universität
innsbruck

SAT and SMT Solving

Sarah Winkler

Computational Logic Group
Department of Computer Science
University of Innsbruck

lecture 3
SS 2019

Outline

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

Definition (Implication Graph)

For derivation $\| F' \Longrightarrow_{\mathcal{B}}^{*} M \| F$ 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 of implication graph has at least all decision literals on the left, and at least the conflict node on the right
- 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* $F' \vDash l_1^c \vee \dots \vee l_k^c$

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}

Observation

every clause C_i corresponds to cut in implication graph

Definition (DPLL with Learning and Restarts)

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

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

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* $M \vDash F$

Outline

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

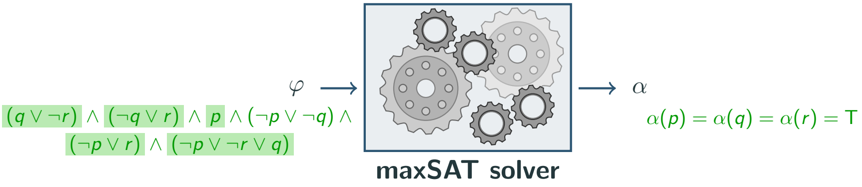
---

maxSAT

maxSAT Problem

input: propositional formula φ in CNF

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



$\varphi \longrightarrow$ **maxSAT solver** $\longrightarrow \alpha$

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

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

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 problems have optimization component

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

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

$$\varphi = \{\ 6 \vee 2,\quad \overline{6} \vee 2,\quad \overline{2} \vee 1,\quad \overline{1},\quad \overline{6} \vee 8,\quad 6 \vee \overline{8},\quad 2 \vee 4,\quad \overline{4} \vee 5,\quad 7 \vee 5,\quad \overline{7} \vee 5,\quad \overline{3},\quad \overline{5} \vee 3\ \}$$

$$\chi = \{\ \overline{1} \vee 2,\quad \overline{2} \vee \overline{3},\quad \overline{5} \vee 1,\quad 3\ \}$$

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

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 ($\mathrm{pmaxSAT}_w$)

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

Example

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

$$\chi = \{x\}$$

- $\mathrm{maxSAT}_w(\varphi) = 11$ e.g. for valuation $v(x) = \mathsf{F}$ and $v(y) = \mathsf{T}$
- $\mathrm{pmaxSAT}_w(\chi, \varphi) = 5$, e.g. for valuation $v(x) = \mathsf{T}$ and $v(y) = \mathsf{F}$

Terminology

Minimum Unsatisfiability (minUNSAT)

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

Notation

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

Lemma

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

Example

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

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

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

Remark

maxSAT and minUNSAT are dual notions

Application: Automotive Configuration (1)

Manufacturer's 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	$C_1, \ldots, C_5$	$= 1$
dashboard	$D_1, \ldots, D_4$	$= 1$
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	$R_1, \ldots, R_5$	$\leqslant 1$

Component families with limitations

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

Component dependencies

Encoding

- for every component c use variable x_c which is assigned T iff c is used
- require manufacturer's constraints φ_{car} by adding respective clauses

Problem 1: Validity of Configuration

- is desired configuration valid? SAT encoding

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

Application: Automotive Configuration (2)

Problem 2: Maximization of Chosen Components

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

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

Problem 3: Minimization of Costs

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

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

Result

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

---

Outline

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

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 $\neg$T by F and $\neg$F by T
  - drops all clauses which contain T
  - removes F from all remaining clauses
- $\Box$ denotes empty clause and #`empty`(φ) number of empty clauses in φ

Example

$$\begin{array}{rlllll} \varphi = & y \vee \neg F, & x \vee y \vee F, & F, & x \vee \neg y \vee T, & x \vee \neg z \\ \mathtt{simp}(\varphi) = & & x \vee y, & \Box, & & x \vee \neg z \end{array}$$

Algorithm (Branch & Bound)

```
function BnB($\varphi$, UB)
  $\varphi$ = simp($\varphi$)
  if $\varphi$ contains only empty clauses then
     return #empty($\varphi$)
  if #empty($\varphi$) $\geqslant$ UB then
     return UB
  x = selectVariable($\varphi$)
  UB$'$ = min(UB, BnB($\varphi_x$, UB))
  return min(UB$'$, BnB($\varphi_{\overline{x}}$, UB$'$))
```

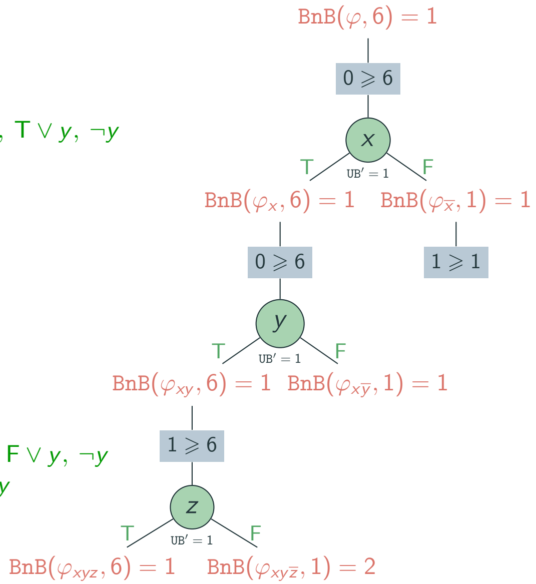
- note that number of clauses falsified by any valuation is $\leqslant |\varphi|$
- start by calling `BnB`(φ, $|\varphi|$)
- idea: #empty(φ) is number of clauses falsified by current valuation

Example

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

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

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



Algorithm (Branch & Bound, improved)

```
function BnB'($\varphi$, UB)
  $\varphi$ = simp($\varphi$)
  if $\varphi$ contains only empty clauses then
    return #empty($\varphi$)
  LB = #empty($\varphi$) + underapproximate($\varphi$)
  if LB $\geqslant$ UB then
    return UB
  x = selectVariable($\varphi$)
  UB$'$ = min(UB, BnB'($\varphi_x$, UB))
  return min(UB$'$, BnB'($\varphi_{\overline{x}}$, UB$'$))
```

Underapproximation (Wallace and Freuder)

- $\text{ic}(x)$ is number of unit clauses x in φ — inconsistency count
- $\texttt{underapproximate}(\varphi) = \sum_{x \text{ in } \varphi} \min(\text{ic}(x), \text{ic}(\neg x))$

Theorem

$\texttt{BnB}(\varphi, |\varphi|) = \texttt{BnB'}(\varphi, |\varphi|) = \text{minUNSAT}(\varphi)$

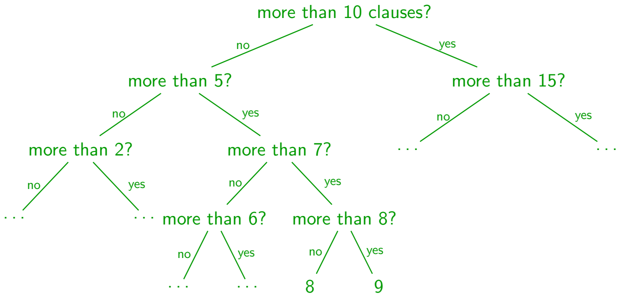
Binary Search

Idea

- gets list of clauses φ as input and returns $\text{minUNSAT}(\varphi)$
- 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 $v(x) = \mathrm{T}$

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 $\texttt{CNF}(\sum_{x \in X} x \bowtie N)$ for CNF encoding
- cardinality constraints occur very frequently! (n-queens, Minesweeper, ...)

Example

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

Algorithm (Binary Search)

```
function BinarySearch($\{C_1, \ldots, C_m\}$)
  $\varphi := \{C_1 \vee b_1, \ldots, C_m \vee b_m\}$
  return search($\varphi, 0, m$)
```

$b_1, \ldots, b_m$ are fresh variables

```
function search($\varphi$, L, U)
  if L $\geqslant$ U then
    return U
  mid := $\lfloor \frac{U+L}{2} \rfloor$
  if SAT($\varphi \wedge$ CNF($\sum_{i=1}^{m} b_i \leqslant$ mid)) then
    return search($\varphi$, L, mid)
  else
    return search($\varphi$, mid + 1, U)
```

Theorem

BinarySearch(ψ) = minUNSAT(ψ)

Example

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

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

Cardinality Constraints in Z3

```
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]]
```

Complexity

Remark

maxSAT is not a decision problem

Definition

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

Theorem

maxSAT *is* $\mathrm{FP}^{\mathrm{NP}}$*-complete*

Remarks

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

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