COL703: Logic for Computer Science (Aug-Nov 2022)

Lectures 11 & 12 (Horn-SAT, 2-SAT, DPLL)

Kumar Madhukar

madhukar@cse.iitd.ac.in

September 15th and 19th, 2022

Horn formulas

- a literal is a boolean variable or its negation
- for a variable x, we have a positive literal (x) and a negative literal ($\neg x$)
- a horn clause is a finite disjunction of literals with at most one positive literal
- a horn formula is a finite conjunction of horn clauses
- example
  $(x \vee \neg y \vee \neg z \vee \neg w) \wedge (\neg x \vee \neg y \vee \neg w) \wedge (\neg x \vee \neg z \vee w) \wedge (\neg x \vee y) \wedge (x) \wedge (\neg z) \wedge (\neg x \vee \neg y \vee w)$

Horn-SAT

- if the formula contains a unit clause, say (ℓ)
  - all clauses containing (ℓ) is removed
  - from all clauses containing $(\neg\ell)$ have $(\neg\ell)$ removed
- this may generate new unit clauses, which are propagated similarly
- if there are no unit clauses left, the formula can be satisfied by setting every remaining variable to false
- formula is unsat if propagation generates an empty clause

2-SAT

- given a 2-CNF formula, is it satisfiable or not
- every clause has 2 literals
- example
  $(\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$

2-SAT satisfiability check

- create a graph with $2n$ vertices (for a formula with n variables)
- corresponding to positive and negative literals for every variable
- for every clause $(a \vee b)$, create directed edges $\neg a \to b$ and $\neg b \to a$

2-SAT satisfiability check

- create a graph with $2n$ vertices (for a formula with n variables)
- corresponding to positive and negative literals for every variable
- for every clause $(a \vee b)$, create directed edges $\neg a \to b$ and $\neg b \to a$
- claim: if the graph contains a path from α to β, then it also contains a path from $\neg\beta$ to $\neg\alpha$

2-SAT satisfiability check

- create a graph with $2n$ vertices (for a formula with n variables)
- corresponding to positive and negative literals for every variable
- for every clause $(a \vee b)$, create directed edges $\neg a \rightarrow b$ and $\neg b \rightarrow a$
- claim: if the graph contains a path from α to β, then it also contains a path from $\neg\beta$ to $\neg\alpha$
- claim: a 2-CNF formula is unsat iff there exists a variable x such that:
  - there is a path from x to $\neg x$
  - there is a path from $\neg x$ to x

2-SAT satisfying assignment

- pick an unassigned literal ℓ, with no path from ℓ to $\neg\ell$
- assign `true` to ℓ and all vertices reachable from ℓ (and assign false to their negations)
- repeat until all vertices are assigned

Example

$$(\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$$

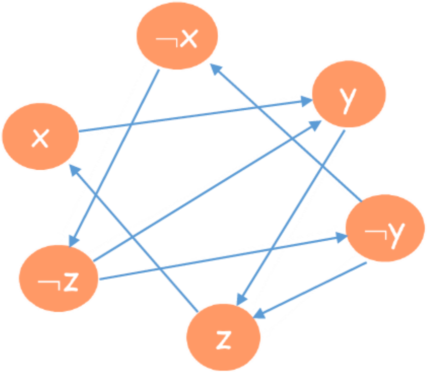

image source: https://www.iitg.ac.in/deepkesh/CS301/assignment-2/2sat.pdf

Another example

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \neg x_3) \wedge (x_4 \vee \neg x_1)$$

Tseitin transformation

- we know that an arbitrary boolean formula can be converted to CNF
- using De Morgan's law and distributivity property
- but this may result in an exponential explosion of the formula
- example: $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$
- Tseitin transformation is guaranteed to only linearly increase the size of the formula

Tseitin transformation: Example

(source: Wikipedia)

Consider the following formula ϕ .

$$\phi := ((p \lor q) \land r) \to (\neg s)$$

Consider all subformulas (excluding simple variables):

$$\begin{aligned}
&\neg s \\
&p \lor q \\
&(p \lor q) \land r \\
&((p \lor q) \land r) \to (\neg s)
\end{aligned}$$

Introduce a new variable for each subformula:

$$\begin{aligned}
&x_1 \leftrightarrow \neg s \\
&x_2 \leftrightarrow p \lor q \\
&x_3 \leftrightarrow x_2 \land r \\
&x_4 \leftrightarrow x_3 \to x_1
\end{aligned}$$

Conjunct all substitutions and the substitution for ϕ:

$$T(\phi) := x_4 \land (x_4 \leftrightarrow x_3 \to x_1) \land (x_3 \leftrightarrow x_2 \land r) \land (x_2 \leftrightarrow p \lor q) \land (x_1 \leftrightarrow \neg s)$$

All substitutions can be transformed into CNF, e.g.

$$\begin{aligned}
x_2 \leftrightarrow p \lor q &\equiv (x_2 \to (p \lor q)) \land ((p \lor q) \to x_2) \\
&\equiv (\neg x_2 \lor p \lor q) \land (\neg(p \lor q) \lor x_2) \\
&\equiv (\neg x_2 \lor p \lor q) \land ((\neg p \land \neg q) \lor x_2) \\
&\equiv (\neg x_2 \lor p \lor q) \land (\neg p \lor x_2) \land (\neg q \lor x_2)
\end{aligned}$$

1-SAT to 3-SAT

$c = (\ell)$

$c' = (\ell \vee u \vee v) \wedge (\ell \vee \neg u \vee v) \wedge (\ell \vee u \vee \neg v) \wedge (\ell \vee \neg u \vee \neg v)$

c' is satisfiable iff c is satisfiable.

2-SAT to 3-SAT

$c = (l_1 \vee l_2)$

$c' = (l_1 \vee l_2 \vee u) \wedge (l_1 \vee l_2 \vee \neg u)$

c' is satisfiable iff c is satisfiable.

k(>3)-SAT to (k-1)-SAT

$c = (l_1 \vee l_2 \vee \ldots \vee l_k)$

$c' = (l_1 \vee l_2 \vee \ldots l_{k-2} \vee u) \wedge (l_{k-1} \vee l_k \vee \neg u)$

c' is satisfiable iff c is satisfiable.

breaking 3SAT similarly to get 2SAT fails!

$c = (l_1 \lor l_2 \lor l_3)$

$c' = (l_1 \lor l_2 \lor u) \land (l_3 \lor \neg u)$ (still 3!)

$c' = (l_1 \lor u) \land (l_2 \lor l_3 \lor \neg u)$ (still 3!)

Davis-Putnam Algorithm

- unit-clause – if there is a unit clause ℓ, delete all clauses containing ℓ, and delete all occurrences of $\neg\ell$ from other clauses
- pure-literal – if there is a pure literal ℓ, delete all clauses containing ℓ
- eliminate a variable by resolution – choose an atom p and perform all possible resolutions on clauses that clash on p and $\neg p$. Add these resolvents to the set of clauses and then delete all the clauses containing p or $\neg p$.
- use these repeatedly; but use resolution only if the the first two rules do not apply

Davis-Putnam Algorithm

- if empty clause is produced, the formula is unsat
- if no more rules are applicable, report sat
- why does this terminate?
- why is this correct?
- example: $(p) \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg r \vee s \vee t)$

DPLL Algorithm

- creating all possible resolvents is very inefficient
- DPLL improves on the DP algorithm by replacing the variable elimination with a search for a model of the formula

DPLL

- Davis-Putnam-Logemann-Loveland algorithm (about 60 years old)
- combines search and deduction to decide satisfiability of CNF formulas
- based around backtrack search for a satisfying valuation

DPLL Algorithm

- the state of the algorithm is a pair $(\mathcal{F}, \mathcal{A})$
- a state is successful if $\mathcal{A}$ sets some literal in each clause of $\mathcal{F}$ to true
- a conflict state is one where $\mathcal{A}$ sets every literal in some clause to false
- let $\mathcal{F}|_{\mathcal{A}}$ denote the formula that we get after simplifying $\mathcal{F}$ using $\mathcal{A}$
- $(\mathcal{F}, \mathcal{A})$ is a conflict state if $\mathcal{F}|_{\mathcal{A}}$ contains the empty clause $\square$
- $(\mathcal{F}, \mathcal{A})$ is a successful state if $\mathcal{F}|_{\mathcal{A}}$ is the empty set of clause

DPLL Algorithm

1. initialize $\mathcal{A}$ to be an empty assignment
2. while there are unit clauses $\{\ell\}$, add $\ell \mapsto 1$ to $\mathcal{A}$
3. if $(\mathcal{F}, \mathcal{A})$ is a successful then stop and output $\mathcal{A}$
4. if $(\mathcal{F}, \mathcal{A})$ is a conflict state then apply clause learning to get a new clause $\mathcal{C}$
   - if $\mathcal{C}$ is $\square$ then stop and output *unsat*
   - add $\mathcal{C}$ to $\mathcal{F}$; backtrack to the highest level at which $\mathcal{C}$ is a unit clause; goto 2
5. add a new decision assignment $p_i \mapsto 1$ to $\mathcal{A}$; goto 2

Example[1]

C_1: $\{\neg p_1, \neg p_4, p_5\}$
C_2: $\{\neg p_1, p_6, \neg p_5\}$
C_3: $\{\neg p_1, \neg p_6, p_7\}$
C_4: $\{\neg p_1, \neg p_7, \neg p_5\}$
C_5: $\{p_1, p_4, p_6\}$

$\mathcal{A}$: $\langle p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1\rangle$

unit propagation generates a sequence of implied assignments: $\langle p_5 \xmapsto{C_1} 1, p_6 \xmapsto{C_2} 1, p_7 \xmapsto{C_3} 1\rangle$

conflict: C_4 becomes false!

[1] https://www.cs.ox.ac.uk/people/james.worrell/lec7-2015.pdf

Learned clause

if clause learning gives a clause $\mathcal{C}$, then we would want

- $\mathcal{F} \equiv \mathcal{F} \cup \mathcal{C}$
- $\mathcal{C}$ should be a conflict clause
- all variables in $\mathcal{C}$ should be decision variables (fixed using decision assignments)

Correctness

- termination – a sequence of decisions leading to a conflict cannot be repeated
- correctness – if empty clause is learned, then $\mathcal{F}$ is unsatisfiable (because $\mathcal{F} \equiv \mathcal{F} \cup \mathcal{C}$)

Clause learning

$\mathcal{A}$: $\langle p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1, p_5 \xmapsto{C_1} 1, p_6 \xmapsto{C_2} 1, p_7 \xmapsto{C_3} 1\rangle$

$$\begin{aligned}
A_8 &:= \{\neg p_1, \neg p_7, \neg p_5\} && \text{(clause } C_4\text{)} \\
A_7 &:= \{\neg p_1, \neg p_5, \neg p_6\} && \text{(resolve } A_8,\ C_3\text{)} \\
A_6 &:= \{\neg p_1, \neg p_5\} && \text{(resolve } A_7,\ C_2\text{)} \\
A_5 &:= \{\neg p_1, \neg p_4\} && \text{(resolve } A_6,\ C_1\text{)} \\
&\vdots \\
A_1 &:= \{\neg p_1, \neg p_4\}
\end{aligned}$$

Clause learning

$\mathcal{A}$: $\langle p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1, p_5 \xmapsto{C_1} 1, p_6 \xmapsto{C_2} 1, p_7 \xmapsto{C_3} 1\rangle$

$$
\begin{aligned}
A_8 &:= \{\neg p_1, \neg p_7, \neg p_5\} && \text{(clause } C_4\text{)}\\
A_7 &:= \{\neg p_1, \neg p_5, \neg p_6\} && \text{(resolve } A_8,\ C_3\text{)}\\
A_6 &:= \{\neg p_1, \neg p_5\} && \text{(resolve } A_7,\ C_2\text{)}\\
A_5 &:= \{\neg p_1, \neg p_4\} && \text{(resolve } A_6,\ C_1\text{)}\\
&\ \vdots\\
A_1 &:= \{\neg p_1, \neg p_4\}
\end{aligned}
$$

what about the things that were desirable from a learned clause?

Syllabus for Minor exam

Everything that has been taught till (including) today!

Next week

- Binary Decision Diagrams or First-Order Logic

Thank you!