# SAT solverek alkalmazása

Iván Szabolcs

# Informatika: a SAT probléma

- a SAT (satisfiability) probléma: input egy (ítéletkalkulusbeli) logikai formula, adjuk meg a változóknak egy kielégítő értékadását! (Ha van.)
- A probléma nehéz: $n$ változó – $2^n$ lehetőség bonya: „**NP**-nehéz"
- $2^{100}$ művelet elvégzése kb. $4$ évbe telne a Föld összes jelenlegi számítási kapacitását egyszerre használva (becslés, 2017)
  - $2^{120}$: $4$ millió év
  - $2^{200}$: $5 \times 10^{30}$ év – annyi nincs
- Heurisztikák kellenek és a keresési tér vágása
- Minden évben van SAT Competition, amin SAT solvereket versenyeztetnek több tracken
- 2016 ősszel: egy laptopon kb. $900$-változós formulát kb. $0.5$sec alatt soon
- A XXI. században intenzíven fejlődik a terület
  - 1995-es évek: kb. $100$ változóra $200$ feltétel
  - 2010: kb. $1.000.000$ változóra $5.000.000$ feltétel

# SAT, de minek?

- SAT Competition egyik track: Application Benchmark
  - Erre cégek küldik be az őket érdeklő nehéz problémákat logikai formulával leírva
  - 2016-ban pl. a francia vasúthálózat forgalomirányításáról is kérdezték, biztonságos-e
- IBM: egy hardware egység teljesíti-e a specifikációt
  - 2006-os SAT Race-re (ez Industrial Only) formulával leírva: $170.000$ változó, $725.000$ feltétel
  - erre egyébként $500+$ ipari benchmark érkezett
- Az Intelnél, IBM-nél és Microsoftnál ma a SAT solving a domináns technológia a hardware tervek verifikációjára
- AI Planning: döntéshozatal egy vagy több cél elérése érdekében, erőforrást optimalizálva – ez is felírható formulával
- Handbook of SAT - 2009, 966 oldal

# SAT, de minek?

- Nagyon sok „kombinatorikus” keresési problémát fel lehet írni SAT problémaként.
- Egy use case-t látni fogunk hamarosan: a tatami coveringet.

Egy másik gyakori alkalmazás: a Code Contractok ellenőrzése (ezen a kurzuson: „Hoare kalkulus”)

# Code contractok – C + ACSL (ANSI C Spec Lang)

```
int f(int A, int B) {
  int Q = 0, R = A;
  while (R >= B) {
    R -= B; Q++;
  }
  return R;
}
```

A jobb oldali annotációk segítségével **automatikusan** ellenőrizhető, hogy a függvény implementációja helyes

```
//@ requires A>=0 && B>0;
//@ ensures \result == A mod B;
int f(int A, int B) {
  int Q = 0, R = A;
  //@ assert A>=0 && B>0 && Q=0 && R==A;
  while (R >= B) {
    //@ assert A>=0 && B>0 && R>=B &&
           A==Q*B+R;
    R -= B; Q++;
  }
  //@ assert A>=0 && B>0 && R>=0 && R<B
         && A==Q*B+R;
  return R;
}
```

# Code contractok – Java + ESC (Extended Static Checker)

```
public class OrderedArray {
  int a[];
  int nb;
  //@invariant nb >= 0 && nb <= 20
  //@invariant (\forall int i; (i >= 0 && i < nb-1) ==> a[i] <= a[i+1])
  public OrderedArray() { a = new int[20]; nb = 0; }
  public void add(int v) {
    if (nb >= 20) return;
    int i;
    for (i = nb; i > 0 && a[i-1] > v; i--) a[i] = a[i-1];
    a[i] = v; nb++;
  }
}
```

Az ESC automatikusan le tudja ellenőrizni, hogy egy OrderedArray a teljes élettartama során tényleg rendezett sorrendben fogja tárolni az elemeket

# Code contractok – Dafny (MS, C#)

```
function Fib( n: nat ) : nat {
  if ( n < 2 ) then n else Fib( n-1 ) + Fib( n-2 )
}
method computeFib( n: nat ) returns ( x: nat )
  ensures x == Fib( n );
{
  var i:=0;
  x := 0;
  var y:=1;
  while( i < n )
  invariant 0 <= i <= n;
  invariant x == Fib( i );
  invariant y == Fib( i + 1 );
  {
    x,y := y,x+y;
    i := i+1;
  }
}
```

Rekurzív, helyes $O(1.62^n)$

Iteratív, gyors $O(n)$

automatikusan ellenőrizhető: ugyanazt adják

némi hinttel

# Ítéletkalkulus vs. elsőrendű logika

A kurzus első felében ítéletkalkulussal fogunk foglalkozni:

- a változók a $\{0, 1\}$ halmazból kapnak értéket
  ($0$: hamis, $1$: igaz – igazságértékek, bitek)
- a formulák változókból épülnek fel ítéletlogikai összekötő jelek
  (konnektívák, mint a $\neg$ és a $\vee$, sőt a $\downarrow$) alkalmazásával (zárójelezve)

A második felében elsőrendű logikával:

- a változók objektumok egy halmazából kapnak értékeket
- a konnektívákon kívül kvantorok is használhatóak lesznek (mint a $\forall$)
- az objektumokat függvények fogják újabb objektumokba, és predikátumok fogják igazságértékké transzformálni

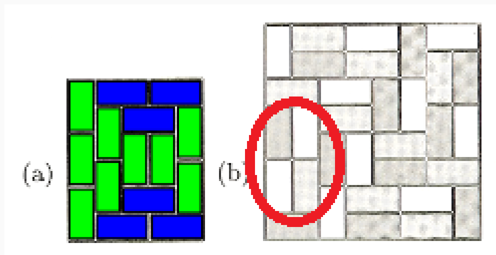
Az ítéletkalkulust hívjuk még ítéletlogikának vagy nulladrendű, esetleg propozicionális logikának is.

Az elsőrendű logikát pedig predikátumkalkulusnak.

# Honnan lesz formulánk?

Vegyük pl. a következő kombinatorikus keresési feladatot.

- Adott egy $n \times m$-es téglalap, melyet $2 \times 1$-es dominókkal (forgatni ér) szeretnénk lefedni.
- A lefedésnek hogy „szép” legyen, tatami lefedésnek kell lennie: négy dominó nem találkozhat egy sarkon.
- Néhány dominó előre fel van rakva a táblára.
- Adjunk meg egy tatami lefedését a táblának, melyben a megadott dominók a megadott módon szerepelnek!




source: https://canadam.math.ca/2013/slides/Erickson.Alejandro.Tatami.pdf

# Modellezés formula-kielégítési problémaként (SATként)

A táblán minden szomszédos mezőket összekötő élhez rendelünk egy változót, pl.

- $p_{i,j}$ legyen az $i$. sor $j$. oszlopából jobbra menő él,
- $q_{i,j}$ pedig a felfele menő él,

amilyen $i, j$-kre ilyen élek vannak. Pl. egy $6 \times 5$-ös táblán:

| | | | | |
|---|---|---|---|---|
| $p_{6,1}$ | $p_{6,2}$ | $p_{6,3}$ | $p_{6,4}$ | |
| $q_{5,1}$ | $q_{5,2}$ | $q_{5,3}$ | $q_{5,4}$ | $q_{5,5}$ |
| $p_{5,1}$ | $p_{5,2}$ | $p_{5,3}$ | $p_{5,4}$ | |
| $q_{4,1}$ | $q_{4,2}$ | $q_{4,3}$ | $q_{4,4}$ | $q_{4,5}$ |
| $p_{4,1}$ | $p_{4,2}$ | $p_{4,3}$ | $p_{4,4}$ | |
| $q_{3,1}$ | $q_{3,2}$ | $q_{3,3}$ | $q_{3,4}$ | $q_{3,5}$ |
| $p_{3,1}$ | $p_{3,2}$ | $p_{3,3}$ | $p_{3,4}$ | |
| $q_{2,1}$ | $q_{2,2}$ | $q_{2,3}$ | $q_{2,4}$ | $q_{2,5}$ |
| $p_{2,1}$ | $p_{2,2}$ | $p_{2,3}$ | $p_{2,4}$ | |
| $q_{1,1}$ | $q_{1,2}$ | $q_{1,3}$ | $q_{1,4}$ | $q_{1,5}$ |
| $p_{1,1}$ | $p_{1,2}$ | $p_{1,3}$ | $p_{1,4}$ | |

Az elképzelés: azokat a változókat (éleket) állítsuk 1-re, amiket egy dominó fed le.

$q_{5,1}$ $p_{6,2}$ $p_{6,4}$ $p_{5,3}$ $q_{4,2}$ $q_{4,5}$ $q_{3,1}$ $q_{3,3}$ $q_{3,4}$ $q_{2,2}$ $q_{2,5}$ $p_{2,3}$ $q_{1,1}$ $p_{1,2}$ $p_{1,4}$

# Modellezés SATként

Azt kell megfogalmaznunk, hogy mikor felel meg egy értékadás egy megoldásnak, vagyis egy tatami lefedésnek.

- Minden mezőt lefed legalább egy dominó: vagyoljuk a rá illeszkedő éleket
- Minden mezőt legfeljebb egy dominó fed: az egy mezőre illeszkedő éleket nand kapcsolatba hozzuk: $\neg(x \wedge y)$
- A tatami feltétel: minden „sarok mellett" van igaz változó – ezeket is vagyoljuk

A fenti feltételeket pedig összeéseljük. Részlet:

- $(p_{4,2} \vee q_{4,2} \vee p_{4,1} \vee q_{3,2})$ – a $4$. sor $2$. mezőjét fedi egy dominó;
- $\neg(p_{4,2} \wedge q_{4,2})$ – ezt a mezőt nem fedi egyszerre fentről és jobbról egy dominó;
- $(p_{3,1} \vee q_{2,1} \vee p_{2,1} \vee q_{2,2})$ – a 2. sor 1. oszlop sarkán nem érintkezik négy sarok

Plusz: az előre megadott dominóknak megfelelő változókat $1$-re állítjuk

# Modellezés SATként

- A megadott teszteken ($30 \times 30$-as tábla, kb $30$ ledobott dominó) ez kb. $900$ változó és $7.000$ feltétel
- Ez egy mai SAT solvernek nem méret

unless ha a formula szándékosan „nehéz” SAT példánynak készült

- A formalizálás implementálása, oda-vissza konverzió tatami lefedés és formula közt: félóra, tops

A mai SAT solverek jók és egyre jobbak $\Rightarrow$ mindig jó ötlet elgondolkodni rajta, hogy az aktuális problémánkat formulává tudjuk-e konvertálni hasonló módon

More of this @ bonya

# A modellek halmaza

$\mathrm{Mod}(F)$

Ha $F$ egy formula, akkor $\mathrm{Mod}(F)$ az $F$ összes modelljének a halmaza.

Tehát hogy $\mathcal{A}(F) = 1$, vagy $\mathcal{A} \vDash F$, úgy is írhatjuk, hogy $\mathcal{A} \in \mathrm{Mod}(F)$.

Pl. ha $\mathcal{A}(p) = 1$, $\mathcal{A}(q) = 0$, $\mathcal{A}(r) = 0$, akkor

$$\mathcal{A} \in \mathrm{Mod}\Big((p \to q) \vee (\neg r \leftrightarrow p)\Big).$$

Így pl. $F$ pontosan akkor kielégíthetetlen, ha $\mathrm{Mod}(F) = \emptyset$.

Ha $\Sigma$ formulák egy halmaza és $\mathcal{A}$ egy értékadás, akkor $\mathcal{A} \vDash \Sigma$ azt jelenti, hogy $\mathcal{A}$ kielégíti $\Sigma$ összes elemét.

Hasonlóan $\mathrm{Mod}(\Sigma)$-ba azok az értékadások tartoznak, melyek kielégítik $\Sigma$ összes elemét.

Pl. $\mathrm{Mod}(\emptyset)$-be az összes értékadás beletartozik, mert egyik sem sért meg egy feltételt sem a nullából

# Tautológiák és kielégíthetetlenség

Könnyű látni, hogy

Az $F$ formula pontosan akkor tautológia, ha $\neg F$ kielégíthetetlen.

**Hiszen**

$$\begin{aligned} F \text{ tautológia} &\Leftrightarrow \text{ minden } \mathcal{A}\text{-ra } \mathcal{A}(F) = 1 \\ &\Leftrightarrow \text{ minden } \mathcal{A}\text{-ra } \neg\mathcal{A}(F) = 0 \\ &\Leftrightarrow \text{ minden } \mathcal{A}\text{-ra } \mathcal{A}(\neg F) = 0 \\ &\Leftrightarrow \neg F \text{ kielégíthetetlen.} \end{aligned}$$

A tautológiák persze kielégíthetők.

# Logikai következmény

A $\vDash$ jel egy másik overloadja:

Ha $F$ és $G$ formulák, akkor $F \vDash G$ („$F$-nek logikai következménye $G$”) azt jelöli, hogy minden $\mathcal{A}$-ra ha $\mathcal{A}(F) = 1$, akkor $\mathcal{A}(G) = 1$.

- ha $F$ igaz, akkor $G$ is igaz
- $\mathrm{Mod}(F) \subseteq \mathrm{Mod}(G)$

„$F$ modellje $G$-nek”, „$F$ kielégíti $G$-t” viszont értelmetlen

Például $(p \wedge q) \vee r \ \vDash \ \neg p \to r$:

| $p$ | $q$ | $r$ | $(p \wedge q) \vee r$ | $\neg p \to r$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |

| $p$ | $q$ | $r$ | $(p \wedge q) \vee r$ | $\neg p \to r$ |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

# Logikai következmény formulahalmazok közt

Ugyanígy használhatjuk a $\Sigma \vDash F$, $\Sigma \vDash \Gamma$ jelöléseket is, ahol $\Sigma$, $\Gamma$ formulahalmazok: pl. $\Sigma \vDash F$ akkor áll fenn, ha minden $\Sigma$ minden modellje modellje $F$-nek is.

**Például**

$$\{p, p \to q\} \ \vDash \ q$$

hiszen ha egy $\mathcal{A}$ értékadásban $p$ is és $p \to q$ is igaz, akkor $q$ is igaz.

Általában arra vagyunk kíváncsiak, hogy egy $F$ formula következik-e axiómák egy $\Sigma$ halmazából.

Az $F \equiv G$ („$F$ ekvivalens $G$-vel") jelölés pedig azt jelenti, hogy $\mathrm{Mod}(F) = \mathrm{Mod}(G)$ (tehát $F \vDash G$ és $G \vDash F$).

# $\mathrm{Mod}(\Sigma) \cap \mathrm{Mod}(\Gamma)$

Hasznos tudnunk a következőt:

$$\mathrm{Mod}(\Sigma \cup \Gamma) \;=\; \mathrm{Mod}(\Sigma) \cap \mathrm{Mod}(\Gamma).$$

**Mert**

- a bal oldalon szereplő halmazban azok az értékadások vannak, melyek kielégítik $\Sigma \cup \Gamma$ összes elemét
- azaz $\Sigma$ összes elemét is és $\Gamma$ összes elemét is
- azaz melyek benne vannak $\mathrm{Mod}(\Sigma)$-ban is és $\mathrm{Mod}(\Gamma)$-ban is
- ez pedig épp a jobb oldal

Nyilván az is igaz, hogy

tetszőleges $\mathcal{A}$ értékadás vagy $\mathrm{Mod}(F)$-ben, vagy $\mathrm{Mod}(\neg F)$-ben szerepel (pontosan az egyikükben), ha $F$ egy formula.

# Az indirekt bizonyítás

Ha következtető-motort akarunk fejleszteni, ahhoz elég a kielégíthetetlenséggel foglalkoznunk:

> $\Sigma \vDash F$ pontosan akkor igaz, ha $\Sigma \cup \{\neg F\}$ kielégíthetetlen.

**Bizonyítás: $\Sigma \models F$ pont akkor, ha**

- $\mathrm{Mod}(\Sigma) \subseteq \mathrm{Mod}(F)$ — $\models$ def
- minden $\mathcal{A}$-ra: ha $\mathcal{A} \in \mathrm{Mod}(\Sigma)$, akkor $\mathcal{A} \in \mathrm{Mod}(F)$ — részhalmaz def
- minden $\mathcal{A}$-ra: ha $\mathcal{A} \in \mathrm{Mod}(\Sigma)$, akkor $\mathcal{A}(F) = 1$ — Mod def
- minden $\mathcal{A}$-ra: ha $\mathcal{A} \in \mathrm{Mod}(\Sigma)$, akkor $\neg\mathcal{A}(F) = 0$ — $\neg$ def + $\neg$ injektív
- minden $\mathcal{A}$-ra: ha $\mathcal{A} \in \mathrm{Mod}(\Sigma)$, akkor $\mathcal{A}(\neg F) = 0$ — $\neg$ szemantika def
- minden $\mathcal{A}$-ra: ha $\mathcal{A} \in \mathrm{Mod}(\Sigma)$, akkor $\mathcal{A} \notin \mathrm{Mod}(\neg F)$ — Mod def
- minden $\mathcal{A}$-ra: nem $\Big(\mathcal{A} \in \mathrm{Mod}(\Sigma)$ és $\mathcal{A} \in \mathrm{Mod}(\neg F)\Big)$ — $p \to \neg q \ \equiv\ \neg(p \wedge q)$
- $\mathrm{Mod}(\Sigma) \cap \mathrm{Mod}(\neg F) = \emptyset$ — $x = \emptyset \ \leftrightarrow\ \forall y(y \notin x)$ halmazelméleti axióma
- $\mathrm{Mod}(\Sigma \cup \{\neg F\}) = \emptyset$ — előző „lemma"

# Az indirekt bizonyítás

Ha következtető-motort akarunk fejleszteni, ahhoz elég a kielégíthetetlenséggel foglalkoznunk:

$\Sigma \models F$ pontosan akkor igaz, ha $\Sigma \cup \{\neg F\}$ kielégíthetetlen.

**Bizonyítás: $\Sigma \models F$ pont akkor, ha**

- $\mathrm{Mod}(\Sigma) \subseteq \mathrm{Mod}(F)$ — $\models$ def
- $\mathrm{Mod}(\Sigma) \cap \mathrm{Mod}(\neg F) = \emptyset$ — $x = \emptyset \ \leftrightarrow \ \forall y (y \notin x)$ halmazelméleti axióma
- $\mathrm{Mod}(\Sigma \cup \{\neg F\}) = \emptyset$ — előző „lemma”

# Conjunctive normal form (CNF)

Usually we specify our knowledge inferring algorithms to formulae in some normal form to make the number of differently handled cases low. The most frequently used normal form is the conjunctice normal form:

- The variables and their negated variants are called literals
- A disjunction of finitely many literals is called a clause
- A conjunction of finitely many clauses is called a CNF

**Example**

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

In this formula

- $p, q, r$ are variables,
- $p, \neg q, \neg p, r$ are literals,
- $(p \vee \neg q)$, $(\neg p \vee \neg q \vee r)$ and $p$ are clauses.

Clauses consisting of a single literal (like $p$ above) are called unit clauses.

# Transforming a formula to CNF

Every formula can be transformed to an equivalent CNF.

- First we eliminate the implication and the equivalence connectives by using the following rewrite rules:

  $$F \rightarrow G \equiv \neg F \vee G \qquad F \leftrightarrow G \equiv (\neg F \vee G) \wedge (F \vee \neg G)$$

  ok, also $\uparrow \vee F \equiv \uparrow$, $\downarrow \vee F \equiv F$, $\downarrow \rightarrow F \equiv \uparrow$ etc.

- Then, we push the $\neg$ symbols next to the variables using the deMorgan identities:

  $$\neg(F \vee G) \equiv \neg F \wedge \neg G \qquad \neg(F \wedge G) \equiv \neg F \vee \neg G \qquad \neg\neg F \equiv F$$

  (Now the formula is in negation normal form, NNF.)

- Finally we push the $\vee$ symbols below the $\wedge$ symbols using the distributivity laws:

  $$F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H), \quad (F \wedge G) \vee H \equiv (F \vee H) \wedge (G \vee H)$$

# CNF example

Formula:

$$(p \lor q) \leftrightarrow (q \to r)$$

Arrow elimination:

$$\big(\neg(p \lor q) \lor (q \to r)\big) \land \big((p \lor q) \lor \neg(q \to r)\big)$$

$$\big(\neg(p \lor q) \lor (\neg q \lor r)\big) \land \big((p \lor q) \lor \neg(\neg q \lor r)\big)$$

Pushing negations below:

$$\big((\neg p \land \neg q) \lor (\neg q \lor r)\big) \land \big((p \lor q) \lor (\neg\neg q \land \neg r)\big)$$

$$\big((\neg p \land \neg q) \lor (\neg q \lor r)\big) \land \big((p \lor q) \lor (q \land \neg r)\big)$$

Distributivity

clauses, CNFs

$$(\neg p \lor \neg q \lor r) \land (\neg q \lor \neg q \lor r) \land (p \lor q \lor q) \land (p \lor q \lor \neg r)$$

# Sets

So the problem we want to solve is the following:

**SAT**

**Input:** a CNF.

**Output:** is it satisfiable?

The CNF is not represented by a string, instead

- a clause is represented by the set of the literals it contains,
- and a CNF is represented as the set of its clauses.

We are able to do this since the $\vee$ and $\wedge$ operations commute, are associative and idempotent.

Our previous example in this representation is the set below:

$$\Sigma = \Big\{\{\neg p, \neg q, r\},\ \{\neg q, r\},\ \{p, q\},\ \{p, q, \neg r\}\Big\}.$$

# Empty clause, empty CNF

Even if there is no empty clause in the input, it can be generated by the algorithms. The empty clause is denoted by $\square$.

The empty clause always evaluates to false.

(E.g. because for arbitrary clauses $C$ and $D$ and assignment $\mathcal{A}$ the relation $\mathcal{A}(C \cup D) = \mathcal{A}(C) \vee \mathcal{A}(D)$ should hold. Now if $D = \square$, then it becomes $\mathcal{A}(C) = \mathcal{A}(C \cup \square) = \mathcal{A}(C) \vee \mathcal{A}(\square)$, which can be true only if $\mathcal{A}(\square) = 0$.)

The empty CNF (that is, the conjunction of zero clauses) will be denoted by $\emptyset$.

The empty CNF always evaluates to true.

$\{\}$ true, if it's a CNF and false, if it's a clause.

$\square$ is false. $\emptyset$ is true..

# Restrictions of CNFs

If $\ell$ is a literal, then $\overline{\ell}$ denotes the complement of $\ell$: $\overline{p} := \neg p$ and $\overline{\neg p} := p$.

Similarly to the restrictions of Boolean functions, we can fix the value of a variable in a CNF as well:

If $\Sigma$ is a set of clauses and $\ell$ is a literal, then $\Sigma|_{\ell=1}$ is also a set of clauses, namely:

- first, we drop those clauses from $\Sigma$ containing $\ell$,
- then, we drop the $\overline{\ell}$ literals from the remaining clauses. (But keep the clauses themselves.)

**Example**

If $\Sigma = \{\{p\}, \{p, q\}, \{\neg p, \neg q\}, \{p, \neg p, r\}\}$ and $\ell = \neg p$, then $\Sigma|_{\neg p=1} = \{\Box, \{q\}\}$.

Let $\Sigma|_{\ell=0}$ be $\Sigma|_{\overline{\ell}=1}$.

# Restrictions of CNFs

Let $\mathcal{A}$ be an assignment, $\Sigma$ a CNF and $\ell$ a literal.
$\mathcal{A} \vDash \Sigma$ holds if and only if one of the following holds:

- $\mathcal{A}(\ell) = 1$ and $\mathcal{A} \vDash \Sigma|_{\ell=1}$
- or $\mathcal{A}(\ell) = 0$ and $\mathcal{A} \vDash \Sigma|_{\ell=0}$.

In short: if $\mathcal{A} \vDash \Sigma|_{\ell=\mathcal{A}(\ell)}$.

**Why?**

If $\mathcal{A}(\ell) = 1$, then $\mathcal{A}$ satisfies all those clauses of $\Sigma$ that contain $\ell$.
In the other clauses, $\overline{\ell}$ evaluates to false under $\mathcal{A}$, we can drop that literal.
The clause set we get this way is exactly $\Sigma|_{\ell=1}$.
(The case $\ell = 0$ is already done since that's the same as the case $\overline{\ell} = 1$.)

# The first algorithm

A corollary of the previous statement:

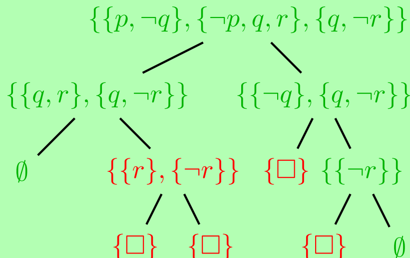
Let $\Sigma$ be a set of clauses and $\ell$ a literal.
The set $\Sigma$ is satisfiable if and only if so is $\Sigma|_{\ell=0}$ or $\Sigma|_{\ell=1}$.

This gives rise to an algorithm:

**function** $\mathrm{A}(\Sigma)$
  **if** $\Box \in \Sigma$ **then return** false
  **if** $\Sigma = \emptyset$ **then return** true
  (let us choose a variable $p$)
  **return** $\mathrm{A}(\Sigma|_{p=0}) \vee \mathrm{A}(\Sigma|_{p=1})$

(It is easy to modify the algorithm to give back a satisfying assignmnent instead of „true".)

**Example**



# Pruning the search space: Unit propagation

If there exists a unit clause in the CNF, that forces the value of its variable:

If $\Sigma$ contains a unit clause $\{\ell\}$, then in each model $\mathcal{A}$ of $\Sigma$ it holds that $\mathcal{A}(\ell) = 1$.

Thus, in this case $\Sigma$ is satisfiable if and only if so is $\Sigma|_{\ell=1}$ (since $\Sigma|_{\ell=0}$ contains an empty clause and is hence unsatisfiable.)

**Unit propagation**

If $\{\ell\} \in \Sigma$, then it suffices to make the recursive call only for $\Sigma|_{\ell=1}$.

# Pruning the search space: Pure literal elimination

If $\ell$ is a literal such that $\Sigma$ does not contain $\overline{\ell}$, then $\Sigma$ is satisfiable if and only if so is $\Sigma|_{\ell=1}$.

Hence: if $\overline{\ell}$ does not occur in $\Sigma$, then we can safely set $\ell$ to $1$.

If $\mathcal{A} \models \Sigma$, then the assignment $\mathcal{A}[\ell = 1]$, which differs from $\mathcal{A}$ only in $\ell$, setting $\ell$ to $1$, and on the other variables it is the same as $\mathcal{A}$, satisfies $\Sigma|_{\ell=1}$:

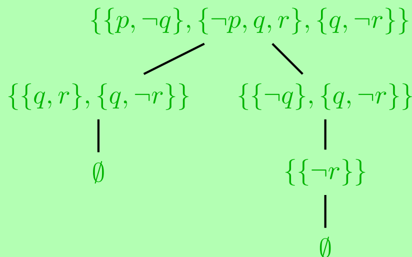
- if $\ell \in C$, then by the setting $[\ell = 1]$
- if $\ell \notin C$, then $\mathcal{A}(C) = \mathcal{A}[\ell = 1](C)$, since in this case the value of $C$ does not depend on the value of $\ell$.

# The DPLL algorithm

Davis–Putnam–Logemann–Loveland

**function** $\textsc{dpll}(\Sigma)$

- **if** $\Sigma = \emptyset$ **then return** true
- **if** $\square \in \Sigma$ **then return** false
- **if** $\{\ell\} \in \Sigma$ holds for some $\ell$ **then**
  - **return** $\textsc{dpll}(\Sigma|_{\ell=1})$
- **if** $\overline{\ell}$ does not occur in $\Sigma$ for some $\ell$
  - **return** $\textsc{dpll}(\Sigma|_{\ell=1})$
- (let us choose a variable $p$)
- **return** $\textsc{dpll}(\Sigma|_{p=0}) \vee \textsc{dpll}(\Sigma|_{p=1})$




Nowadays' fastest SAT solvers implement variants of this base algorithm.