The DIMACS format

- Comments
- Header [#variables(=5)] [#clauses(=7)]
- Variables are numbered 1 to *n*
- One line per clause '0' is a delimiter
- positive (negative) numbers are positive (negative) literals
  - $(\neg x_1 \vee x_3 \vee \neg x_5 \vee x_4)$

```
c This line is a comment.
p cnf 5 7
-1 3 -5 4 0
2 -3 0
1 5 0
-3 -4 0
-1 2 4 0
-2 0
2 -3 -5 0
```

Basic definitions

- Typename/classes
  - **Variable**: used for indexing → e.g., `int` from 0 to $n-1$
  - **Literal**: used for indexing → e.g., `int` from 0 to $2n-1$
  - **TruthValue**: three possibility $(\mathbf{true}, \mathbf{false}, \mathbf{undef}) \to \{1, 0, -1\}$
  - **Clause**: iterable list of literals

- Functions on variables
  - pos(**Variable**:x) $\mapsto$ **Literal** x (e.g., $2x+1$)
  - neg(**Variable**:x) $\mapsto$ **Literal** $\neg x$ (e.g., $2x$)

- Functions on literals
  - sign(**Literal**:l) $\mapsto \{\mathbf{false}, \mathbf{true}\}$ (e.g., $l\%2$)
  - not(**Literal**:l) $\mapsto \neg l$ (e.g., l^1)
  - var(**Literal**:l) $\mapsto x$ (e.g., $l/2$)

Data structures

- Data structures
  - model [**Variable** : x] ↦ **TruthValue** stores the current truth value of x
  - clauses [**Literal** : l] ↦ [**Clause**,...] list of clauses containing literal l
  - unit-literals stack of true literals (efficient push(**Literal**:l) and **Literal**:back() and pop-back())
- Functions
  - val(**Variable**:x) ↦ **TruthValue** truth value of variable x
  - falsified(**Literal**:l) ↦ Boolean literal is falsified in model
  - satisfied(**Literal**:l) ↦ Boolean literal is satisfied in model
- IN/OUT
  - Functions from-dimacs(**int**:d) ↦ **Literal** and to-dimacs(**Literal**:l) ↦ **int**
  - Functions read-dimacs() and write-dimacs()

Additionnal structures

Structure

watches [**Literal** : l] $\mapsto$ [**Clause**,...]	list of clauses watching literal l
int:to-propagate	the first non-unit-propagated literal in unit-literals

Functions

get-rank(**Clause**:c, **Literal**:l) $\mapsto$ $\{0, 1\}$	0 if l is the first watched in c, 1 otherwise
get-index(**Clause**:c, $\{0, 1\}$:r) $\mapsto$ **int**	index of the $(r + 1)$-th watched in c
set-watcher(**Clause**:c, **Literal**:l, $\{0, 1\}$:r)	set l as $(r + 1)$-th watcher of c
assign(**Literal**:l)	push l onto unit-literals and set model [var(l)]

Unit propagation algorithm (watched literals)

Algorithm: unit-propagate()

```
while to-propagate $<$ |unit-literals| do
    $l \leftarrow$ not(unit-literals [to-propagate ])
    if not unit-propagate($l$) then
        return false
    to-propagate $\leftarrow$ to-propagate $+ 1$
return true
```

Algorithm: unit-propagate(l)

Input: A non-unit propagated false literal l

Output: false in case of a contradiction, true otherwise

```
foreach $c \in$ watches[$l$] do
    $r \leftarrow$ get-rank($c$, $l$); $start \leftarrow i \leftarrow$ get-index($c$, $r$)
    $p \leftarrow c$[get-index($c$, $1$-$r$)]
    if not satisfied($p$) then
        while true do
            $i \leftarrow i + 1$
            if $i = |c|$ then $i \leftarrow 0$
            if $i = start$ then break
            if $c[i] \neq p$ then
                if not falsified($c[i]$) then
                    set-watcher($c$, $c[i]$, $r$)
                    break
        if $i = start$ then
            if falsified($p$) then return false
            assign($p$)
return true
```

Implementation

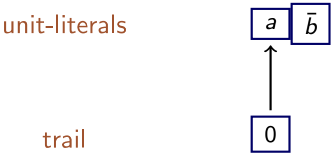
- Data structures
  - trail: stores the information required to backtrack
    - $|\text{trail}|$ is the current level in the search tree
    - trail(i) is the number of true literals at level i
    - Stack: push(),back(),pop-back() in $O(1)$
- Functions
  - unassign-back() pop l from unit-literals and reset model [var(l)]

Backtracking

unit-literals

trail

Backtracking



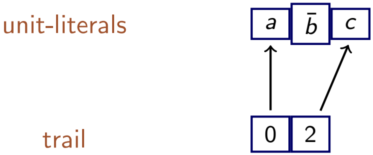
unit-literals

a $\bar{b}$

trail

0

Backtracking



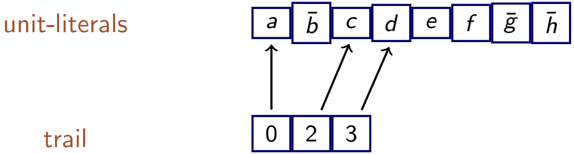
unit-literals

a $\bar{b}$ c

trail

0 2

Backtracking



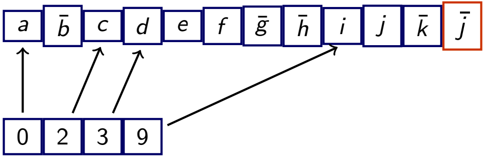
unit-literals

a $\bar{b}$ c d e f $\bar{g}$ $\bar{h}$

trail

0 2 3

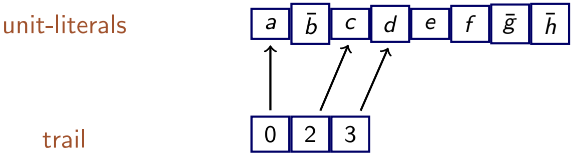
Backtracking



unit-literals: a $\bar{b}$ c d e f $\bar{g}$ $\bar{h}$ i j $\bar{k}$ $\bar{j}$

trail: 0 2 3 9

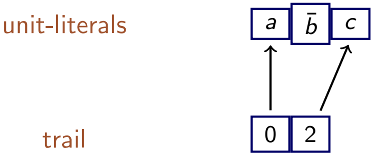
Backtracking



unit-literals	a	$\bar{b}$	c	d	e	f	$\bar{g}$	$\bar{h}$
trail	0	2	3					

- Backtrack to *decision level* 3

Backtracking



- Backtrack to *decision level* 3

- Backtrack to *decision level* 2

DPLL: Pseudocode

Algorithm: DPLL

```
while satisfiability = UNKNOWN do
    if unit-propagate() then
        if |unit-literals| = $n$ then  satisfiability ← SAT // a model is found ;
        else
            trail.push(|unit-literals|) // save current level
            assign(select-lit()) // add a new true literal
    else
        if |trail| = 0 then  satisfiability ← UNSAT // search tree exhausted ;
        else
            $d$ ← unit-literals[trail.back()] // retrieve previous decision
            while |unit-literals| > trail.back() do  unassign-back() // backtrack ;
            to-propagate ← trail.back();
            trail.pop-back();
            assign($\bar{d}$) // branch out of previous decision
```

Implementation

- Functions
  - unit-propagate() return the failed clause if there is an inconsistency (null otherwise)
  - backjump(**Clause**:*c*) conflict analysis and backjump

CDCL: Pseudocode

Algorithm: CDCL

```
while satisfiability $=$ UNKNOWN do
    $c =$ unit-propagate();
    if $c = Null$ then
        if $|$unit-literals$| = n$ then
            satisfiability $\leftarrow$ SAT ;
        else
            trail.push($|$unit-literals$|$);
            assign(select-lit());
    else
        if $|$trail$| = 0$ then
            satisfiability $\leftarrow$ UNSAT;
        else
            backjump($c$);
```

Algorithm: Backjump

```
Input: Conflict clause $c$
$learnt \leftarrow$ analyze-conflict($c$);
$l \leftarrow \arg\max_l(\{$level$(l) \mid l \in learnt\})$;
$lvl \leftarrow \max(\{$level$[p] \mid p \neq l \in learnt\})$;
while $|$unit-literals$| >$ trail$[lvl]$ do  unassign-back() ;
while $|$trail$| > lvl$ do trail.pop-back();
add($learnt$) ;        // $l$ should be watched by $learnt$!
assign($l$);
```

Implementation

- Data structures
  - level [**Variable** : x] ↦ **int** — the decision level at which x was unit propagated
  - reason [**Variable** : x] ↦ **Clause** — the clause responsible for x's unit propagation
    - ★ Change assign(**Literal**:l) and unassign-back(**Literal**:l)
- Functions
  - analyze-conflict(**Clause**:c) ↦ **Clause** — analyze conflict on clause c and returns a firt UIP clause
  - backjump(**Clause**:c) ↦ Boolean — returns **false** if the search tree is exhausted and **true** otherwise

Implementation

Algorithm: First UIP

```
Input: $c$
$seen \leftarrow \emptyset$ $learnt \leftarrow ()$;
$reason \leftarrow c$;
$n_{cur} \leftarrow 0$;
$l \leftarrow$ None;
$i \leftarrow |$unit-literals$| - 1$;
repeat
    foreach $p \neq l \in reason \setminus seen$ do
        add $p$ to $seen$;
        if level$[p] = |$trail$|$ then
            $n_{cur} \leftarrow n_{cur} + 1$;
        else
            add $p$ to $learnt$;
    while unit-literals$[i]$ $is$ $not$ $in$ $seen$ do $i \leftarrow i - 1$;
    $l \leftarrow$ unit-literals$[i]$;
    $reason \leftarrow$ reason$[l]$;
    $n_{cur} \leftarrow n_{cur} - 1$;
until $n_{cur} > 0$;
add the last explore literal $l$ to $learnt$;
```