Boolean Solving

Pascal Fontaine & Christophe Ringeissen

Procédures de décision et vérification de programmes: Lecture 5
SAT: NP-complete problem
No efficient solution for all cases on sequential computers (or \( P = NP \))
Truth tables: how many lines for \( n \) variables?
Many problems require many variables
Nowadays: \( 10^6 \) Boolean variables, \( 10^7 \) clauses \( (2^{(10^6)} = 10^{301029} > 10^{82}) \)
SAT4J, MiniSAT, Glucose, Crypto-MiniSAT, PicoSAT, ... 

Today’s lecture
- history: truth tables, DPLL (Davis, Putnam, Logemann, Loveland)
- now: CDCL (Conflict Driven Clause Learning)
  - Boolean propagation
  - Clause representation
  - Conflict analysis and non-chronological backtracking
Prerequisites / Notations

- **Boolean logic:**
  - Boolean (propositional) variable \((v)\), literal \((\ell)\)
    \[
    \overline{\overline{v}} = v, \quad \overline{v} = \neg v
    \]
  - clause, formula \((\neg, \land, \lor, \Rightarrow, \ldots)\)
  - interpretation, (un)satisfiable formula, valid formula
- Unit clause: clause with one literal only
- Empty clause: \(\Box\), unsatisfiable
- Resolution rule:
  \[
  \frac{C \lor \ell \quad C' \lor \overline{\ell}}{C' \lor C'}
  \]
You are chief of protocol for the embassy ball. The crown prince instructs you either to invite Peru or to exclude Qatar. The queen asks you to invite either Qatar or Romania or both. The king, in a spiteful mood, wants to snub either Romania or Peru or both. Is there a guest list that will satisfy the whims of the entire royal family?
You are chief of protocol for the embassy ball. The crown prince instructs you either to invite Peru or to exclude Qatar. The queen asks you to invite either Qatar or Romania or both. The king, in a spiteful mood, wants to snub either Romania or Peru or both. Is there a guest list that will satisfy the whims of the entire royal family?

\[(P \lor \neg Q)\]
A propositional problem (1/2)

You are chief of protocol for the embassy ball. The crown prince instructs you either to invite Peru or to exclude Qatar. The queen asks you to invite either Qatar or Romania or both. The king, in a spiteful mood, wants to snub either Romania or Peru or both. Is there a guest list that will satisfy the whims of the entire royal family?

\[(P \vee \neg Q) \land (Q \vee R)\]
A propositional problem (1/2)

You are chief of protocol for the embassy ball. The crown prince instructs you either to invite Peru or to exclude Qatar. The queen asks you to invite either Qatar or Romania or both. The king, in a spiteful mood, wants to snub either Romania or Peru or both. Is there a guest list that will satisfy the whims of the entire royal family?

\[(P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P)\]
A propositional problem (2/2)

\[ \varphi = \text{def } (P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P) \]

Truth table for \( \varphi \)
A propositional problem (2/2)

\[ \varphi = \text{def} \ (P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P) \]

Truth table for \( \varphi \)

<table>
<thead>
<tr>
<th>( P )</th>
<th>( Q )</th>
<th>( R )</th>
<th>( \varphi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
A propositional problem (2/2)

\[ \varphi = \text{def} \ (P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P) \]

Truth table for \(\varphi\)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>(\varphi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
DPLL: rule-based view

Let $S$ be a set of clauses

\[
\begin{align*}
\text{Unit Resolution} &:\quad \frac{S \cup \{\ell, C \lor \bar{\ell}\}}{S \cup \{\ell, C\}} \\
\text{(Unit Subsumption)} &:\quad \frac{S \cup \{\ell, C \lor \ell\}}{S \cup \{\ell\}} \\
\text{Splitting} &:\quad \frac{S}{S \cup \{v\} \mid S \cup \{\neg v\}} \quad \text{if } v \text{ is a variable occurring in } S
\end{align*}
\]

Exercise (mandatory): explain how a Boolean model can be extracted from the application of these rules

(Hint: think of the derivation trees and collect the various unit clauses...)
DPLL: example

\[(P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P)\]

Already a set of clauses:

\[
\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P\}
\]
DPLL: example

\[(P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P)\]

Already a set of clauses:

\[
\text{Split} \quad \frac{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P\}}{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\}} \quad | \quad \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}
\]
DPLL: example

\[(P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P)\]

Already a set of clauses:

\[
\begin{align*}
\text{Split} & \quad \frac{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P\}}{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\}} \\
& \quad \frac{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P\}}{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}}
\end{align*}
\]

And then only *Unit Resolution* rules

\[
\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\}
\]
DPLL: example

\[(P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P)\]

Already a set of clauses:

\[
\text{Split} \quad \frac{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P\}}{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\} \mid \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}}
\]

And then only *Unit Resolution* rules

\[
\frac{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\}}{\{P \lor \neg Q, Q \lor R, \neg R, P\}}
\]

Remarks:

Same two models again

Satisfiability procedure: find one model (and stop)

Much less sensitive to the number of variables than truth tables
DPLL: example

\[(P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P)\]

Already a set of clauses:

\[
\text{Split} \quad \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P\} \quad \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\} \quad \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}
\]

And then only *Unit Resolution* rules

\[
\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\} \quad \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}
\]

\[
\{P \lor \neg Q, Q \lor R, \neg R, P\} \quad \{P \lor \neg Q, Q, \neg R, P\} \quad \{P \lor \neg Q, Q, R, \neg P, \neg P\}
\]

Remarks:

Same two models again

Satisfiability procedure: find one model (and stop)

Much less sensitive to the number of variables than truth tables
DPLL: example

\[(P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P)\]

Already a set of clauses:

\[\text{Split} \quad \frac{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P\}}{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\} \mid \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}}\]

And then only Unit Resolution rules

\[\frac{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\}}{\{P \lor \neg Q, Q \lor R, \neg R, P\}} \quad \frac{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}}{\{\neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}}\]
DPLL: example

\[(P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P)\]

Already a set of clauses:

\[
\begin{align*}
\text{Split} & \quad \frac{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P\}}{
\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\} \mid \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}\}

& \quad \frac{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\}}{
\{P \lor \neg Q, Q \lor R, \neg R, P\}\}
\end{align*}
\]

And then only Unit Resolution rules

\[
\begin{align*}
\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\} & \quad \frac{\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}}{
\{\neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}\}

\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\} & \quad \frac{\{\neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}}{
\{\neg Q, R, \neg R \lor \neg P, \neg P\}\}
\end{align*}
\]
DPLL: example

\[(P \lor \neg Q) \land (Q \lor R) \land (\neg R \lor \neg P)\]

Already a set of clauses:

\[
\begin{align*}
\text{Split} & : \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P\} \\
& \quad | \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, \neg P\}
\end{align*}
\]

And then only Unit Resolution rules

\[
\begin{align*}
\{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, P\} & \quad | \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P, \neg P\} \\
\{P \lor \neg Q, Q \lor R, \neg R, P\} & \quad | \{\neg Q, Q \lor R, \neg R \lor \neg P, \neg P\} \\
\{P \lor \neg Q, Q, \neg R, P\} & \quad | \{\neg Q, R, \neg R \lor \neg P, \neg P\}
\end{align*}
\]

Remarks:

- Same two models again
- Satisfiability procedure: find one model (and stop)
- Much less sensitive to the number of variables than truth tables
Boolean formulas, CNF, DNF

- Boolean formulas: built with variables $\neg$, $\land$, $\lor$, $\Rightarrow$, $\ldots$
- Conjunctive Normal Form (CNF): (conjunctive) set of clauses
- Disjunctive Normal Form (DNF): (disjunctive) set of cubes

**Theorem**

*Every formula is logically equivalent to a CNF (DNF)*

**Remark:**

- Converting to DNF, then finding one satisfiable cube is a trivial satisfiability procedure
- checking the satisfiability of a (set of) cube(s) is linear
- so DNF conversion cannot be efficient (i.e. polynomial) or $P = NP$
- computing DNF of formula: negation of CNF of negation of formula
- so CNF conversion cannot be efficient
Consider
\[ \Phi = (a_1 \land \cdots \land a_m) \lor (b_1 \land \cdots \land b_n) \]

Equivalent CNF:
\[ \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{n} (a_i \lor b_j) \]

Equisatisfiable CNF:
\[ (X \lor Y) \land (X \iff a_1 \land \cdots \land a_m) \land (Y \iff b_1 \land \cdots \land b_n) \]

where \((X \iff a_1 \land \cdots \land a_m)\) can be represented as a conjunction of clauses (Exercise).

**Theorem**

*Every formula can be transformed in linear time into an equisatisfiable CNF*

Doesn’t it contradict the previous slide?
DPLL: from rules to algorithm

From rules to algorithm:

- a way to enumerate splittings
- some splittings may lead to closed branches (clause unsatisfied / empty clause derived)
- backtracking
- avoid copying sets of clauses and modifying clauses

Stack of assigned literals:

- some *decided*, others *propagated*
- *level*: number of decided literals on stack
- some clause become unit
  - all literals but one assigned to false, last literal not assigned
    \[\rightarrow\] new propagated literal on the stack
- clause may be falsified: all literals assigned to false
  \[\rightarrow\] backtrack the last decision
- never mind about satisfied clause
DPLL: algorithmic view

1: procedure SAT(C)
2: while ⊤ do
3: if PROPAGATE() then
4: if ¬DECIDE() then
5: return SAT
6: continue
7: if level = 0 then
8: return UNSAT
10: BACKTRACK()

Write successive stacks for runs on

- \{P \lor \neg Q, Q \lor R, \neg R \lor \neg P\}
- \{a \lor b, \neg b \lor c \lor d, \neg b \lor e, \neg d \lor \neg e \lor f, a \lor c \lor f, \neg a \lor g, \neg g \lor b, \neg h \lor j, \neg i \lor k\}
DPLL: abstract view

Rules handle a data-structure $M \parallel F$ where $M$ is a partial assignment of Boolean variables, and $F$ is a set of clauses

- **Propagate** $M \parallel F, C \lor \ell \vdash M \ell \parallel F, C \lor \ell$
  
  if $M \models \neg C$, $\ell$ undefined in $M$

- **Decide** $M \parallel F \vdash M \ell^d \parallel F$
  
  if $\ell$ or $\bar{\ell}$ in $F$, $\ell$ undefined in $M$

- **Fail** $M \parallel F, C \vdash \bot$
  
  if $M \models \neg C$, no decision literals in $M$

- **Backtrack** $M \ell^d N \parallel F, C \vdash M \bar{\ell} \parallel F, C$
  
  if $\begin{cases} M \ell^d N \models \neg C \\ no \text{ decision literals in } N \end{cases}$
Decisions

1: `procedure SAT(C)`
2: `while ⊤ do`
3: `if PROPAGATE() then`
4: `if ¬DECIDE() then`
5: `return SAT`
6: `continue`
7: `if level = 0 then`
8: `return UNSAT`
10: `BACKTRACK()`

Concretely, `DECIDE`:

Many heuristics
Decisions

1: procedure SAT(C)
2: while ⊤ do
3:     if PROPAGATE() then
4:         if ¬DECIDE() then
5:             return SAT
6:             continue
7:     if level = 0 then
8:         return UNSAT
10: B acktrack()  

Concretely, Decide:
  • maintain a “set” of unassigned literals

Many heuristics
Decisions

1: procedure SAT(C)
2:   while ⊤ do
3:     if PROPAGATE() then
4:       if ¬DECIDE() then
5:         return SAT
6:       continue
7:     if level = 0 then
8:       return UNSAT
10:   BACKTRACK()

Concretely, Decide:

- maintain a “set” of unassigned literals
- add and remove literals as needed

Many heuristics
Decisions

1: `procedure SAT(C)`
2: `while $\top$ do`
3: `if PROPAGATE() then`
4: `if $\neg$DECIDE() then`
5: `return SAT`
6: `continue`
7: `if level = 0 then`
8: `return UNSAT`
10: `BACKTRACK()`

Concretely, **Decide:**
- maintain a “set” of unassigned literals
- add and remove literals as needed
- can be implemented efficiently
  constant time for add and remove

Many heuristics
Decisions

1: \textbf{procedure} SAT(C)
2: \hspace{1em} \textbf{while} $\top$ \textbf{do}
3: \hspace{2em} \textbf{if} Propagate() \textbf{then}
4: \hspace{3em} \textbf{if} $\neg$Decide() \textbf{then}
5: \hspace{4em} \textbf{return} SAT
6: \hspace{4em} \textbf{continue}
7: \hspace{2em} \textbf{if} level = 0 \textbf{then}
8: \hspace{3em} \textbf{return} UNSAT
10: \textbf{Backtrack()}

Concretely, Decide:

- maintain a “set” of unassigned literals
- add and remove literals as needed
- can be implemented efficiently
- constant time for add and remove
- if selection of variables based on activity, then heap (constant remove, $\ln n$ add)

Many heuristics
Propagación (1/2)

Concretamente, Propagate:

1: procedure SAT(C)
2:     while ⊤ do
3:         if Propagate() then
4:             if ¬Decide() then
5:                 return SAT
6:             continue
7:         if level = 0 then
8:             return UNSAT
10:     Backtrack()
Propagation (1/2)

Concretely, Propagate:

- check every clause for conflicts or units, terminate as soon as conflict, or when no more unit
Propagation (1/2)

Concretely, Propagate:
- check every clause for conflicts or units, terminate as soon as conflict, or when no more unit
- impractical!

1: procedure SAT(C)
2: while ⊤ do
3: if Propagate() then
4: if ¬Decide() then
5: return SAT
6: continue
7: if level = 0 then
8: return UNSAT
10: Backtrack()
Propagation (1/2)

1: **procedure** SAT($C$)
2: **while** ⊤ **do**
3: **if** Propagate() **then**
4: **if** ¬Decide() **then**
5: return SAT
6: continue
7: **if** level = 0 **then**
8: return UNSAT
10: **Backtrack()**

Concretely, Propagate:

- check every clause for conflicts or units, terminate as soon as conflict, or when no more unit
- impractical!
- set a pointer on literal stack to last propagated literal. For each $\ell$ to propagate, investigate all clauses containing $\overline{\ell}$. Only those may become unit (or unsat) because of assignment.
Propagation (1/2)

1: procedure SAT($C$)
2: while $\top$ do
3: if Propagate() then
4: if $\neg$Decide() then
5: return SAT
6: continue
7: if level = 0 then
8: return UNSAT
10: Backtrack()

Concretely, Propagate:

- check every clause for conflicts or units, terminate as soon as conflict, or when no more unit
- impractical!
- set a pointer on literal stack to last propagated literal. For each $\ell$ to propagate, investigate all clauses containing $\ell$. Only those may become unit (or unsat) because of assignment.
- one can build (linear time) an index of clauses by literal
Propagation (1/2)

1: procedure SAT($C$)
2: while $\top$ do
3: if Propagate() then
4: if $\neg$Decide() then
5: return SAT
6: continue
7: if level = 0 then
8: return UNSAT
10: Backtrack()

Concretely, Propagate:

- check every clause for conflicts or units, terminate as soon as conflict, or when no more unit
- impractical!
- set a pointer on literal stack to last propagated literal. For each $\ell$ to propagate, investigate all clauses containing $\ell$. Only those may become unit (or unsat) because of assignment.
- one can build (linear time) an index of clauses by literal
- is it necessary to examine the clause each time one of its literals becomes false?
Propagation (2/2)

- is it necessary to examine the clause for each time one of its literals becomes false? No!
Propagation (2/2)

- is it necessary to examine the clause for each time one of its literals becomes false? No!
- only the penultimate one (then the last would be propagated)
  only examine when just one literal remains not assigned
Propagation (2/2)

- Is it necessary to examine the clause for each time one of its literals becomes false? No!
- Only the penultimate one (then the last would be propagated)
  only examine when just one literal remains not assigned
- As long as two literals are unassigned, do not care for the clause
Propagation (2/2)

- is it necessary to examine the clause for each time one of its literals becomes false? No!
- only the penultimate one (then the last would be propagated) only examine when just one literal remains not assigned
- as long as two literals are unassigned, do not care for the clause
- one can build (linear time) an index of clauses by literal
Propagation (2/2)

- is it necessary to examine the clause for each time one of its literals becomes false? No!
- only the penultimate one (then the last would be propagated)
- only examine when just one literal remains not assigned
- as long as two literals are unassigned, do not care for the clause
- one can build (linear time) an index of clauses by literal
- it is only necessary to index two of its unassigned literals
Propagating (2/2)

- Is it necessary to examine the clause for each time one of its literals becomes false? No!
- Only the penultimate one (then the last would be propagated).
  Only examine when just one literal remains not assigned.
- As long as two literals are unassigned, do not care for the clause.
- One can build (linear time) an index of clauses by literal.
- It is only necessary to index two of its unassigned literals.
- Clauses indexed by $\ell$ have to be reindexed as soon as $\ell$ becomes false.
Propagation (2/2)

- Is it necessary to examine the clause for each time one of its literals becomes false? No!
- Only the penultimate one (then the last would be propagated)
  only examine when just one literal remains not assigned
- As long as two literals are unassigned, do not care for the clause
- One can build (linear time) an index of clauses by literal
- It is only necessary to index two of its unassigned literals
- Clauses indexed by $\ell$ have to be reindexed as soon as $\ell$ becomes false
- They have to be indexed by another literal
  always by two different literals, called *watched literals*
Propagation (2/2)

- is it necessary to examine the clause for each time one of its literals becomes false? No!
- only the penultimate one (then the last would be propagated)
  only examine when just one literal remains not assigned
- as long as two literals are unassigned, do not care for the clause
- one can build (linear time) an index of clauses by literal
- it is only necessary to index two of its unassigned literals
- clauses indexed by $\ell$ have to be reindexed as soon as $\ell$ becomes false
- they have to be indexed by another literal
  always by two different literals, called watched literals
- if one cannot find a replacement for falsified literal, then the other watcher has to be propagated
Conflict analysis

- depending on decisions, the same dead end may be tried again and again (TODO: example)
- would be much better to remember the very reason why conflict: new clause
- then forget about backtracking and changing decision. Just add clause, backtrack to when it is propagating
- see Heule
DPLL: algorithmic view

1: procedure SAT(C)
2: while ⊤ do
3:    if PROPAGATE() then
4:        if ¬DECIDE() then
5:            return SAT
6:        continue
7:    if level = 0 then
8:        return UNSAT
9: 10: BACKTRACK()
1: `procedure SAT(C)`
2: `while ⊤ do`
3: `if PROPAGATE() then`
4: `if ¬DETERMINE() then`
5: `return SAT`
6: `continue`
7: `if level = 0 then`
8: `return UNSAT`
9: `ANALYSE()`
10: `BACKTRACK()`

- **PROPAGATE**: find unit clauses repeatedly and push literals on the stack. Returns ⊥ iff unsatisfied clause
- **DETERMINE**: choses one non assigned literal, push on stack. Returns ⊥ iff no literal
- **ANALYSE**: analyse the conflict from propagate, create conflict clause, add it in the set of clauses
- **BACKTRACK**: backtrack (eliminate literals from stack) until conflict clause is unit
DPLL: abstract view

Rules handle a data-structure $M \ || \ F$ where $M$ is a partial assignment of Boolean variables, and $F$ is a set of clauses.

**Propagate**

$M \ || \ F, C \lor \ell \vdash M \ \ell \ || \ F, C \lor \ell$

if $M \models \neg C$, $\ell$ undefined in $M$

**Decide**

$M \ || \ F \vdash M \ \ell^d \ || \ F$

if $\ell$ or $\overline{\ell}$ in $F$, $\ell$ undefined in $M$

**Fail**

$M \ || \ F, C \vdash \bot$

if $M \models \neg C$, no decision literals in $M$

**Backtrack**

$M \ \ell^d \ N \ || \ F, C \vdash M \ \overline{\ell} \ || \ F, C$

if $
\begin{cases} 
M \ \ell^d \ N \models \neg C \\
\text{no decision literals in } N
\end{cases}$
CDCL: abstract view

Propagate, Decide, Fail as before

Learn

\[ M \parallel F \vdash M \parallel F, C \]

if \[
\begin{cases} 
\text{each atom of } C \text{ in } F \text{ or in } M \\
F \models C 
\end{cases}
\]

Backjump

\[ M \ell^d N \parallel F, C \vdash M \ell' \parallel F, C \]

\[
\begin{cases} 
M \ell^d N \models \neg C \\
\exists C', \ell': F, C \models C' \lor \ell' 
\end{cases}
\]

if \[
\begin{cases} 
M \models \neg C' \\
\ell' \text{ undefined in } M \\
\ell' \text{ or } \ell' \text{ in } F \text{ or in } M \ell^d N 
\end{cases}
\]

Forget

\[ M \parallel F, C \vdash M \parallel F \]

if \[ F \models C \]
Conclusion

- only a brief glimpse on the practical aspects of SAT solving
- many more tricks, hacks, techniques, heuristics, and even theories
- oldest logic problem, still very active research subject (particularly since mid 90s)
SAT Solving: suggested reading

- Eén, Sörensson: MiniSAT 2011 (TODO)
SAT Solvers

- input: CNF. File extension .cnf
  - Boolean variable: number \( \geq 1 \)
  - literal either positive (represented by positive number)
    - negative (represented by negative number)
  - clause: series of numbers separated by spaces, terminated by 0
  - cnf: series of clauses
  - file starts with \texttt{p cnf X Y (X var., Y clauses)}
  - comments start by \texttt{c}

\textit{Exemple}

\[
\begin{align*}
 p \lor q \\
 p \lor r \\
 \neg q \lor \neg r \\
 \neg p 
\end{align*}
\]
SAT Solvers

- input: CNF. File extension .cnf
  - Boolean variable: number \( \geq 1 \)
  - literal either positive (represented by positive number)
  - negative (represented by negative number)
  - clause: series of numbers separated by spaces, terminated by 0
  - cnf: series of clauses
  - file starts with \( p \) cnf X Y (X var., Y clauses)
  - comments start by \( c \)

**Exemple**

\( p \rightarrow 1, q \rightarrow 2, r \rightarrow 3 \)

\[
\begin{align*}
p \lor q \\
p \lor r \\
\neg q \lor \neg r \\
\neg p
\end{align*}
\]
**SAT Solvers**

- **input**: CNF. File extension .cnf
  - Boolean variable: number $\geq 1$
  - literal either positive (represented by positive number)
    - literal either negative (represented by negative number)
  - clause: series of numbers separated by spaces, terminated by 0
  - cnf: series of clauses
  - file starts with `p cnf X Y (X var., Y clauses)`
  - comments start by `c`

**Exemple**

```
p → 1, q → 2, r → 3

p ∨ q → 1 2 0
p ∨ r → 1 3 0
¬q ∨ ¬r → -2 -3 0
¬p → -1 0
```
SAT Solvers

- input: CNF. File extension .cnf
  - Boolean variable: number ≥ 1
  - literal either positive (represented by positive number) negative (represented by negative number)
  - clause: series of numbers separated by spaces, terminated by 0
  - cnf: series of clauses
  - file starts with p cnf X Y (X var., Y clauses)
  - comments start by c

**Exemple**

\[
p → 1, \ q → 2, \ r → 3
\]

\[
p \land q \rightarrow 1 \ 2 \ 0
\]

\[
p \land r \rightarrow 1 \ 3 \ 0
\]

\[
¬q \land ¬r \rightarrow -2 \ -3 \ 0
\]

\[
¬p \rightarrow -1 \ 0
\]
A farmer wants to cross a river in his small boat, with a wolf, a goat and a cabbage. He should make sure:

- to only take one animal or object with him, the boat being so small
- not to leave the wolf and the goat alone (or no more goat)
- not to leave the goat and the cabbage alone (or no more cabbage)

Is this possible? With how many crossings?
Use logic to encode the problem.
Use logic to encode the problem.
Four variables (that can be true or false):

- $f$ farmer
- $w$ wolf
- $g$ goat
- $c$ cabbage

E.g. $f$ is true if $f$ is on the left side, false if on the right side

- We start with
- We want to finish with
Use logic to encode the problem.
Four variables (that can be true or false):
- $f$ farmer
- $w$ wolf
- $g$ goat
- $c$ cabbage

E.g. $f$ is true if $f$ is on the left side, false if on the right side

- We start with $\text{init} \defines f \land w \land g \land c$
- We want to finish with
Use logic to encode the problem.
Four variables (that can be true or false):
- $f$ farmer
- $w$ wolf
- $g$ goat
- $c$ cabbage

E.g. $f$ is true if $f$ is on the left side, false if on the right side

- We start with $\text{init} = \text{def } f \land w \land g \land c$
- We want to finish with $\text{fin} = \text{def } \neg f \land \neg w \land \neg g \land \neg c$
The Wolf, the Goat, and the Cabbage (3/5)

How to express there is some danger

A state is dangerous if the wolf and the goat (or the goat and the cabbage) are on one bank, and the farmer on the other.

Formally:

$$\text{danger} = (w \equiv g) \land (w \equiv \neg f) \lor (g \equiv c) \land (g \equiv \neg f)$$
How to express there is some danger

A state is *dangerous* if the wolf and the goat (or the goat and the cabbage) are on one bank, and the farmer on the other

Formally:
How to express there is some danger

A state is *dangerous* if the wolf and the goat (or the goat and the cabbage) are on one bank, and the farmer on the other.

Formally:

\[
\text{danger} \;=\; \text{def} \;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;\;}
To find out if it is possible to find a solution with \( n \) crossing, we will use \( n + 1 \) copies of the variables \( f_i, w_i, g_i, c_i \).

First, let’s write the formula corresponding the \( i\)-th crossing of the farmer.

\[
cross_i \overset{\text{def}}{=} \left( \land \left( \lor \left( \lor \right) \right) \right)
\]
The Wolf, the Goat, and the Cabbage (4/5)

To find out if it is possible to find a solution with \( n \) crossing, we will use \( n + 1 \) copies of the variables \( f_i, w_i, g_i, c_i \).

First, let’s write the formula corresponding the \( i \)-th crossing of the farmer.

- The farmer is crossing so \( f_{i+1} \) and \( \neg f_i \) should be different

\[
\text{cross}_i \equiv \text{def} \\
\land \left( \\
\lor \\
\lor \\
\right)
\]
To find out if it is possible to find a solution with $n$ crossing, we will use $n + 1$ copies of the variables $f_i, w_i, g_i, c_i$.

First, let’s write the formula corresponding the $i$-th crossing of the farmer.

- The farmer is crossing so $f_{i+1}$ and $\neg f_i$ should be different

\[
cross_i \overset{\text{def}}{=} (f_{i+1} \equiv \neg f_i) \land \left( \lor \left( \lor \right) \right)
\]
To find out if it is possible to find a solution with $n$ crossing, we will use $n + 1$ copies of the variables $f_i, w_i, g_i, c_i$.

First, let’s write the formula corresponding the $i$-th crossing of the farmer.

- The farmer is crossing so $f_{i+1}$ and $\neg f_i$ should be different
- Only one animal/object is changing bank; two (or more) variables among $w, g, c$ should stay the same

\[
\text{cross}_i \overset{\text{def}}{=} (f_{i+1} \equiv \neg f_i) \\
\land \left( \lor \left( \lor \right) \right)
\]

\[
\land \left( \lor \left( \lor \right) \right)
\]
The Wolf, the Goat, and the Cabbage (4/5)

To find out if it is possible to find a solution with $n$ crossing, we will use $n + 1$ copies of the variables $f_i$, $w_i$, $g_i$, $c_i$.

First, let’s write the formula corresponding the $i$-th crossing of the farmer.

- The farmer is crossing so $f_{i+1}$ and $\neg f_i$ should be different
- Only one animal/object is changing bank; two (or more) variables among $w$, $g$, $c$ should stay the same

$$\text{cross}_i \triangleq \text{def} \quad (f_{i+1} \equiv \neg f_i)$$

$$\land \left( \right.\left((w_{i+1} \equiv w_i) \land (g_{i+1} \equiv g_i)\right)$$

$$\lor \left. \left((w_{i+1} \equiv w_i) \land (c_{i+1} \equiv c_i)\right) \right)$$

$$\lor \left. \left((g_{i+1} \equiv g_i) \land (c_{i+1} \equiv c_i)\right) \right)$$
We want to write a formula to encode solutions in $n$ crossings.
The Wolf, the Goat, and the Cabbage (5/5)

We want to write a formula to encode solutions in $n$ crossings

- Starting and ending state have been defined

\[
\text{init}_1 \land fin_{n+1} \\
\land \\
\land
\]
The Wolf, the Goat, and the Cabbage (5/5)

We want to write a formula to encode solutions in $n$ crossings

- Starting and ending state have been defined
- Two successive states should correspond to one crossing

\[
\begin{align*}
\text{init}_1 & \land \text{fin}_{n+1} \\
\land & \text{cross}_1 \land \text{cross}_2 \land \ldots \land \text{cross}_n
\end{align*}
\]
We want to write a formula to encode solutions in $n$ crossings

- Starting and ending state have been defined
- Two successive states should correspond to one crossing
- No state should be dangerous

\[
init_1 \land fin_{n+1} \\
\land cross_1 \land cross_2 \land \ldots \land cross_n \\
\land \neg danger_1 \land \neg danger_2 \land \ldots \land \neg danger_{n+1}
\]
We want to write a formula to encode solutions in $n$ crossings

- Starting and ending state have been defined
- Two successive states should correspond to one crossing
- No state should be dangerous

\[
\begin{align*}
&\text{init}_1 \land \text{fin}_{n+1} \\
\land &\text{cross}_1 \land \text{cross}_2 \land \ldots \land \text{cross}_n \\
\land &\neg \text{danger}_1 \land \neg \text{danger}_2 \land \ldots \land \neg \text{danger}_{n+1}
\end{align*}
\]

A SAT solver can find out that there is no solution in 4 traversals, but that 6 traversals are enough.
Sudoku (1/3)

1  
---
2 3

- at line $i$, column $j$, is number $x$ 
  $(p_{i,j,x})$
Sudoku (1/3)

- Number 1 is at line 1, column 1
- Number 3 is at line 2, column 4
- At location (1,2), there is at most one number
  \[ \neg p_{1,2,1} \lor \neg p_{1,2,2} \lor \neg p_{1,2,3} \lor \neg p_{1,2,4} \]
Sudoku (1/3)

- Number 1 is at line 1, column 1
  \( p_{1,1,1} \)

- at line \( i \), column \( j \), is number \( x \)
  \( p_{i,j,x} \)
Sudoku (1/3)

- Number 1 is at line 1, column 1
  \(- p_{1,1,1}\)
- Number 3 is at line 2, column 4
- Number 2 is at line 2, column 1
- Number 3 is at line 2, column 4

\(p_{i,j,x}\)

at line \(i\), column \(j\), is number \(x\)
Sudoku (1/3)

- Number 1 is at line 1, column 1
  \[ p_{1,1,1} \]
- Number 3 is at line 2, column 4
  \[ p_{2,4,3} \]

- at line \( i \), column \( j \), is number \( x \)
  \[ (p_{i,j,x}) \]
Sudoku (1/3)

- Number 1 is at line 1, column 1
  \[ p_{1,1,1} \]
- Number 3 is at line 2, column 4
  \[ p_{2,4,3} \]
- At location (1, 2), there is at most one number

- at line \( i \), column \( j \), is number \( x \)
  \( (p_{i,j,x}) \)
Number 1 is at line 1, column 1
\( p_{1,1,1} \)

Number 3 is at line 2, column 4
\( p_{2,4,3} \)

At location (1, 2), there is at most one number
\[
\neg p_{1,2,1} \lor \neg p_{1,2,2} \\
\neg p_{1,2,1} \lor \neg p_{1,2,3} \\
\neg p_{1,2,1} \lor \neg p_{1,2,4} \\
\neg p_{1,2,2} \lor \neg p_{1,2,3} \\
\neg p_{1,2,2} \lor \neg p_{1,2,4} \\
\neg p_{1,2,3} \lor \neg p_{1,2,4}
\]
at line $i$, column $j$, is number $x$

$(p_{i,j,x})$
Sudoku (2/3)

- At line $i$, column $j$, is number $x$ ($p_{i,j,x}$)

- At location $(1, 2)$, there is either 1, 2, 3, or 4 (repeat $\forall$ location)
Sudoku (2/3)

- At location (1, 2), there is either 1, 2, 3, or 4 (repeat \(\forall\) location)
  \[p_{1,2,1} \lor p_{1,2,2} \lor p_{1,2,3} \lor p_{1,2,4}\]

- at line \(i\), column \(j\), is number \(x\) 
  \((p_{i,j,x})\)
Sudoku (2/3)

- At line $i$, column $j$, is number $x$ \((p_{i,j,x})\)

- At location \((1, 2)\), there is either 1, 2, 3, or 4 (repeat \(\forall\) location) \(p_{1,2,1} \lor p_{1,2,2} \lor p_{1,2,3} \lor p_{1,2,4}\)

- Number 1 should be somewhere at line 1 (repeat \(\forall\) number, line...)
Sudoku (2/3)

- At location (1, 2), there is either 1, 2, 3, or 4 (repeat ∀ location)
  \[ p_{1,2,1} \lor p_{1,2,2} \lor p_{1,2,3} \lor p_{1,2,4} \]

- Number 1 should be somewhere at line 1 (repeat ∀ number, line...)
  \[ p_{1,1,1} \lor p_{1,2,1} \lor p_{1,3,1} \lor p_{1,4,1} \]

- at line \( i \), column \( j \), is number \( x \) 
  \( p_{i,j,x} \)
Sudoku (2/3)

- At line $i$, column $j$, is number $x$ \( (p_{i,j,x}) \)

- At location (1, 2), there is either 1, 2, 3, or 4 (repeat \( \forall \) location)
  \[ p_{1,2,1} \lor p_{1,2,2} \lor p_{1,2,3} \lor p_{1,2,4} \]

- Number 1 should be somewhere at line 1 (repeat \( \forall \) number, line . . .)
  \[ p_{1,1,1} \lor p_{1,2,1} \lor p_{1,3,1} \lor p_{1,4,1} \]

- Number 1 should be at most once at line 1
Sudoku (2/3)

- At location (1, 2), there is either 1, 2, 3, or 4 (repeat ∀ location)
  
  \[ p_{1,2,1} \lor p_{1,2,2} \lor p_{1,2,3} \lor p_{1,2,4} \]

- Number 1 should be somewhere at line 1 (repeat ∀ number, line...)
  
  \[ p_{1,1,1} \lor p_{1,2,1} \lor p_{1,3,1} \lor p_{1,4,1} \]

- Number 1 should be at most once at line 1
  
  \[ \neg p_{1,1,1} \lor \neg p_{1,2,1} \\
  \neg p_{1,1,1} \lor \neg p_{1,3,1} \\
  \neg p_{1,1,1} \lor \neg p_{1,4,1} \\
  \neg p_{1,2,1} \lor \neg p_{1,3,1} \\
  \neg p_{1,2,1} \lor \neg p_{1,4,1} \\
  \neg p_{1,3,1} \lor \neg p_{1,4,1} \]
Sudoku (3/3)

Demo