8. előadás

XI. 8.

$((b \vee c) \wedge (\neg b \vee \neg c) \wedge (a \vee b) \wedge (\neg a \vee \neg b))$

1. x + +
2. x − −
3. + + x
4. − − x

---

14. − x +
23. + x −

$\underbrace{(b \vee c) \wedge (\neg a \vee \neg b)}_{c \vee \neg a}$

$(\neg a \vee c) \wedge (a \vee \neg c)$

Első megoldás a: igaz
b: hamis
c: igaz

ha a + − + megoldás, akkor az 5. sorba felvéve a − + − ellentmondást,

és a + − + sem lesz megoldás

DPLL algoritmus: (branching algoritmus)

- ha van unit, akkor azt propagálom (levonom a következtetéseket)
- ha nincs unit, akkor kiválasztok egy változót

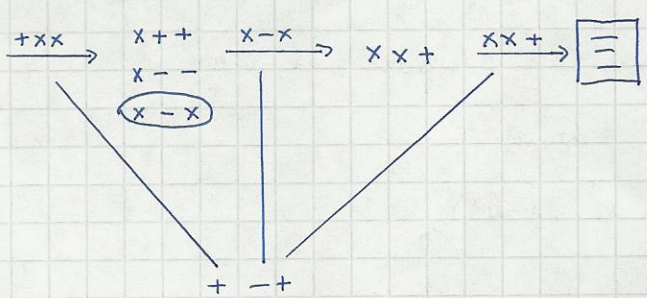
  1) igaz DPLL rekurzívan

  2) hamis DPLL

  return, nincs megoldás

unit: az a clause, amiben csak egy egy − vagy + van és Ezt x.

x + +
x − −
+ + x
− − x



$UP(S, \{u\}) := \{C \setminus \{\neg u\} \mid u \notin C, C \in S\}$

unit propagáció:

1) Törlöm az összes olyan clause-t, melyben előfordul az „u" literál.

2) Ha az „u" negáltja ($\neg u$) előfordul valamely clause-ban, akkor azt törlöm.

Egyik lehetőség, hogy eljutok az üres clause set-ig.

Másik lehetőség, hogy eljutok az üres clause-ig.

DPLL: Davis & Putnam & Loveland & Longman

```
function DPLL (S:clause set): bool;
begin
    if ∅ ∈ S then return False;
    if ∅ = S then return True;   (az üres clause set mindig kielégíthető)
    while
        létezik u ∈ S úgy, hogy |u| = 1 do
        Legyen u ∈ S úgy, hogy |u| = 1
            S := UP(S, u);
    end;
```

S := BCP(S); boolean constant progation

Legyen u egy literál az S clause set-ből.

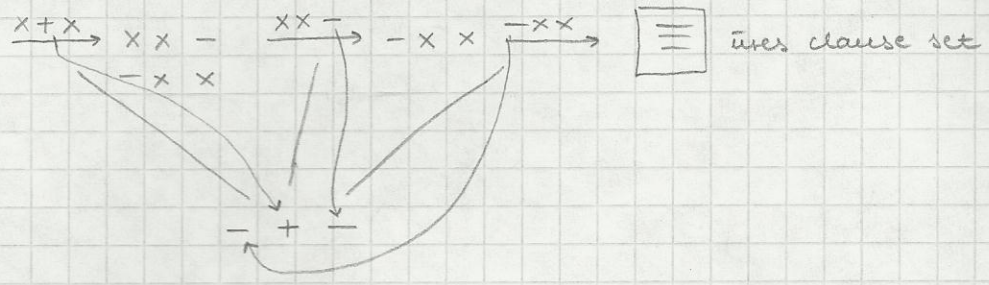
if DPLL (UP(S, {u})) return True

else return DPLL (UP(S, {¬u}));

end;

Pl.:

x	+	+
x	−	−
+	+	x
−	+	−



Pl.:

x	+	+
x	−	−
+	+	x
−	−	x
−	+	−
+	−	+

x−x →

x	x	+
+	x	x
−	x	−

+xx →

x	x	+
x	x	−

xx+ → x xx

üres clause,
tovább nem mehet, amíg
üres clause set-et nem
kapok

Akiről a 2. helyen
− van, azt törlöm

ez eddig unit propagáció

ebbe van unit,
kiválasztok egyet

eddig while állásban voltunk

x	+	+
x	−	−
+	+	x
−	−	x
−	+	−
+	−	+

x+x →

x	x	−
−	x	x
+	x	+

xx− → −xx / +xx

−xx → xxx

Loggnog

9. előadás

XI. 15.

backtracking algoritmus → ábrával rajzolva fán lapul

unit : egyelemű clause

```
+ + −
+ − +
X − X   → unit
```

X : az a változó nem fordul elő az adott clause-ban

Minden sor clause, minden oszlop változó

\+ : pozitív literál
− : negatív literál

Unit : Azok a clause -ok, amelyikben egy literál van.

Unit kielégítése: a clause-t igazzá kell tenni

+	+	−	
+	−	+	
X	−	X	
a	b	c	b : hamis

Literálok közötti kapcsolat : VAGY kapcsolat

A 2. clause ki van elégítve, nem kell vele foglalkozni.

És a végzett művelet a UNIT-propagáció

A 2 1. clause-ban a b helyére, ha „+" volt X-et kell írni

↓ Unit Propagáció (X−X)

$+ \; X \; -$ $\qquad a \vee F \vee \neg c = a \vee \neg c$

2. sor $\quad a \vee \underbrace{\overline{T}}_{\neg b} \vee c = T$

Az utolsó két sort nem kell írni

$\emptyset = S$ → clause set üressé vált

if $\emptyset = S$ then return true;

if $\emptyset \in S$ (→ üres clause) then return false; → nem lehet kielégíteni, mert nincs benne változó, amit igazzá lehet tenni

Breaking rész

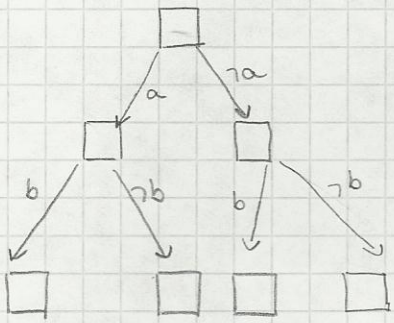
Let a be a literal appearing in S (legyen a egy literál S-ből)

if DPLL(UP(S, {a}) (→ feltételezzük, h. az a változó T.) then return true

else if DPLL(UP(S, {¬a}) (→ a hamis) then return true

else return false

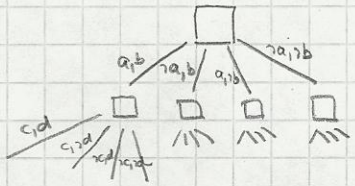
↳ ez a clause set kielégíthetetlen.



unit propagációval propagálom azt a unitot, amely ... →

feltesszük, hogy a igaz v. hamis.

- Breaking felbontja a problémát egyszerűbb problémákra.
- A térfelé vezetés garantálja, h. az algoritmus jó. Vagy T-t ad vissza, vagy F-t.
- Nem csak úgy lehet breaking-algoritmust csinálni, hogy egy változó szerint bontjuk szét.
  - lehet két változó szerint szétszedni



minden lehetőségre ki kell térni

a és b között $\wedge$ kapcsolat van, mert mindkettőnek egyenesnek kell igaznak lennie

$$UP(S, \{u\}) := \{ C \setminus \{\neg u\} \mid u \notin C \}_{C \in S}$$

assignment: hozzárendelés

$$UP(S, \{u\}) := \{ C \setminus \{\neg u\} \mid C \cap \{u\} = \emptyset \}_{C \in S}$$

$$\text{Clause}(C) :\Leftrightarrow \overset{\text{kiper}}{\not\Box} UP(S, A) := \{ C \setminus A \mid C \cap A = \emptyset \}_{C \in S}$$

A: kikiválok halmaza

Assignment(A) :⇔ A kikiválok halmaza

Interpretáció
Clause: $\vee$
assignment: $\wedge$

HUP

a	b	c
+	+	+
+	+	−
+	−	+
−	+	+

a b c
x − −

+	x	x
−	x	x

b = false
c = false

UP

+	+	+
+	+	−
+	−	+
−	+	+

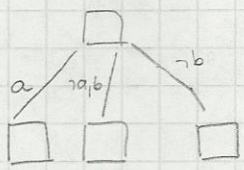
x − x

+	x	+
+	x	−
−	x	+

xx −

+	x	x
−	x	x

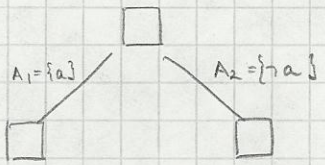
Másik branching algoritmus



SAT megoldó
Egy $\vee$ algoritmus branching algoritmus, ha minden szintre igaz az, hogy a szinten ismert clause-ok $C_1, C_2, \dots C_n$, és a szintről induló (propagált) hozzárendelések $A_1, A_2 \dots A_m$, és a következő clause set

$C_1 \wedge C_2 \wedge \dots \wedge C_n \wedge \neg A_1 \wedge \neg A_2 \wedge \dots \wedge \neg A_m$ kielégíthetetlen.

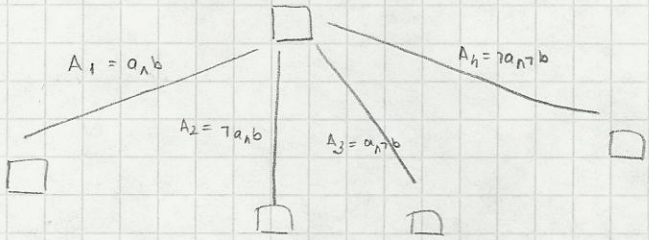
DPLL-re igaz a fenti definíció?



kielégíthetetlen

$\neg A_1 \wedge \neg A_2 = (\neg a) \wedge (a)$

BRENTCHNIG - alg.

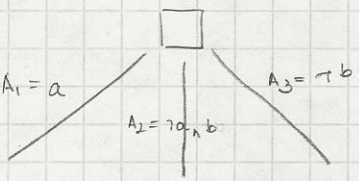


kielégíthetetlen

$\neg A_1 \wedge \neg A_2 \wedge \neg A_3 \wedge \neg A_4 = (\neg a \vee \neg b) \wedge (a \vee \neg b) \wedge (\neg a \vee b) \wedge (a \vee b)$

$- -$
$+ -$
$- +$
$+ +$

A clause tagadása hozzárendelés és fordítva.

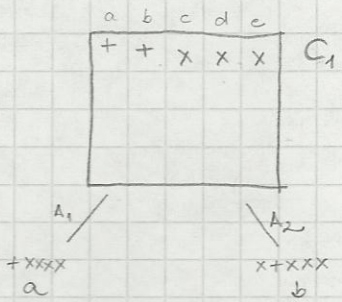


$\neg A_1 \wedge \neg A_2 \wedge \neg A_3 = (\neg a) \wedge (a \vee \neg b) \wedge (b)$

1 − X
2 + −
3 X +
12 X −
123 X X → kielégíthetetlen

DPLL-ből nem használjuk ki azt, hogy ismert clause-okat.

Feladat: Írjunk egy binary branching algoritmust, amelyben benne van a C_1 clause!



$C_1 \wedge \neg A_1 \wedge \neg A_2$

$(a \vee b) \wedge (\neg a) \wedge (\neg b)$

+ − x x x

x + x x x

$(a \vee b) \wedge (\neg a \vee b) \wedge (\neg b)$