Algorithms for Computational Logic

Summary

Contents

Chapter 1

SAT and Modeling with SAT

1.1 Cardinality Constraints

In order to handle cardinality constraints we have two options: encode the cardinality constraints to CNF and use a SAT solver, or use a pseudo boolean (PB) solver.

1.1.1 AtMost1

- $\sum_{j=1}^{n} x_j = 1$ can be encoded with $\left(\sum_{j=1}^{n} x_j \leq 1\right) \wedge \left(\sum_{j=1}^{n} x_j \geq 1\right)$
- $\sum_{j=1}^{n} x_j \geq 1$ can be encoded with $(x_1 \vee x_2 \vee ... \vee x_n)$
- $\sum_{j=1}^{n} x_j \leq 1$ can be encoded with:
  - Pairwise encoding
  - Sequential counter encoding
  - Bitwise encoding

Sequential Counter

In order to realize this encoding, we need to add new variables s_i for the fact "there is a 1 on some position $1..i$":

$$s_i \text{ is true if } \sum_{j=1}^{i} x_j \geq 1$$

Encoding $\sum_{j=1}^{n} x_j \leq 1$ with sequential counter:

$$\begin{aligned}
&(\neg x_1 \vee s_1) \wedge \\
&(\neg x_i \vee s_i), i \in 2..n-1 \wedge \\
&(\neg s_{i-1} \vee s_i), i \in 2..n-1 \wedge \\
&(\neg x_i \vee \neg s_{i-1}), i \in 2..n
\end{aligned}$$

If $x_j = 1$, then all s_i variables are assigned and all other x variables must take value 0. There are $\mathcal{O}(n)$ clauses and $\mathcal{O}(n)$ auxiliary variables.

Bitwise Encoding

In bitwise encoding, we represent the constraint $\sum_{j=1}^{n} x_j \leq 1$ by encoding the index of the potential true variable in binary. For this, we add new auxiliary variables:

$$v_0, ...v_r - 1;\ r = \lceil \log n \rceil (\text{with } n > 1)$$

Each variable x_j is assigned a unique binary number that represents its index. Then, for each variable x_j with binary index representation i, we create clauses that enforce the condition:

- If $x_j = 1$, assignment to v_i variables must encode $j - 1$, and all other x variables must take value 0
- If all $x_j = 0$, any assignment to v_i variables is consistent

For example, $x_1 + x_2 + x_3 \leq 1$:

	$j-1$	$v_1 v_0$
x_1	0	00
x_2	1	01
x_3	2	10

$$(\neg x_1 \vee \neg v_1) \wedge (\neg x_1 \vee \neg v_0)$$
$$(\neg x_2 \vee \neg v_1) \wedge (\neg x_2 \vee v_0)$$
$$(\neg x_3 \vee v_1) \wedge (\neg x_3 \vee \neg v_0)$$

There are $\mathcal{O}(n \log n)$ clauses and $\mathcal{O}(\log n)$ auxiliary variables

1.1.2 General Cardinality Constraints

Constraints of the form $\sum_{j=1}^{n} x_j \leq k$ or $\sum_{j=1}^{n} x_j \geq k$ can be added with:

- Sequential Counters
- BDDs
- Sorting Networks
- Cardinality Networks
- Totalizer

Sequential Counter Encoding

For each variable x_i, create k additional variables $s_{i,j}$ that are used as counters:

- $s_{i,j} = 1$ if at least j variables $\{x_1...x_i\}$ are assigned value 1
- $s_{i,j} = 0$ if at most $j - 1$ variables $\{x_1...x_i\}$ are assigned value 1

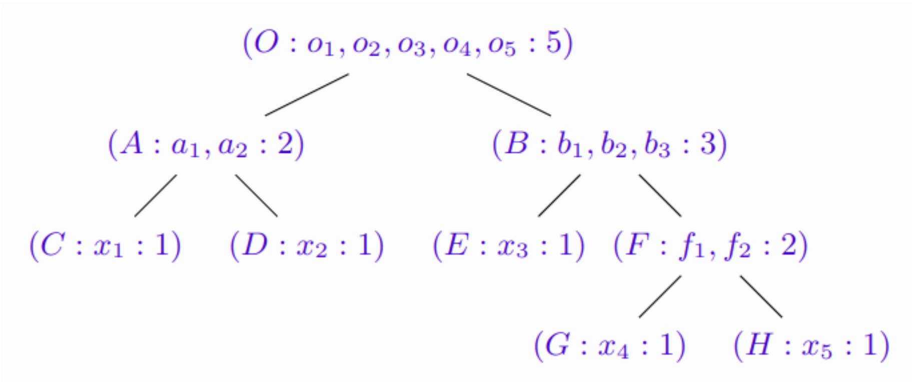
Encoding:

$$
\begin{array}{lr}
(\neg x_1 \vee s_{1,1}) & \\
(\neg s_{1,j}), & \forall j : 1 < j \leq k \\
 & \\
(\neg x_i \vee s_{i,1}), & \forall i : 1 < i < n \\
(\neg s_{i-1,1} \vee s_{i,1}), & \forall i : 1 < i < n \\
 & \\
(\neg s_{i-1,j} \vee s_{i,j}) & \forall i,j : 1 < i < n, 1 < j \leq k \\
(\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}) & \forall i,j : 1 < i < n, 1 < j \leq k \\
 & \\
(\neg x_i \vee \neg s_{i-1,k}) & \forall i : 1 < i \leq n
\end{array}
$$

Totalizer Encoding

In this encoding we count in unary how many of the n variables $(x_1 ... x_n)$ are assigned to 1. It can be visualized as a tree:

- Each node is $(name : variable : sum)$
- Root node has the output variables $(o_1 ... o_n)$ that count how many variables are assigned to 1
- Literals are at the leaves
- Each node counts in unary how many leaves are assigned to 1 in its subtree
- Example: if $b_2 = 1$, then at least 2 of the leaves (x_3, x_4, x_5) are assigned to 1



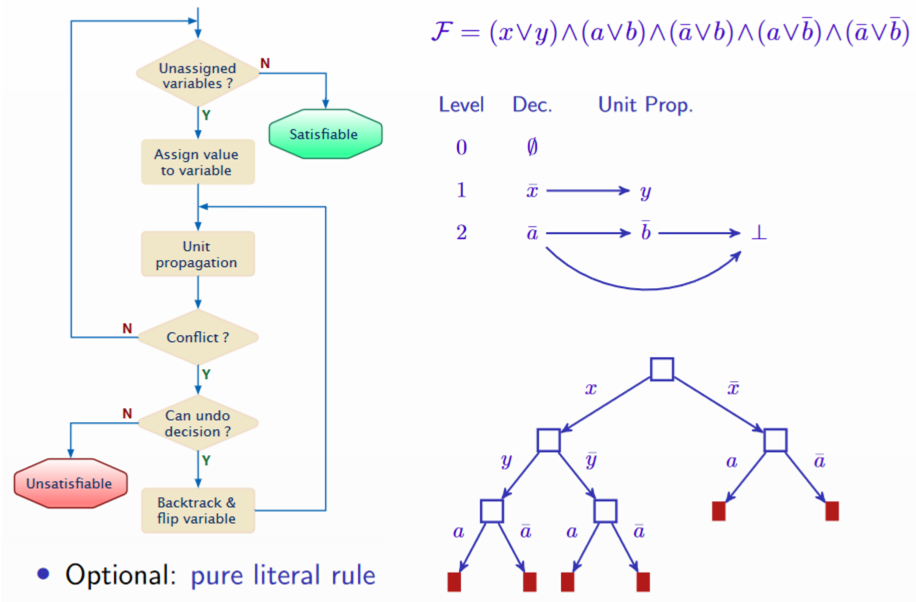
To encode $x_1 + x_2 + x_3 + x_4 + x_5 \leq 3$ just set $o_4 = 0$ and $o_5 = 0$.
Encoding:

$$
\bigwedge_{\substack{0 \leq \alpha \leq n_2 \\ 0 \leq \beta \leq n_3 \\ 0 \leq \sigma \leq n_1 \\ \alpha + \beta = \sigma}} \neg q_\alpha \vee \neg r_\beta \vee p_\sigma \quad \text{where, } p_0 = q_0 = r_0 = 1
$$

There are $\mathcal{O}(n \log n)$ new variables and $\mathcal{O}(n^2)$ new clauses

1.2 SAT Algorithms

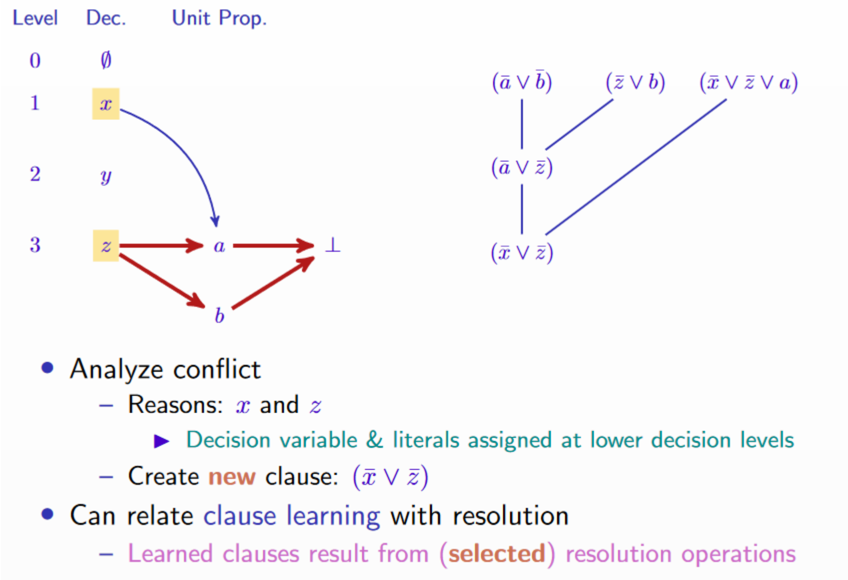
1.2.1 DPLL Solvers



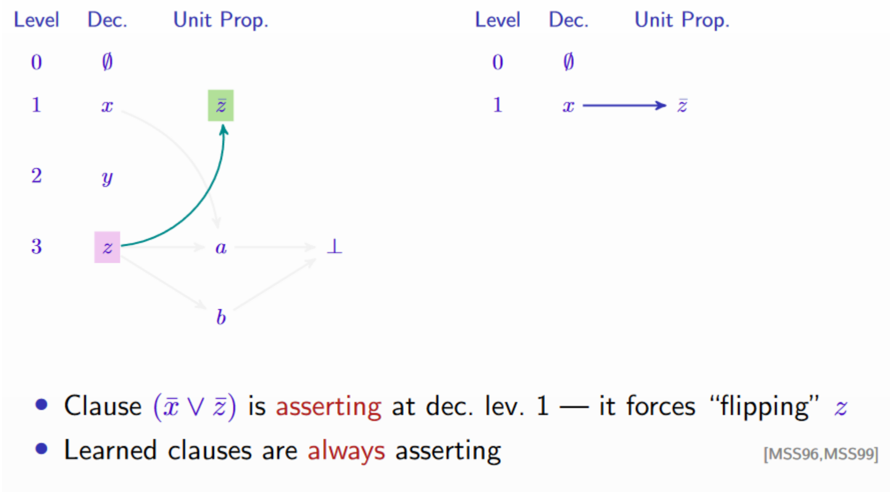
1.2.2 CDCL Solvers

CDCL solvers extend DPLL solvers with clause learning and non-chronological backtracking, search restarts, lazy data structures, conflict-guided branching, etc.

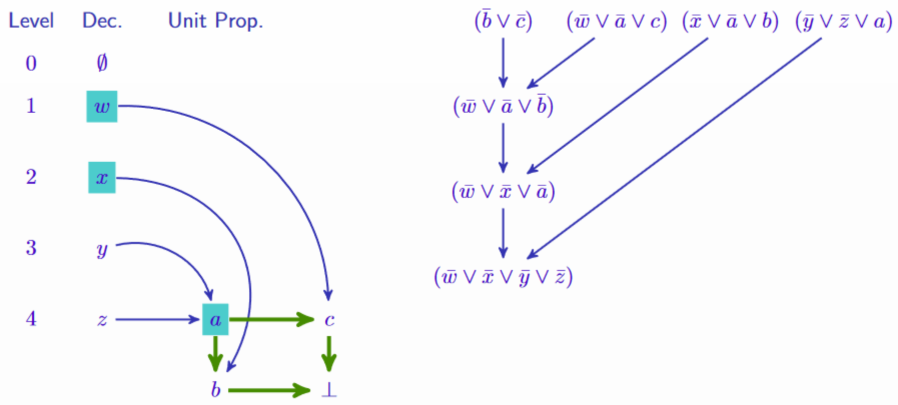
Clause Learning



And after backtracking:

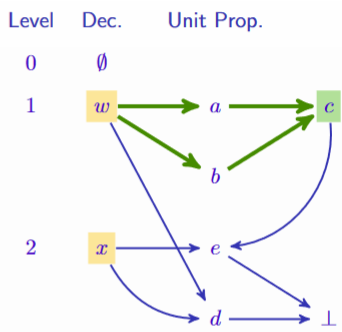


Unique Implication Points



- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$~~
- But a is an UIP — it implies conflict at dec. lev. 4
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{a})$

Clause Minimization



- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$
- Can apply recursive minimization
- Learn clause $(\bar{w} \vee \bar{x})$

- Marked nodes: literals in learned clause [SB09]
- Trace back from c until marked nodes or new decision nodes
  - Learn clause only if search ends in marked nodes

Chapter 2

Optimization problems and SAT-Based Problem Solving

A set of constraints is overconstrained if it is inconsistent. In a given an unsatisfiable formula, there may be several explanations for its unsatisfiability. The goal of MaxSAT is to find largest subset of clauses that is satisfiable.

		Hard Clauses?	
		No	Yes
Weights?	No	Plain	Partial
	Yes	Weighted	Weighted Partial

- Must satisfy hard clauses, if any
- Compute set of satisfied soft clauses with maximum cost
  - Without weights, cost of each falsified soft clause is 1
- **Or**, compute set of falsified soft clauses with minimum cost (s.t. hard & remaining soft clauses are satisfied)
- **Note**: goal is to compute **set** of satisfied (or falsified) clauses; **not** just the cost **!**

2.1 MaxSAT Algorithms

2.1.1 Fu and Malik

$\varphi_W \leftarrow \varphi$ — Join Soft and Hard Constraints
while *true* **do**
  $(\text{st}, \varphi_C, \mathcal{A}) \leftarrow$ SAT (φ_W)
  if st $= UNSAT$ **then**
    $V_R \leftarrow \emptyset$
    foreach $\omega \in \varphi_C \wedge$ Soft (ω) **do** — Relax all soft clauses from unsatisfiable core
      $\omega_R \leftarrow \omega \cup \{r_i\}$ // r_i is a new relaxation variable
      $\varphi_W \leftarrow \varphi_W \backslash \{\omega\} \cup \{\omega_R\}$ — Relaxed soft clause remains soft
      $V_R \leftarrow V_R \cup \{r_i\}$
    $\varphi_W \leftarrow \varphi_W \cup$ CNF $(\sum_{r_i \in V_R} r_i \leq 1)$ — Cardinality constraint is hard
  else
    return $\mathcal{A}$ — Return when formula becomes SAT

2.1.2 MSU3

$V_R \leftarrow \emptyset$
$\varphi_W \leftarrow \varphi$
$LB \leftarrow 0$
while *true* **do**
  $(\text{st}, \varphi_C, \mathcal{A}) \leftarrow$ SAT $(\varphi_W \cup$ CNF $(\sum_{r_i \in V_R} r_i \leq LB))$
  if st $= UNSAT$ **then**
    foreach $\omega_i \in \varphi_C \wedge$ Soft (ω_i) **do**
      $\omega_R \leftarrow \omega_i \cup \{r_i\}$ // r_i is a new relaxation variable
      $\varphi_W \leftarrow \varphi_W \backslash \{\omega_i\} \cup \{\omega_R\}$ // ω_R is a **hard** clause
      $V_R \leftarrow V_R \cup \{r_i\}$
    $LB \leftarrow LB + 1$
  else
    return $\mathcal{A}$

2.2 Minimal Unsatisfiable Subsets

Given $\mathcal{F}$ unsatisfiable, $\mathcal{M} \subseteq \mathcal{F}$ is a MUS iff $\mathcal{M}$ is unsatisfiable and $\forall_{c \in \mathcal{M}}, \mathcal{M} \backslash \{c\}$ is satisfiable.

2.2.1 Algorithms

The following algorithms may be used to identify minimal unsatisfiable subsets.

Deletion-Based

```
Input  : Set 𝓡
Output: Minimal subset 𝓜
begin
    𝓜 ← 𝓡
    foreach c ∈ 𝓜 do
        if ¬SAT(𝓜 \ {c}) then
            𝓜 ← 𝓜 \ {c}                      // Remove c from 𝓜
    return 𝓜                                // Final 𝓜 is minimal set
end
```

Insertion-Based

```
Input  : Set 𝓡
Output: Minimal subset 𝓜
begin
    𝓜 ← ∅
    while 𝓡 ≠ ∅ do
        𝓢 ← ∅                                // Subset of 𝓡
        c_r ← ∅
        while SAT(𝓜 ∪ 𝓢) do
            c_i ← SelectRemoveElement(𝓡)
            𝓢 ← 𝓢 ∪ {c_i}
            c_r ← c_i
        𝓜 ← 𝓜 ∪ {c_r}                        // c_r is transition element
        𝓡 ← 𝓢 \ {c_r}
    return 𝓜                                // Final 𝓜 is minimal subset
end
```

Dichotomic

Input : Set $\mathcal{R} = \{c_1, \ldots, c_m\}$
Output: Minimal subset $\mathcal{M}$
begin
  $\mathcal{M} \leftarrow \emptyset$
  while SAT($\mathcal{M}$) **do**
    min $\leftarrow$ 1
    max $\leftarrow |\mathcal{R}|$
    while min $\neq$ max **do**
      mid $= \lfloor(\text{min} + \text{max})/2\rfloor$ // Execute binary search
      $\mathcal{S} \leftarrow \{c_1, \ldots, c_{\text{mid}}\}$ // Extract sub-sequence of $\mathcal{R}$
      if SAT($\mathcal{M} \cup \mathcal{S}$) **then**
        min $\leftarrow$ mid + 1
      else
        max $\leftarrow$ mid
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{c_{\text{min}}\}$ // c_{min} is transition element
    $\mathcal{R} \leftarrow \{c_1, \ldots, c_{\text{min}-1}\}$
  return $\mathcal{M}$ // Final $\mathcal{M}$ is minimal subset
end

2.3 Minimal Correction Subsets

$\mathcal{C} \subseteq \mathcal{F}$ is an MCS iff $\mathcal{F} \setminus \mathcal{C}$ is satisfiable and $\forall_{c \in \mathcal{C}}, \mathcal{F} \setminus (\mathcal{C} \setminus \{c\})$ is unsatisfiable.

2.3.1 Algorithms

The following algorithms may be used to identify minimal correction subsets.

Basic Linear Search

- Let $\mathcal{S} \subseteq \mathcal{F}$, such that $\mathcal{S} \nvDash \bot$, initially $\mathcal{S} = \emptyset$
- Let $\mathcal{C} \subseteq \mathcal{F}$, such that $\forall_{c \in \mathcal{C}}, \mathcal{S} \cup \{c\} \vDash \bot$, initially $\mathcal{C} = \emptyset$
- At each iteration, analyze one clause of $c \in \mathcal{F} \setminus (\mathcal{S} \cup \mathcal{C})$:
  - If $\mathcal{S} \cup \{c\} \vDash \bot$, then add c to $\mathcal{C}$, i.e. c is part of MCS
  - If $\mathcal{S} \cup \{c\} \nvDash \bot$, then add c to $\mathcal{C}$, i.e. c is part of MCS

There are $\mathcal{O}(m)$ calls to the oracle. An example:

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
$(x_1 \vee x_2)$	$(x_3 \vee x_4)$	$(\neg x_3 \vee \neg x_4)$	$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_5)	$(\neg x_5 \vee x_6)$	(x_2)

$\mathcal{C}$	$\mathcal{S}$	c	$\mathcal{S} \cup \{c\}$	SAT($\mathcal{S} \cup \{c\}$)	Outcome
$\emptyset$	$\emptyset$	c_1	c_1	1	Update $\mathcal{S}$
$\emptyset$	c_1	c_2	c_1c_2	1	Update $\mathcal{S}$
$\emptyset$	c_1c_2	c_3	$c_1..c_3$	1	Update $\mathcal{S}$
$\emptyset$	$c_1..c_3$	c_4	$c_1..c_4$	1	Update $\mathcal{S}$
$\emptyset$	$c_1..c_4$	c_5	$c_1..c_5$	1	Update $\mathcal{S}$
$\emptyset$	$c_1..c_5$	c_6	$c_1..c_6$	1	Update $\mathcal{S}$
$\emptyset$	$c_1..c_6$	c_7	$c_1..c_7$	1	Update $\mathcal{S}$
$\emptyset$	$c_1..c_7$	c_8	$c_1..c_8$	0	Update $\mathcal{C}$

- MCS: $\{c_8\}$

Clause D

- Pick an assignment and let $\mathcal{S} \subseteq \mathcal{F}$ be the satisfied clauses and $\mathcal{U} \subseteq \mathcal{F}$ be the falsified clauses, with $\mathcal{F} = \mathcal{S} \cup \mathcal{U}$
- Repeat:
  - Create clause $D = \cup_{l \in c, c \in \mathcal{U}} l$
  - If $\mathcal{S} \cup \{D\} \vDash \perp$, then $\mathcal{U}$ is MCS: Report MCS and terminate
  - If $\mathcal{S} \cup \{D\} \nvDash \perp$, then add to $\mathcal{S}$ the satisfied clauses in $\mathcal{U}$, remove from $\mathcal{U}$ the satisfied clauses and loop

There are $\mathcal{O}(m - r)$ calls to the oracle, where r is the size of the smallest MCS. An example:

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
$(x_1 \vee x_2)$	$(x_3 \vee x_4)$	$(\neg x_3 \vee \neg x_4)$	$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_5)	$(\neg x_5 \vee x_6)$	(x_2)

$\mathcal{S}$	$\mathcal{U}$	D	SAT($\mathcal{S} \cup \{D\}$)	Variables $= 1$
$\emptyset$	$\emptyset$	—	1	$\emptyset$
$c_3c_4c_7$	$c_1c_2c_5c_6c_8$	$\{x_1, \ldots, x_5\}$	1	$\{x_1, x_3\}$
$c_1..c_5c_7$	c_6c_8	$\{x_2, x_5\}$	1	$\{x_1, x_3, x_5, x_6\}$
$c_1..c_7$	c_8	$\{x_2\}$	0	—

- MCS: $\{c_8\}$

Chapter 3

Satisfiability Modulo Theories

Chapter 4

Answer Set Programming