \in Q$: start state
- $t \in Q$: accept state

$\vdash$	a	b	c	a	$\textvisiblespace$	b	a	$\textvisiblespace$	$\textvisiblespace$
		$\uparrow$							
		q							

such that

$$\forall\, a \in \Gamma\ \exists\, b, b' \in \Gamma\ \exists\, d, d' \in \{L, R\}\colon\ \delta(t, a) = (t, b, d)$$

$$\forall\, p \in Q\ \exists\, q \in Q\colon\ \delta(p, \vdash) = (q, \vdash, R)$$

Definition

$\mathcal{N}$ accepts w if there is accepting run $(s, \vdash w, 0) \xrightarrow[\mathcal{N}]{*} (t, \dots)$

Example (Turing machine to recognize palindromes)

$\mathcal{N} = (\mathcal{Q}, \Sigma, \Gamma, \vdash, \textvisiblespace, \delta, q_{init}, q_{acc})$ with

- $\mathcal{Q} = \{q_{init}, q_{read0}, q_{read1}, q_{acc}, q_{search0}, q_{search1}, q_{back}\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, \vdash, \textvisiblespace\}$
- start state q_{init}, accept state q_{acc}
-

δ	$\vdash$	0	1	$\textvisiblespace$
q_{init}	$(q_{init}, \vdash, R)$	$(q_{read0}, \vdash, R)$	$(q_{read1}, \vdash, R)$	$(q_{acc}, \textvisiblespace, R)$
q_{read0}		$(q_{read0}, 0, R)$	$(q_{read0}, 1, R)$	$(q_{search0}, \textvisiblespace, L)$
q_{read1}		$(q_{read1}, 0, R)$	$(q_{read1}, 1, R)$	$(q_{search1}, \textvisiblespace, L)$
$q_{search0}$	$(q_{acc}, \vdash, R)$	$(q_{back}, \textvisiblespace, L)$		
$q_{search1}$	$(q_{acc}, \vdash, R)$		$(q_{back}, \textvisiblespace, L)$	
q_{back}	$(q_{init}, \vdash, R)$	$(q_{back}, 0, L)$	$(q_{back}, 1, L)$	

Proof: SAT is NP hard

- given nondeterministic Turing machine $\mathcal{N}$ running in polynomial time
- i.e. there is some polynomial $p(n)$ such that for any input w of size n, $\mathcal{N}$ needs at most $p(n)$ steps
- in $p(n)$ steps, $\mathcal{N}$ can write at most $p(n)$ tape cells
- represent run of $\mathcal{N}$ as computation table of size $(p(n)+1) \times (p(n)+1)$
  - every cell contains a symbol in Γ
  - the first row represents the initial configuration
  - all other rows are configuration that follows from the previous one
- encode in huge (but polynomial-size) formula that table models accepting run

Encoding: Variables

			how many?
$T_{i,j,s}$	$0 \leqslant i, j \leqslant p(n)$, $s \in \Gamma$	in ith configuration, jth symbol on tape is s	$\mathcal{O}(p(n)^2)$
$H_{i,j}$	$0 \leqslant i, j \leqslant p(n)$	in ith configuration, read head is at position j	$\mathcal{O}(p(n)^2)$
$Q_{i,q}$	$0 \leqslant i \leqslant p(n)$, $q \in \mathcal{Q}$	state is q in ith configuration	$\mathcal{O}(p(n))$

Example (TM $\mathcal{N}$ for palindromes)

- needs at most $p(n) = (n+1)(n+2)/2 + 1$ steps on input of length n
- for input 010, have computation table

q_{init}	$\vdash$	0	1	0	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$
q_{init}	$\vdash$	0	1	0	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$
q_{read0}	$\vdash$	$\vdash$	1	0	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$
q_{read0}	$\vdash$	$\vdash$	1	0	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$
q_{read0}	$\vdash$	$\vdash$	1	0	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$
$q_{search0}$	$\vdash$	$\vdash$	1	0	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$
q_{back}	$\vdash$	$\vdash$	1	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$
q_{back}	$\vdash$	$\vdash$	1	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$
q_{init}	$\vdash$	$\vdash$	1	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$
$q_{search1}$	$\vdash$	$\vdash$	$\vdash$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$
$q_{search1}$	$\vdash$	$\vdash$	$\vdash$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$
q_{acc}	$\vdash$	$\vdash$	$\vdash$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$	$\textvisiblespace$

Proof: SAT is NP hard

- given nondeterministic Turing machine $\mathcal{N}$ running in polynomial time
- i.e. there is some polynomial $p(n)$ such that for any input w of size n, $\mathcal{N}$ needs at most $p(n)$ steps
- in $p(n)$ steps, $\mathcal{N}$ can write at most $p(n)$ tape cells
- represent run of $\mathcal{N}$ as computation table of size $(p(n)+1) \times (p(n)+1)$
  - every cell contains a symbol in Γ
  - the first row represents the initial configuration
  - all other rows are configuration that follows from the previous one
- encode in huge (but polynomial-size) formula that table models accepting run

Encoding: Variables

			how many?
$T_{i,j,s}$	$0 \leqslant i,j \leqslant p(n)$, $s \in \Gamma$	in ith configuration, jth symbol on tape is s	$\mathcal{O}(p(n)^2)$
$H_{i,j}$	$0 \leqslant i,j \leqslant p(n)$	in ith configuration, read head is at position j	$\mathcal{O}(p(n)^2)$
$Q_{i,q}$	$0 \leqslant i \leqslant p(n)$, $q \in \mathcal{Q}$	state is q in ith configuration	$\mathcal{O}(p(n))$

Encoding: Constraints (1)

- initial state of TM is q_{init}, initial head position is 0 $\mathcal{O}(1)$
  $$Q_{0,q_{init}} \wedge H_{0,0}$$
- initial tape content is w $\mathcal{O}(p(n))$
  $$T_{0,0,\vdash} \wedge \bigwedge_{1\leqslant j\leqslant n} T_{0,j,w_j} \wedge \bigwedge_{n<j\leqslant p(n)} T_{0,j,\sqcup}$$
- at least one symbol in every tape cell in every configuration $\mathcal{O}(p(n)^2)$
  $$\bigwedge_{0\leqslant i,j\leqslant p(n)} \bigvee_{s\in\Gamma} T_{i,j,s}$$
- at most one symbol in every tape cell in every configuration $\mathcal{O}(p(n)^2)$
  $$\bigwedge_{0\leqslant i,j\leqslant p(n)} \bigwedge_{s\neq s'\in\Gamma} \neg T_{i,j,s} \vee \neg T_{i,j,s'}$$
- at most one state at a time $\mathcal{O}(p(n))$
  $$\bigwedge_{0\leqslant i,j\leqslant p(n)} \bigwedge_{q\neq q'\in\mathcal{Q}} \neg Q_{i,q} \vee \neg Q_{i,q'}$$
- read head is in at most one position at a time $\mathcal{O}(p(n)^3)$
  $$\bigwedge_{0\leqslant i\leqslant p(n)} \bigwedge_{\bigwedge_{0\leqslant j<j'\leqslant p(n)}} \neg H_{i,j} \vee \neg H_{i,j'}$$

Encoding: Constraints (2)

- possible transitions* $\mathcal{O}(p(n)^2)$
  $$\begin{aligned}\bigwedge_{0\leqslant i,j\leqslant p(n)} \bigwedge_{q\in\mathcal{Q}} \bigwedge_{s\in\Gamma} (H_{i,j} \wedge Q_{i,q} \wedge T_{i,j,s}) \rightarrow \\ \bigvee_{(q',s',L)\in\delta(q,s)} (H_{i+1,j-1} \wedge Q_{i+1,q'} \wedge T_{i+1,j,s'}) \vee \\ \bigvee_{(q',s',R)\in\delta(q,s)} (H_{i+1,j+1} \wedge Q_{i+1,q'} \wedge T_{i+1,j+1,s'})\end{aligned}$$

  * needs some adjustments for $j=0$ and $j=p(n)$
- at some point accepting state q_{acc} is reached $\mathcal{O}(p(n)^2)$
  $$\bigwedge_{0\leqslant i\leqslant p(n)} Q_{i,q_{acc}}$$

Conclusion

- conjunction of constraints φ is satisfiable iff $\mathcal{N}$ admits accepting run on w
- size of φ is polynomial in n
- so problem in NP reduced to SAT ■

Literature

Rouven Walter, Christoph Zengler and Wolfgang Küchlin.
Applications of MaxSAT in Automotive Configuration.
Proc. International Configuration Workshop 2013, pp. 21-28, 2013.

André Abramé and Djamal Habet.
ahmaxsat: Description and Evaluation of a Branch and Bound Max-SAT Solver.
Journal on Satisfiability, Boolean Modeling and Computation 9, pp. 89–128, 2015.

Chu-Min Li and Felip Manyà.
MaxSAT, hard and soft constraints.
In: Handbook of Satisfiability, IOS Press, pp. 613–631, 2009.

Zhaohui Fu and Sharad Malik.
On solving the partial MAX-SAT problem.
In Proc. Theory and Applications of Satisfiability Testing, pp. 252–265, 2006