

# SAT solvery

6. prednáška

Logika pre informatikov a Úvod do matematickej logiky

---

Ján Klúka, Ján Mazák, Jozef Šiška

Letný semester 2024/2025

Univerzita Komenského v Bratislave  
Fakulta matematiky, fyziky a informatiky

## Obsah 6. prednášky

---

### SAT solvery

Problém výrokovologickej splniteľnosti (SAT)

Výpočtová zložitosť: teória a prax (*informatívne*)

Algoritmy na riešenie problému splniteľnosti

Backtracking

DPLL a sledované literály

CDCL

Ďalšie aspekty (*informatívne*)

Verifikácia hardvéru (*informatívne*)

Kombinatorické problémy (*informatívne*)

Časti tejto prednášky sa netreba učiť na skúšku, slúžia len na ilustráciu historických či vecných súvislostí a rozšírenie všeobecného prehľadu. Sú označené slovom „*informatívne*“ na slajde alebo v názve podkapitoly.

## SAT solvery

---

# SAT solvery

---

Problém výrokovologickej splniteľnosti  
(SAT)

## Definícia 5.1 (Problém SAT)

*Problémom výrokovologickej splniteľnosti (SAT)* je problém určenia toho, či je daná množina výrokovologických formúl splniteľná.

- Zvyčajne sa redukuje na problém splniteľnosti **klauzálnej** teórie (teda formuly v CNF).
- **SAT solver** je program, ktorý rieši problém SAT.

## Príklad 5.2

Nech  $a, b, c$  sú predikátové atómy.

Nech  $S = \{(a \vee b), (a \vee \neg b), (\neg a \vee b), (\neg a \vee \neg b \vee \neg c), (\neg a \vee c)\}$ .

Je množina klauzúl  $S$  splniteľná?

Súvisiace problémy:

- AllSAT — nájsť všetky ohodnotenia, pre ktoré je formula splnená
- #SAT — zistiť počet ohodnotení, pre ktoré je formula splnená
- MaxSAT — zistiť najväčší možný počet klauzúl formuly v CNF, ktoré je možné splniť súčasne
- weighted MaxSAT — klauzuly majú rôznu váhu a maximalizujeme súčet váh splnených klauzúl
- 3-SAT — klauzuly majú  $\leq 3$  literály (NP-ťažké)
- 2-SAT — klauzuly majú  $\leq 2$  literály (P)

Praktické využitie:

- verifikácia hardvéru (Intel i7)
- verifikácia softvéru (Windows 7 device drivers)
- manažment softvérových závislostí (Eclipse plugins, Python Conda)
- konfigurácia produktov (Daimler)
- bioinformatika, kryptológia
- expertné systémy, letová kontrola, rozvrhovanie, ...



- výrazný pokrok v rokoch 1996–2001, keď sa SAT solvery stali dostatočne rýchle pre praktické využitie
- od r. 2002 každoročne SAT Competition
- o.i. kategória „Glucose hack“ — modifikácia existujúceho solvera nesmie presiahnuť 1000 znakov
- desiatky SAT solverov s otvoreným zdrojovým kódom
- 2013+ SAT *configuration* competition: pre obmedzený okruh vstupov možno dosiahnuť zrýchlenie typicky 2–10× (4,5× pre verifikáciu hardvéru)

# SAT solvery

---

Výpočtová zložitosť: teória a prax  
(*informatívne*)

- **zložitosť algoritmu** — počet krokov výpočtu ako funkcia veľkosti vstupu  $n$  (nezávisí od hardvéru)

- **zložitosť algoritmu** — počet krokov výpočtu ako funkcia veľkosti vstupu  $n$  (nezávisí od hardvéru)
- **zložitosť problému** — zložitosť optimálneho algoritmu riešiaceho daný problém; je známa len veľmi výnimočne, napr. triedenie porovnávaním je  $O(n \log n)$

- **zložitosť algoritmu** — počet krokov výpočtu ako funkcia veľkosti vstupu  $n$  (nezávisí od hardvéru)
- **zložitosť problému** — zložitosť optimálneho algoritmu riešiaceho daný problém; je známa len veľmi výnimočne, napr. triedenie porovnávaním je  $O(n \log n)$
- zložitosť porovnáваме za predpokladu  $n$  idúceho do nekonečna

- od cca 1970 problémy delíme na „ľahké“ (známy polynomiálny algoritmus, trieda P) a „ťažké“ (nik nepozná polynomiálny algoritmus, triedy NP, PSPACE...)
- veľa ťažkých problémov patrí do NP: riešenie je možné overiť v polynomiálnom čase

- od cca 1970 problémy delíme na „ľahké“ (známy polynomiálny algoritmus, trieda P) a „ťažké“ (nik nepozná polynomiálny algoritmus, triedy NP, PSPACE...)
- veľa ťažkých problémov patrí do NP: riešenie je možné overiť v polynomiálnom čase
- napriek rozsiahlemu výskumu vôbec nevieme, či  $P \neq NP$
- niektoré problémy sú nerozhodnuteľné (vieme dokázať, že nemôže existovať algoritmus)

- $2^n$  je lepšie ako  $n^{100}$   
(ale ak sme na niečo našli polynomiálny algoritmus, zväčša sme do pár rokov našli aj prakticky použiteľný polyn. algoritmus)



- $2^n$  je lepšie ako  $n^{100}$   
(ale ak sme na niečo našli polynomiálny algoritmus, zväčša sme do pár rokov našli aj prakticky použiteľný polyn. algoritmus)
- asymptotické porovnávanie ignoruje konštanty, a tie sú niekedy podstatné (napr. v quicksorte sa nepoužíva lineárny algoritmus na hľadanie mediánu)

- $2^n$  je lepšie ako  $n^{100}$   
(ale ak sme na niečo našli polynomiálny algoritmus, zväčša sme do pár rokov našli aj prakticky použiteľný polyn. algoritmus)
- asymptotické porovnávanie ignoruje konštanty, a tie sú niekedy podstatné (napr. v quicksorte sa nepoužíva lineárny algoritmus na hľadanie mediánu)
- teoreticky najlepšie algoritmy neraz nie sú implementované — sú výhodné len pre obrovské vstupy (ktoré sa možno ani nezmestia do pamäte)

- $2^n$  je lepšie ako  $n^{100}$   
(ale ak sme na niečo našli polynomiálny algoritmus, zväčša sme do pár rokov našli aj prakticky použiteľný polyn. algoritmus)
- asymptotické porovnávanie ignoruje konštanty, a tie sú niekedy podstatné (napr. v quicksorte sa nepoužíva lineárny algoritmus na hľadanie mediánu)
- teoreticky najlepšie algoritmy neraz nie sú implementované — sú výhodné len pre obrovské vstupy (ktoré sa možno ani nezmestia do pamäte)
- hardvér je neraz dôležitejší ako algoritmus (hodinky dnes majú viac výkonu ako niekdajšie superpočítače)
- niekedy je podstatný špecializovaný hardvér (napr. Bitcoin mining, AI)

- strojový čas je lacnejší ako ľudský; komplexita alg. prináša chyby

- strojový čas je lacnejší ako ľudský; komplexita alg. prináša chyby
- niekedy využívame pravdepodobnostné algoritmy, ktoré napr. negarantujú čas behu v najhoršom prípade, ale „takmer vždy“ sú rýchle

- strojový čas je lacnejší ako ľudský; komplexita alg. prináša chyby
- niekedy využívame pravdepodobnostné algoritmy, ktoré napr. negarantujú čas behu v najhoršom prípade, ale „takmer vždy“ sú rýchle
- NP-úplné problémy sú teoreticky ekvivalentné, ale v praxi výrazne odlišné (edge colouring vs. circular edge colouring)

- strojový čas je lacnejší ako ľudský; komplexita alg. prináša chyby
- niekedy využívame pravdepodobnostné algoritmy, ktoré napr. negarantujú čas behu v najhoršom prípade, ale „takmer vždy“ sú rýchle
- NP-úplné problémy sú teoreticky ekvivalentné, ale v praxi výrazne odlišné (edge colouring vs. circular edge colouring)
- algoritmy s veľkou zložitosťou občas fungujú prekvapivo dobre, najmä ak sú doplnené efektívnymi heuristikami

- strojový čas je lacnejší ako ľudský; komplexita alg. prináša chyby
- niekedy využívame pravdepodobnostné algoritmy, ktoré napr. negarantujú čas behu v najhoršom prípade, ale „takmer vždy“ sú rýchle
- NP-úplné problémy sú teoreticky ekvivalentné, ale v praxi výrazne odlišné (edge colouring vs. circular edge colouring)
- algoritmy s veľkou zložitou občas fungujú prekvapivo dobre, najmä ak sú doplnené efektívnymi heuristikami
- klasická teória zložitosti nezohľadňuje nerovnomernú distribúciu vstupov vyskytujúcich sa v praxi



- strojový čas je lacnejší ako ľudský; komplexita alg. prináša chyby
- niekedy využívame pravdepodobnostné algoritmy, ktoré napr. negarantujú čas behu v najhoršom prípade, ale „takmer vždy“ sú rýchle
- NP-úplné problémy sú teoreticky ekvivalentné, ale v praxi výrazne odlišné (edge colouring vs. circular edge colouring)
- algoritmy s veľkou zložitou občas fungujú prekvapivo dobre, najmä ak sú doplnené efektívnymi heuristikami
- klasická teória zložitosti nezohľadňuje nerovnomernú distribúciu vstupov vyskytujúcich sa v praxi
- pre problém splniteľnosti sú praktické vstupy aj 10–100× väčšie, než naznačuje teória

# Problém SAT

---

- prvý problém s dokázanou NP-úplnosťou
- teoretická zložitosť najlepších algoritmov cca  $1.3^n$  v najhoršom prípade
- ale v praxi riešiteľný pre tisíce až milióny premenných/atómov

# SAT solvery

---

Algoritmy na riešenie problému  
splniteľnosti



- hrubá sila (tabuľka všetkých ohodnotení)
- backtracking
- DPLL [1960]
- CDCL (conflict-driven clause learning) [1996]
- watched literals [2001]
- VSIDS heuristic [2001]
- VSIDS combined with machine learning [Maple 2016+]

# Tabuľková metóda

---

Tabuľková metóda:

- Skúma všetky ohodnotenia predikátových atómov
- Trvá  $O(s \cdot 2^n)$  krokov,
  - $n$  je počet atómov a  $s$  je súčet veľkostí klauzúl
  - $2^n$  ohodnotení, pre každé treba zistiť, či sú všetky klauzuly pravdivé
- Zaberá priestor  $O(k \cdot 2^n)$ 
  - $k$  je počet klauzúl
  - Pamätáme si (píšeme na papier) celú tabuľku
- Tabuľka slúži **aj** ako dôkaz prípadnej **nesplniteľnosti**
  - kratší dôkaz nesplniteľnosti než kompletný záznam činnosti algoritmu riešiaceho SAT zatiaľ nemáme
  - ak by existoval dôkaz s polynomiálnou dĺžkou, bolo by  $NP = coNP$

# SAT solvery

---

## Backtracking

# Naivný backtracking v Pythone

```
#!/usr/bin/env python3
```

```
class Sat(object):
```

```
    def __init__(self, n, clauses):
```

```
        self.n, self.clauses, self.solution = n, clauses, None
```

```
    def checkClause(self, v, c):
```

```
        return any( ( v[abs(lit)] if lit > 0 else not v[abs(lit)] )
```

```
                    for lit in c )
```

```
    def check(self, v):
```

```
        return all( self.checkClause(v, cl) for cl in self.clauses )
```

```
    def solve(self, i, v):
```

```
        if i >= self.n: # ohodnotili sme vsetky atomy
```

```
            if self.check(v):
```

```
                self.solution = v
```

```
                return True
```

```
            return False
```

```
        for b in [True, False]:
```

```
            v[i] = b
```

```
            if self.solve(i+1, v):
```

```
                return True
```

```
        return False
```

```
Sat(20, [[]]).solve(0, {})
```

Čas:  $O(s \cdot 2^n)$ , priestor:  $O(s+n)$ ;

$n$  — počet atómov,

$s$  — súčet veľkostí klauzúl

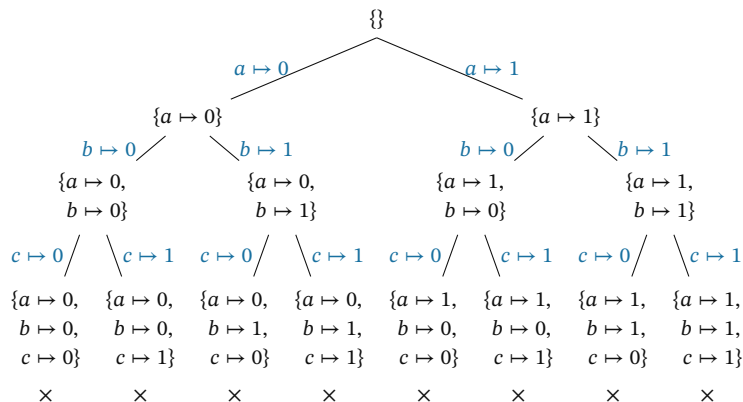


# Strom prehľadávania ohodnotení

$$S = \{(a \vee b), (a \vee \neg b), (\neg a \vee b), (\neg a \vee \neg b \vee \neg c), (\neg a \vee c)\}$$

$\times$  znamená  $\not\models_p S$

$f := 0, t := 1$



# Priebežné vyhodnocovanie klauzúl

Strom ohodnotení:

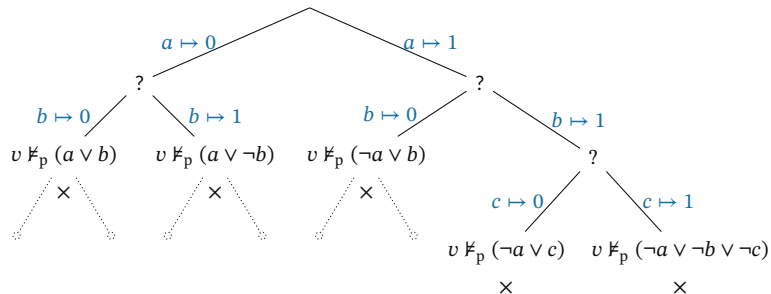
- List — ohodnotenie všetkých premenných
- Každý uzol — *čiastočné ohodnotenie*
- Ohodnotenie v uzle je *rozšírením* ohodnotenia v rodičovi
- Niektoré klauzuly sa dajú vyhodnotiť aj v čiastočnom ohodnotení
  - V čiastočnom ohodnotení  $v = \{a \mapsto 0, b \mapsto 1\}$   
sa dá určiť pravdivosť  $(a \vee b)$ ,  $(a \vee \neg b)$ ,  $(\neg a \vee b)$  z našej  $S$
- Ak nájdeme nepravdivú, môžeme hneď „backtracknúť“ — zastaviť prehľadávanie vetvy a vrátiť sa o úroveň vyššie
  - V čiastočnom ohodnotení  $v = \{a \mapsto 0, b \mapsto 0\}$   
je nepravdivá  $(a \vee b)$  z  $S$

# Prehľadávanie s priebežným vyhodnocovaním

$$S = \{(a \vee b), (a \vee \neg b), (\neg a \vee b), (\neg a \vee \neg b \vee \neg c), (\neg a \vee c)\}$$

$\times$  znamená  $v \not\models_p S$

? znamená zatiaľ žiadna nepravdivá klauzula



## Zjednodušenie množiny klauzúl podľa literálu

Nech  $v$  je čiastočné ohodnotenie, v ktorom  $v(a) = 1$ .

V každom rozšírení ohodnotenia  $v$ :

- sú pravdivé klauzuly obsahujúce  $a$ 
  - $\{a \mapsto 1, \dots\} \models_p (a \vee b)$
  - $\{a \mapsto 1, \dots\} \models_p (a \vee \neg b)$
- je pravdivá klauzula  $(\ell_1 \vee \dots \vee \neg a \vee \dots \vee \ell_n)$  obsahujúca  $\neg a$   
**vtt** je pravdivá **zjednodušená** klauzula  $(\ell_1 \vee \dots \vee \dots \vee \ell_n)$ 
  - $\{a \mapsto 1, \dots\} \models_p (\neg a \vee \neg b \vee \neg c)$  vtt  $\{a \mapsto 1, \dots\} \models_p (\neg b \vee \neg c)$

Takže množinu  $S$  môžeme **zjednodušiť**:

- klauzuly s  $a$  môžeme **vynechať**;
- klauzuly s  $\neg a$  môžeme **zjednodušiť**.

## Zjednodušenie množiny klauzúl podľa literálu

Množinu klauzúl

$$S = \{(a \vee b), (a \vee \neg b), (\neg a \vee b), (\neg a \vee \neg b \vee \neg c), (\neg a \vee c)\}$$

môžeme **zjednodušiť podľa  $a \mapsto 1$**  na

$$S|_{a \mapsto 1} = \{ \quad \quad \quad b, \quad \quad (\neg b \vee \neg c), \quad \quad c \quad \quad \}.$$

Analogicky môžeme  $S$  zjednodušiť podľa  $a \mapsto 0$  na

$$S|_{a \mapsto 0} = \{ \quad b, \quad \quad \neg b \quad \quad \}.$$

## Zjednodušenie množiny klauzúl podľa literálu

### Definícia 5.3

Nech  $P$  je predikátový atóm,  $S$  je množina klauzúl,  $(t, f)$  je dvojica pravdivostných hodnôt. Potom definujeme

$$S|_P \mapsto f = \{(\ell_1 \vee \dots \vee \ell_n) \mid (\ell_1 \vee \dots \vee P \vee \dots \vee \ell_n) \in S\} \\ \cup \{C \mid C \in S, \text{ v } C \text{ sa nevyskytuje } P \text{ ani } \neg P\}$$

$$S|_P \mapsto t = \{(\ell_1 \vee \dots \vee \ell_n) \mid (\ell_1 \vee \dots \vee \neg P \vee \dots \vee \ell_n) \in S\} \\ \cup \{C \mid C \in S, \text{ v } C \text{ sa nevyskytuje } P \text{ ani } \neg P\}$$

$$S|_{\neg P} \mapsto t = S|_P \mapsto f$$

$$S|_{\neg P} \mapsto f = S|_P \mapsto t$$

### Tvrdenie 5.4

Nech  $P$  je predikátový atóm,  $S$  je množina klauzúl,  $(t, f)$  dvojica pravdivostných hodnôt. Nech  $b \in \{t, f\}$  a  $v$  je ohodnotenie také, že  $v(P) = b$ . Potom  $v \models_p S$  vtt  $v \models_p S|_P \mapsto b$ .

## Propagácia jednotkových klauzúl

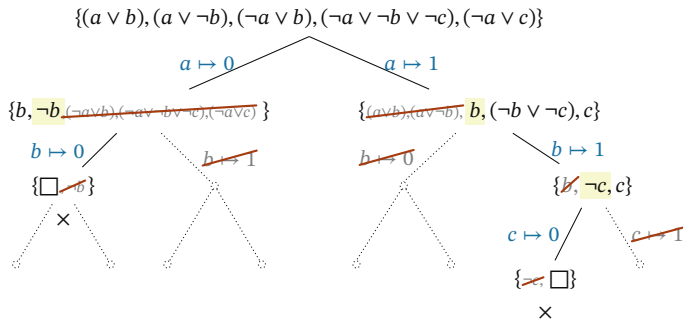
Nech  $T = \{(a \vee \neg b), (a \vee b \vee c)\}$ .

Začnime zjednodušením podľa  $a \mapsto 0$ :

- $T' := T|_{a \mapsto 0} = \{\neg b, (b \vee c)\}$ 
  - $\neg b$  – **jednotková klauzula** (unit clause alebo iba **unit**)
  - $T'$  spĺňajú iba ohodnotenia  $v$ , kde  $v(b) = 0$
  - Takže  $T'$  zjednodušíme podľa  $b \mapsto 0$
- $T'' := T'|_{b \mapsto 0} = \{c\}$ 
  - $c$  – jednotková klauzula
  - $T''$  spĺňajú iba ohodnotenia  $v$ , kde  $v(c) = 1$
  - Takže  $T''$  zjednodušíme podľa  $c$
- $T''' := T''|_{c \mapsto 1} = \{\}$  prázdna, pravdivá v hocijakom ohodnotení. Podľa tvrdenia 5.4:
  - $T''$  je pravdivá v každom ohodnotení, kde  $v(c) = 1$ .
  - $T'$  je pravdivá v každom ohodnotení, kde  $v(b) = 0, v(c) = 1$ .
  - $T$  je pravdivá v ohodnotení  $v = \{a \mapsto 0, b \mapsto 0, c \mapsto 1\}$ .

# Prehľadávanie so zjednodušovaním klauzúl a unit propagation

**Propagácia jednotkových klauzúl (unit propagation)** je proces opakovaného rozširovania ohodnotení podľa jednotkových klauzúl a zjednodušovania.





## Eliminácia nezmiešaných literálov

Všimnime si literál  $P$  v množine klauzúl:

$$T = \{(\neg a \vee \neg b \vee c), (\neg a \vee P), (\neg b \vee P), a, b, \neg c\}$$

Literál  $P$  je **nezmiešaný** (angl. *pure*) v  $T$ :

$P$  sa vyskytuje v  $T$ , ale jeho komplement  $\neg P$  sa tam nevyskytuje.

Nech  $T' := T|_{P \mapsto 1} = \{(\neg a \vee \neg b \vee c), a, b, \neg c\}$

- Ak nájdeme ohodnotenie  $v \models_p T'$ ,  
tak  $v_0 := v[P \mapsto 0]$  aj  $v_1 := v[P \mapsto 1]$  sú modelmi  $T'$   
a  $v_1$  je navyše modelom  $T$ , teda  $T$  je splniteľná.
- Ak je  $T'$  nespĺniteľná,  
tak je nespĺniteľná každá jej nadmnožina, teda aj  $T$ .

Z hľadiska splniteľnosti sú klauzuly obsahujúce  $P$  nepodstatné.

Stačí uvažovať  $T|_{P \mapsto 1}$ .

## Definícia 5.5

Nech  $P$  je predikátový atóm.

**Komplementom literálu  $P$**  je  $\neg P$ . **Komplementom literálu  $\neg P$**  je  $P$ .

Komplement literálu  $\ell$  označujeme  $\bar{\ell}$ .

## Definícia 5.6

Nech  $\ell$  je literál a  $S$  je množina klauzúl.

Literál  $\ell$  je **nezmiešaný** (pure) v  $S$  vtt  $\ell$  sa vyskytuje v niektorej klauzule z  $S$ , ale jeho komplement  $\bar{\ell}$  sa nevyskytuje v žiadnej klauzule z  $S$ .

## Tvrdenie 5.7

Nech  $\ell$  je literál a  $S$  je množina klauzúl.

Ak  $\ell$  je nezmiešaný v  $S$ , tak  $S$  je splniteľná vtt  $S|_{\ell} \mapsto 1$  je splniteľná.

# SAT solvery

---

DPLL a sledované literály

**Algoritmus 5.8 (Davis and Putnam [1960], Davis et al. [1962])**

```
1: def DPLL( $\Phi, v$ ):
2:   if  $\Phi$  obsahuje prázdnu klauzulu:
3:     return False
4:   if  $v$  ohodnocuje všetky atómy:
5:     return True
6:   while existuje jednotková (unit) klauzula  $\ell$  vo  $\Phi$ :
7:      $\Phi, v = \text{unit-propagate}(\ell, \Phi, v)$ 
8:   while existuje nezmiešaný (pure) literál  $\ell$  vo  $\Phi$ :
9:      $\Phi, v = \text{pure-literal-assign}(\ell, \Phi, v)$ 
10:   $x = \text{choose-branch-atom}(\Phi, v)$ 
11:  return DPLL( $\Phi|_x \mapsto t, v(x \mapsto t)$ ) or DPLL( $\Phi|_x \mapsto f, v(x \mapsto f)$ )
```

## Technika sledovaných literálov (watched literals)

Aby sme nemuseli zjednodušovať množinu klauzúl:

- Pre každú klauzulu vyberieme 2 **sledované literály**.

$$(\neg a^{\odot} \vee \neg b^{\odot} \vee \neg c)$$

- Sledovaný literál musí byť **nenastavený** alebo **true**, ak sa to dá.
- Ak sa sledovaný literál stane *true*: nič nemusíme robiť.

$$\{a \mapsto 0\} \quad (\neg a^{\odot} \vee \neg b^{\odot} \vee \neg c)$$

- Ak sa sledovaný literál stane *false*: musíme nájsť iný.

$$\{a \mapsto 1\} \quad (\neg a^{\odot} \vee \neg b^{\odot} \vee \neg c^{\odot})$$

Ak iný nie je, práve sme vyrobili jednotkovú klauzulu (všetky literály okrem druhého sledovaného sú *false*),

$$\{a \mapsto 1, b \mapsto 1\} \quad (\neg a \vee \neg b^{\odot} \vee \neg c_u^{\odot})$$

alebo spor (aj druhý sledovaný je už *false*).

$$\{a \mapsto 1, b \mapsto 1, c \mapsto 0\} \quad (\neg a^{\odot} \vee c^{\odot})$$

- Keď backtrackujeme: nič nemusíme robiť (možno sa niektoré sledované literály stanú *nenastavenými*).

## Technika sledovaných literálov (watched literals)

---

- netreba v každom kroku prepisovať skúmanú formulu
- pri unit propagation máme priamo odkaz na relevantné klauzuly, nemusíme prepisovať všetky ani hľadať ich vo formule
- žiadna práca pri kroku naspäť
- pre 3-SAT sa ušetrí len málo, preto preferovaná veľkosť klauzúl je výrazne viac ako 3 (dosiahne sa predspracovaním vstupu)
- nezlepšuje asymptotickú zložitosť, ale veľmi užitočné v praxi

# Prehľadávanie s unit propagation a sledovaním

$$\begin{aligned} &\{ (a^\oplus \vee b^\oplus), (a^\oplus \vee \neg b^\oplus), \\ &(\neg a^\oplus \vee b^\oplus), (\neg a^\oplus \vee c^\oplus), \\ &(\neg a^\oplus \vee \neg b^\oplus \vee \neg c^\oplus) \} \end{aligned}$$

$$\begin{array}{c} a \mapsto 0 \qquad a \mapsto 1 \end{array}$$

$$\begin{aligned} &\{ (a_\perp^\oplus \vee b_u^\oplus), (a_\perp^\oplus \vee \neg b_u^\oplus), \\ &(\neg a^\oplus \vee b^\oplus), (\neg a^\oplus \vee c^\oplus), \\ &(\neg a^\oplus \vee \neg b^\oplus \vee \neg c^\oplus) \} \end{aligned}$$

$$\begin{aligned} &\{ (a^\oplus \vee b^\oplus), (a^\oplus \vee \neg b^\oplus), \\ &(\neg a_\perp^\oplus \vee b_u^\oplus), (\neg a_\perp^\oplus \vee c_u^\oplus), \\ &(\neg a^\oplus \vee \neg b^\oplus \vee \neg c^\oplus) \} \end{aligned}$$

$$\begin{aligned} &b \mapsto 0 / \\ &\{ (a_\perp^\oplus \vee b_\perp^\oplus), (a_\perp^\oplus \vee \neg b^\oplus), \\ &(\neg a^\oplus \vee b_\perp^\oplus), (\neg a^\oplus \vee c^\oplus), \\ &(\neg a^\oplus \vee \neg b^\oplus \vee \neg c^\oplus) \} \end{aligned}$$

$\times$

$$b \mapsto 1$$

$$b \mapsto 0$$

$$\backslash b \mapsto 1$$

$$\begin{aligned} &\{ (a^\oplus \vee b^\oplus), (a^\oplus \vee \neg b^\oplus), \\ &(\neg a_\perp^\oplus \vee b^\oplus), (\neg a_\perp^\oplus \vee c_u^\oplus), \\ &(\neg a \vee \neg b_\perp^\oplus \vee \neg c_u^\oplus) \} \end{aligned}$$

$$c \mapsto 0$$

$$\backslash c \mapsto 1$$

$$\begin{aligned} &\{ (a^\oplus \vee b^\oplus), (a^\oplus \vee \neg b^\oplus), \\ &(\neg a_\perp^\oplus \vee b^\oplus), (\neg a_\perp^\oplus \vee c^\oplus), \\ &(\neg a \vee \neg b_\perp^\oplus \vee \neg c_\perp^\oplus) \} \end{aligned}$$

$\times$

**SAT solvers**

---

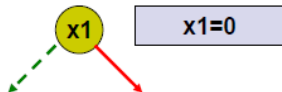
**CDCL**



# CDCL — conflict-driven clause learning

## Step 1

$x_1 + x_4$   
 $x_1 + x_3' + x_8'$   
 $x_1 + x_8 + x_{12}$   
 $x_2 + x_{11}$   
 $x_7' + x_3' + x_9$   
 $x_7' + x_8 + x_9'$   
 $x_7 + x_8 + x_{10}'$   
 $x_7 + x_{10} + x_{12}'$

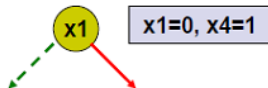
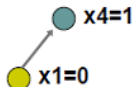


  $x_1=0$

(žltá — rozhodnutie, šedá — unit propagation)

## Step 2

$x_1 + x_4$   
 $x_1 + x_3' + x_8'$   
 $x_1 + x_8 + x_{12}$   
 $x_2 + x_{11}$   
 $x_7' + x_3' + x_9$   
 $x_7' + x_8 + x_9'$   
 $x_7 + x_8 + x_{10}'$   
 $x_7 + x_{10} + x_{12}'$

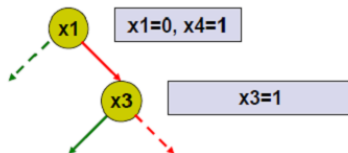
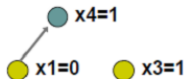


(žltá — rozhodnutie, šedá — unit propagation)

# CDCL — conflict-driven clause learning

## Step 3

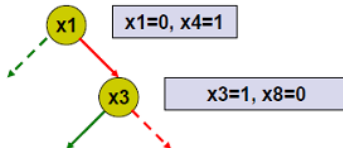
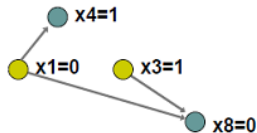
$x1 + x4$   
 $x1 + x3' + x8'$   
 $x1 + x8 + x12$   
 $x2 + x11$   
 $x7' + x3' + x9$   
 $x7' + x8 + x9'$   
 $x7 + x8 + x10'$   
 $x7 + x10 + x12'$



(žltá — rozhodnutie, šedá — unit propagation)

## Step 4

$x1 + x4$   
 $x1 + x3' + x8'$   
 $x1 + x8 + x12$   
 $x2 + x11$   
 $x7' + x3' + x9$   
 $x7' + x8 + x9'$   
 $x7 + x8 + x10'$   
 $x7 + x10 + x12'$

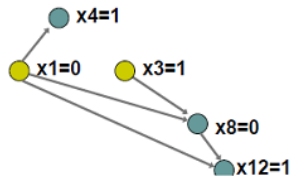
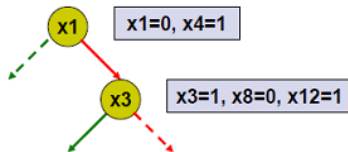


(žltá — rozhodnutie, šedá — unit propagation)

# CDCL — conflict-driven clause learning

Step 5

$x1 + x4$   
 $x1 + x3' + x8'$   
 $x1 + x8 + x12$   
 $x2 + x11$   
 $x7' + x3' + x9$   
 $x7' + x8 + x9'$   
 $x7 + x8 + x10'$   
 $x7 + x10 + x12'$

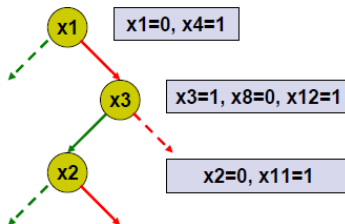
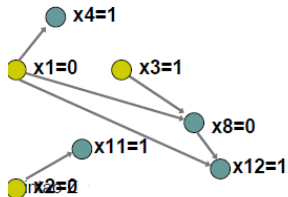


(žltá — rozhodnutie, šedá — unit propagation)

# CDCL — conflict-driven clause learning

Step 7

$x_1 + x_4$   
 $x_1 + x_3' + x_8'$   
 $x_1 + x_8 + x_{12}$   
 $x_2 + x_{11}$   
 $x_7' + x_3' + x_9$   
 $x_7' + x_8 + x_9'$   
 $x_7 + x_8 + x_{10}'$   
 $x_7 + x_{10} + x_{12}'$

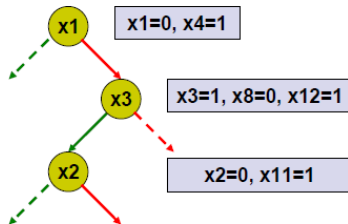
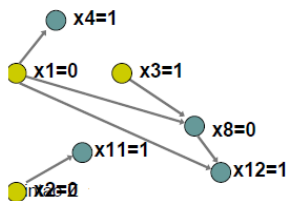


(žltá — rozhodnutie, šedá — unit propagation)

# CDCL — conflict-driven clause learning

Step 8

$x_1 + x_4$   
 $x_1 + x_3' + x_8'$   
 $x_1 + x_8 + x_{12}$   
 $x_2 + x_{11}$   
 $x_7' + x_3' + x_9$   
 $x_7' + x_8 + x_9'$   
 $x_7 + x_8 + x_{10}'$   
 $x_7 + x_{10} + x_{12}'$

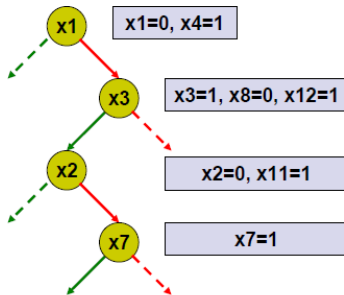
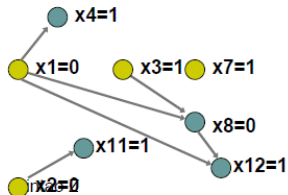


(žltá — rozhodnutie, šedá — unit propagation)

# CDCL — conflict-driven clause learning

Step 9

$x_1 + x_4$   
 $x_1 + x_3' + x_8'$   
 $x_1 + x_8 + x_{12}$   
 $x_2 + x_{11}$   
 $x_7' + x_3' + x_9$   
 $x_7' + x_8 + x_9'$   
 $x_7 + x_8 + x_{10}'$   
 $x_7 + x_{10} + x_{12}'$



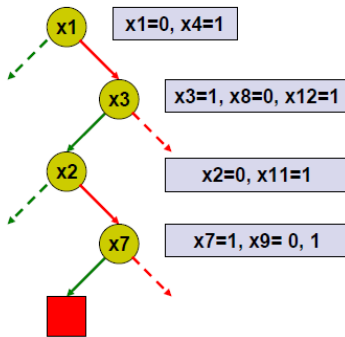
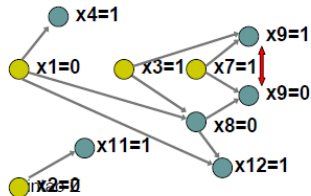
(žltá — rozhodnutie, šedá — unit propagation)



# CDCL — conflict-driven clause learning

Step 10

$x1 + x4$   
 $x1 + x3' + x8'$   
 $x1 + x8 + x12$   
 $x2 + x11$   
 $x7' + x3' + x9$   
 $x7' + x8 + x9'$   
 $x7 + x8 + x10'$   
 $x7 + x10 + x12'$

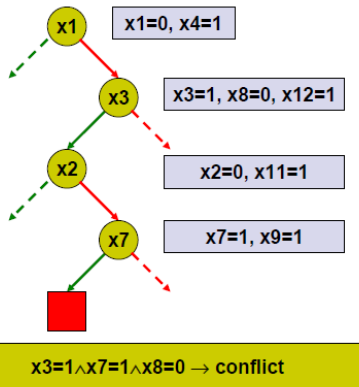
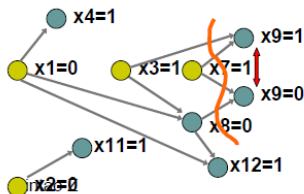


(žltá — rozhodnutie, šedá — unit propagation)

# CDCL — conflict-driven clause learning

$x_1 + x_4$   
 $x_1 + x_3' + x_8'$   
 $x_1 + x_8 + x_{12}$   
 $x_2 + x_{11}$   
 $x_7' + x_3' + x_9$   
 $x_7' + x_8 + x_9'$   
 $x_7 + x_8 + x_{10}'$   
 $x_7 + x_{10} + x_{12}'$

Step 11

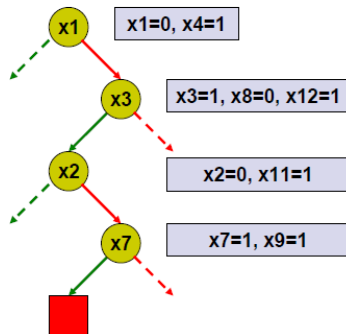
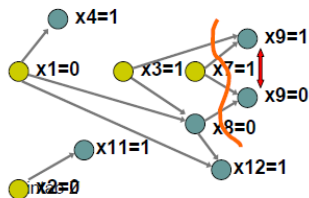


Uvedený rez nie je jediný, mohli by sme pridať  $x_1 \vee \neg x_3 \vee \neg x_7$ .

# CDCL — conflict-driven clause learning

Step 13

$x_1 + x_4$   
 $x_1 + x_3' + x_8'$   
 $x_1 + x_8 + x_{12}$   
 $x_2 + x_{11}$   
 $x_7' + x_3' + x_9$   
 $x_7' + x_8 + x_9'$   
 $x_7 + x_8 + x_{10}'$   
 $x_7 + x_{10} + x_{12}'$



$x_3=1 \wedge x_7=1 \wedge x_8=0 \rightarrow \text{conflict}$

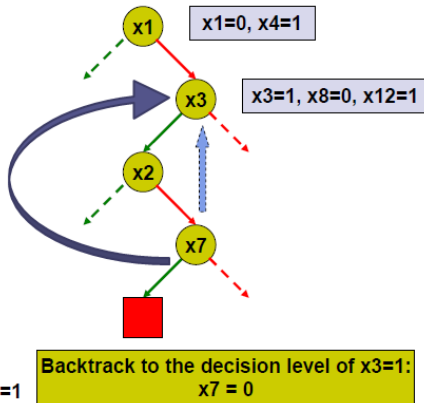
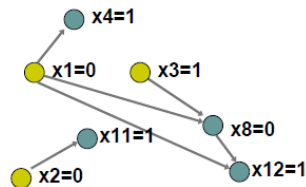
Add conflict clause:  $x_3' + x_7' + x_8$

Uvedený rez nie je jediný, mohli by sme pridať  $x_1 \vee \neg x_3 \vee \neg x_7$ .

# CDCL — conflict-driven clause learning

$x_1 + x_4$   
 $x_1 + x_3' + x_8'$   
 $x_1 + x_8 + x_{12}$   
 $x_2 + x_{11}$   
 $x_7' + x_3' + x_9$   
 $x_7' + x_8 + x_9'$   
 $x_7 + x_8 + x_{10}'$   
 $x_7 + x_{10} + x_{12}'$   
 **$x_3' + x_8 + x_7'$**

Step 14

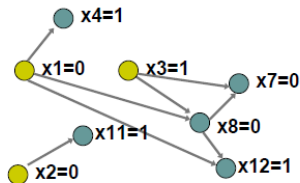
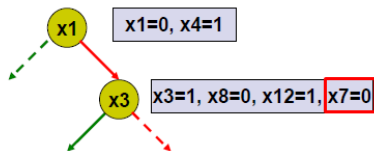


Návrat do bodu, kde pridaná klauzula vynúti ohodnotenie jednej doteraz neohodnotenej premennej (čo zabráni vzniku tohto konfliktu kdekoľvek v podstrome).

# CDCL — conflict-driven clause learning

$x_1 + x_4$   
 $x_1 + x_3' + x_8'$   
 $x_1 + x_8 + x_{12}$   
 $x_2 + x_{11}$   
 $x_7' + x_3' + x_9$   
 $x_7' + x_8 + x_9'$   
 $x_7 + x_8 + x_{10}'$   
 $x_7 + x_{10} + x_{12}'$   
 $x_3' + x_8 + x_7'$

Step 15



- vytvárame implikačný graf

- vytvárame implikačný graf
- keď nájdeme konflikt, zvolíme rez oddelujúci rozhodnutia od konfliktu a odvodíme novú klauzulu, ktorá konfliktu predchádza (*learning*); ak je takých rezov viac, heuristikou niektorý vyberieme

- vytvárame implikačný graf
- keď nájdeme konflikt, zvolíme rez oddelujúci rozhodnutia od konfliktu a odvodíme novú klauzulu, ktorá konfliktu predchádza (*learning*); ak je takých rezov viac, heuristikou niektorý vyberieme
- vrátime sa k predposlednému z rozhodnutí, ktoré viedli ku konfliktu (nie chronologicky — preskočíme rozhodnutia o literáloch nesúvisiacich s konfliktom)



Problémy (viac v [Zhang]):

- exponenciálne veľa klauzúl, ktoré takto možno odvodiť; ktoré si pamätať a ako dlho?  
riešenie: rôzne heuristiky, aktívna oblasť výskumu (Kruger et al. [2022])

Problémy (viac v [Zhang]):

- exponenciálne veľa klauzúl, ktoré takto možno odvodiť; ktoré si pamätať a ako dlho?  
riešenie: rôzne heuristiky, aktívna oblasť výskumu (Kruger et al. [2022])
- čas výpočtu má distribúciu s ťažkým chvostom (fat-tailed — pre niektoré postupnosti rozhodnutí trvá výpočet výrazne dlhšie ako pre iné)  
riešenie: občasný reštart backtrackingu (napr. „Luby restarts“, založené na štatistickej analýze náhodných procesov)

## Ako vybrať nasledujúci literál? (*informatívne*)

---

- voľba literálu pre ďalšie rozhodnutie má výrazný efekt na čas výpočtu

## Ako vybrať nasledujúci literál? (*informatívne*)

---

- voľba literálu pre ďalšie rozhodnutie má výrazný efekt na čas výpočtu
- heuristika VSIDS: „additive bumping, multiplicative decay“
- pre každý literál počítame počet jeho výskytov v odvodených klauzulách (t.j. konfliktoch)
- periodicky toto skóre predelíme konštantou (zdôrazníme tak nedávno naučené klauzuly)

## Ako vybrať nasledujúci literál? (*informatívne*)

---

- voľba literálu pre ďalšie rozhodnutie má výrazný efekt na čas výpočtu
- heuristika VSIDS: „additive bumping, multiplicative decay“
- pre každý literál počítame počet jeho výskytov v odvodených klauzulách (t.j. konfliktoch)
- periodicky toto skóre predelíme konštantou (zdôrazníme tak nedávno naučené klauzuly)
- prekvapivo efektívne, využíva sa vo mnohých súčasných solveroch

## Ako vybrať nasledujúci literál? (*informatívne*)

---

- voľba literálu pre ďalšie rozhodnutie má výrazný efekt na čas výpočtu
- heuristika VSIDS: „additive bumping, multiplicative decay“
- pre každý literál počítame počet jeho výskytov v odvodených klauzulách (t.j. konfliktoch)
- periodicky toto skóre predelíme konštantou (zdôrazníme tak nedávno naučené klauzuly)
- prekvapivo efektívne, využíva sa vo mnohých súčasných solveroch
- heuristika LRB [Maple 2016]: reinforcement learning (multi-armed bandit problem)
- pravidelné prepínanie medzi VSIDS a LRB

## SAT solvery

---

Ďalšie aspekty (*informatívne*)

- všetky moderné SAT solvery venujú značnú pozornosť predspracovaniu formuly
- počet premenných je zvyčajne podstatnejší ako veľkosť formuly



- všetky moderné SAT solvery venujú značnú pozornosť predspracovaniu formuly
- počet premenných je zvyčajne podstatnejší ako veľkosť formuly
- rezolvenciou možno znížiť počet klauzúl (ale narastie ich veľkosť)

- všetky moderné SAT solvery venujú značnú pozornosť predspracovaniu formuly
- počet premenných je zvyčajne podstatnejší ako veľkosť formuly
- rezolvenciou možno znížiť počet klauzúl (ale narastie ich veľkosť)
- rezolvenciou možno znížiť počet premenných (ale výrazne narastie počet klauzúl)

- všetky moderné SAT solvery venujú značnú pozornosť predspracovaniu formuly
- počet premenných je zvyčajne podstatnejší ako veľkosť formuly
- rezolvenciou možno znížiť počet klauzúl (ale narastie ich veľkosť)
- rezolvenciou možno znížiť počet premenných (ale výrazne narastie počet klauzúl)
- poradie klauzúl nemá zásadný vplyv na dĺžku výpočtu
- redundantné klauzuly môžu pomôcť

- desiatky rôznych techník, často doménovo špecifických
- napr. neúplné BDD reprezentácie umožňujú získať klauzuly, ktoré nemožno odvodiť počas CDCL

- desiatky rôznych techník, často doménovo špecifických
- napr. neúplné BDD reprezentácie umožňujú získať klauzuly, ktoré nemožno odvodiť počas CDCL
- cryptominisat akceptuje XOR-klauzuly a pri predspracovaní sa na ne díva ako na sústavu lineárnych rovníc nad  $\mathbb{Z}_2$  a používa Gaussovu elimináciu

- desiatky rôznych techník, často doménovo špecifických
- napr. neúplné BDD reprezentácie umožňujú získať klauzuly, ktoré nemožno odvodiť počas CDCL
- cryptominisat akceptuje XOR-klauzuly a pri predspracovaní sa na ne díva ako na sústavu lineárnych rovníc nad  $\mathbb{Z}_2$  a používa Gaussovu elimináciu
- pri „ľahkých“ inštanciách môže predspracovanie zabráť viac času než následné riešenie, treba nájsť vhodný kompromis
- v niektorých prípadoch zase predspracovanie zvyšuje dobu následného riešenia

## Vstupné formuly

---

- niektoré problémy prirodzene vedú skôr k disjunktívnej normálnej forme, štandardný algoritmus úpravy potom vytvára exponenciálne veľkú CNF

## Vstupné formuly

- niektoré problémy prirodzene vedú skôr k disjunktívnej normálnej forme, štandardný algoritmus úpravy potom vytvára exponenciálne veľkú CNF
- riešenie: ekvisplniteľné formuly (*equisatisfiable*)

$$\bigvee_i (a_i \wedge b_i \wedge c_i)$$

$$\left( \bigvee_i z_i \right) \wedge \bigwedge_i [(\overline{z_i} \vee a_i) \wedge (\overline{z_i} \vee b_i) \wedge (\overline{z_i} \vee c_i)]$$

(nie ekvivalentné, lebo sú tam premenné navyše, ale jedna je splniteľná práve vtedy, keď druhá)



- šanca na rýchle objavenie ohodnotenia, v ktorom je formula pravdivá
- neúplné solvery negarantujú dôkaz nesplniteľnosti
- založené na heuristikách (random walks, genetic algorithms, simulated annealing...)

- šanca na rýchle objavenie ohodnotenia, v ktorom je formula pravdivá
- neúplné solvery negarantujú dôkaz nesplniteľnosti
- založené na heuristikách (random walks, genetic algorithms, simulated annealing...)
- úspešné využitie metód štatistickej fyziky (napr. survey propagation pre 3-SAT), lebo náhodný SAT vykazuje podobné správanie (*threshold*, *clustering*)
- automatizované plánovanie: bežne kombinácia neúplného a úplného solvera

- existujúce solvery nie sú dobre paralelizovateľné: vedia použiť mnoho vlákien, ale s otáznym efektom (ak chceme riešiť niekoľko vstupných inštancií, je lepšie riešiť každú v osobitnom vlákne)

- existujúce solvery nie sú dobre paralelizovateľné: vedia použiť mnoho vlákien, ale s otáznym efektom (ak chceme riešiť niekoľko vstupných inštancií, je lepšie riešiť každú v osobitnom vlákne)
- solvery sú konfigurovateľné (lingeling: 300 parametrov); na optimalizáciu na úzkej triede vstupov možno použiť strojové učenie
- zmeny parametrov vedú typicky k zrýchleniu 2–10x

# SAT solvery

---

Verifikácia hardvéru (*informatívne*)

- verifikácia hardvéru je azda najvýznamnejšia oblasť využitia — bez moderných procesorov nevieme robiť žiadne iné výpočty
- softvér sa vymení ľahko, vymieňať hardvér je prakticky nemožné alebo neekonomické; nedá sa opraviť časť procesora
- pri desiatkach miliónov tranzistorov nemáme inú dostatočne výkonnú alternatívu

## 1. Simulácia

- užitočná, ale nič nezaručuje
- je ťažké až nemožné zachytiť všetky možné stavy, v ktorých sa má systém používať

## 1. Simulácia

- užitočná, ale nič nezaručuje
- je ťažké až nemožné zachytiť všetky možné stavy, v ktorých sa má systém používať

## 2. Formálna verifikácia

- v princípe úplný matematický dôkaz správnosti
- nedá sa použiť pre fyzickú vrstvu, ale ideálna pre logickú



## 1. Simulácia

- užitočná, ale nič nezaručuje
- je ťažké až nemožné zachytiť všetky možné stavy, v ktorých sa má systém používať

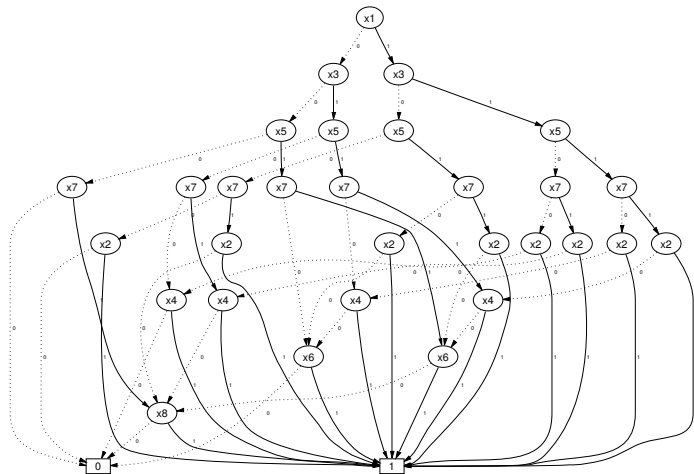
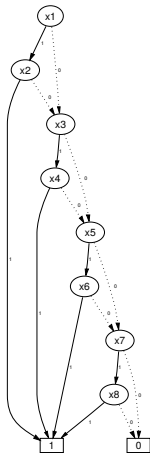
## 2. Formálna verifikácia

- v princípe úplný matematický dôkaz správnosti
- nedá sa použiť pre fyzickú vrstvu, ale ideálna pre logickú
- používa sa zriedka — drahá a vyžaduje vysokú odbornosť
- atómové elektrárne, vesmírne lety, veľké série procesorov

- ekvivalencia boolovských výpočtových okruhov (napr. po optimalizácii)
- dôkaz invariantov
- *safety*: is a state reachable?
- *liveness*: is a state  $T$  always reached after  $S$ ?

# Binary decision diagrams (BDDs)

$$f(x_1, \dots, x_8) = x_1x_2 + x_3x_4 + x_5x_6 + x_7x_8$$



- využívané desiatročia
- nevýhody:
  - poradie premenných musí byť vo všetkých vetvách rovnaké
  - poradie má významný efekt na veľkosť diagramu
  - diagramy môžu byť exponenciálne veľké

- vyjadrenie verifikačných problémov cez splniteľnosť výrokovologických formúl [Biere et al. 1999]
- rozviníme  $k$  krokov výpočtu, skontrolujeme neporušenosť invariantov, zvýšime  $k$
- vyjadrenie v CNF, ráta sa SAT solvermi
- riešiteľné vstupy: 0.4M premenných, 7M klauzúl [2004]

# SAT solvery

---

Kombinatorické problémy (*informatívne*)

- pre mnoho problémov v diskkrétnej matematike je SAT solver jediné v súčasnosti použiteľné riešenie
- pre určité problémy sú špecializované solvery rýchlejšie (napr. TSP)
- pre riešiteľné inštancie sú neraz rýchlejšie špecifické heuristiky
- SAT solverom sa darí výborne, ak počet premenných rastie s veľkosťou problému lineárne (napr. farbenia grafov)

## 3-edge-colouring of cubic graphs

---

Rôzne spôsoby vyjadrenia regulárnosti hranového 3-farbenia pre graf, ktorý má všetky vrcholy stupňa 3:

1. incidentné hrany majú navzájom rôzne farby
2. v každom vrchole je každá farba použitá na práve jednej hrane
3. farby hrán incidentných s každým jedným vrcholom tvoria trojicu z povoleného pevného zoznamu trojíc
4. formula založená na combinatorial nullstellensatz



## 3-edge-colouring of cubic graphs

---

- $O(n)$  premenných,  $O(n)$  klauzúl, veľkosť formuly  $O(n)$
- čas výpočtu pre rôzne solvery a ich konfigurácie computation time: nízka variácia, cca 4x
- voľne koreluje s veľkosťou formuly

## 3-edge-colouring of cubic graphs

---

- $O(n)$  premenných,  $O(n)$  klauzúl, veľkosť formuly  $O(n)$
- čas výpočtu pre rôzne solvery a ich konfigurácie computation time: nízka variácia, cca 4x
- voľne koreluje s veľkosťou formuly
- všetky možnosti fungujú lepšie ako formulácia cez ILP (integer linear programming) a riešenie GLPK či Gurobi

## 3-edge-colouring of cubic graphs

---

- $O(n)$  premenných,  $O(n)$  klauzúl, veľkosť formuly  $O(n)$
- čas výpočtu pre rôzne solvery a ich konfigurácie computation time: nízka variácia, cca 4x
- voľne koreluje s veľkosťou formuly
- všetky možnosti fungujú lepšie ako formulácia cez ILP (integer linear programming) a riešenie GLPK či Gurobi
- pre grafy do 50–100 vrcholov vyhráva backtracking (najmä ak sú zafarbitelné)
- SAT solvery fungujú aj pre tisíce vrcholov
- absencia krátkych kružníc v grafe (čiže lokálne vyzerá ako strom) predlžuje výpočet

## Hamiltonovské kružnice

---

1. premenné  $x_{v,i}$  — true ak  $v$  je  $i$ -ty vrchol kružnice
2. každá pozícia na kružnici má priradený vrchol
3. žiadne dva vrcholy nemajú priradenú tú istú pozíciu
4. každý vrchol je použitý najviac raz
5. každé dva po sebe idúce vrcholy na kružnici sú spojené hranou

# Hamiltonovské kružnice

---

1. premenné  $x_{v,i}$  — true ak  $v$  je  $i$ -ty vrchol kružnice
  2. každá pozícia na kružnici má priradený vrchol
  3. žiadne dva vrcholy nemajú priradenú tú istú pozíciu
  4. každý vrchol je použitý najviac raz
  5. každé dva po sebe idúce vrcholy na kružnici sú spojené hranou
- $O(n^2)$  premenných,  $O(n^3)$  klauzúl!
  - pre kubické grafy funguje po 40–50 vrcholov, podobne ako backtracking
  - ak má byť redukcia na SAT naozaj efektívna, potrebujeme lineárne veľa premenných

# Hamiltonovské kružnice

---

- hamiltonovská kružnica je súvislý 2-faktor (podgraf s vrcholmi stupňa práve 2)
- pomocou boolovskej formuly možno stupeň vrcholo ľahko popísať lokálne
- CNF s veľkosťou  $O(n)$

- hamiltonovská kružnica je súvislý 2-faktor (podgraf s vrcholmi stupňa práve 2)
- pomocou boolovskej formuly možno stupeň vrcholo ľahko popísať lokálne
- CNF s veľkosťou  $O(n)$
- stačí overiť súvislosť každého 2-faktora — AllSAT
- málo dostupných solverov: clasp, BDD\_MINISAT\_ALL
- pre skúmané grafy rýchlejšie ako redukcia na SAT, ale počet 2-faktorov rastie exponenciálne
- pre kubické grafy funguje do cca 40 vrcholov (milióny 2-faktorov)

- z každého SAT solvera možno spraviť AlISAT solver (stačí po každom nájdenom riešení pridať na vstup klauzulu, ktorá ho zakazuje)
- neefektívne, klauzuly objavené CDCL zakaždým zahodíme, keď nanovo štartuje výpočet po objavení riešenia
- vstupná formula narastá príliš rýchlo (napr. pre cryptominisat možno takto realisticky nájsť desaťtisíce riešení, ale nie milióny)
- iná možnosť: obmedziť skoky pre non-chronological backtracking



- doteraz najlepšie riešenie: *formula-BDD caching* [Toda 2015]
- dá sa pridať k akémukoľvek backtrackingu, ak vopred zafixujeme poradie premenných
- zruší prínosy VSIDS (vyberáme si len hodnotu, nie premennú, ktorú ideme ohodnotiť)
- BDD\_MINISAT\_ALL je náročný na pamäť, ale ako jediný dokáže pracovať s miliardami riešení
- prekvapivo vie v niektorých prípadoch dokázať nespĺniteľnosť rýchlejšie ako SAT solvery

# Literatúra

---

Martin Davis and Hillary Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

T. Kruger et al. Too much information: Why CDCL solvers need to forget learned clauses. *Plos One*, 2022.  
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9417043/>.

L. Zhang. SAT-Solving: From Davis-Putnam to Zchaff and beyond. [Online]  
[https://www.inf.ed.ac.uk/teaching/courses/propm/papers/Zhang/sat\\_course1.pdf](https://www.inf.ed.ac.uk/teaching/courses/propm/papers/Zhang/sat_course1.pdf),  
[https://www.inf.ed.ac.uk/teaching/courses/propm/papers/Zhang/sat\\_course2.pdf](https://www.inf.ed.ac.uk/teaching/courses/propm/papers/Zhang/sat_course2.pdf),  
[https://www.inf.ed.ac.uk/teaching/courses/propm/papers/Zhang/sat\\_course3.pdf](https://www.inf.ed.ac.uk/teaching/courses/propm/papers/Zhang/sat_course3.pdf).