Kapitola 3

Problém splnitelnosti

Problém splnitelnosti výrokových formulí, známý také jako *problém SAT*[1] je následující výpočetní problém: Vstupem je výrok φ v CNF (v nějakém rozumném kódování[2]), a úkolem je rozhodnout, zda je φ *splnitelný*.

Jak jsme si ukázali v předchozí kapitole, můžeme každý výrok, nebo i každou výrokovou teorii v konečném jazyce, převést do CNF. Problém SAT je tedy v jistém smyslu univerzální; odpovídá na otázku, zda existuje model.

Známá Cook-Levinova věta říká, že problém SAT je *NP-úplný*, tedy je v třídě NP (pokud nám orákulum prozradí správné ohodnocení proměnných, můžeme snadno ověřit, že všechny klauzule jsou splněny) a každý problém z třídy NP na něj lze převést v polynomiálním čase (konkrétně, výpočet Turingova stroje lze popsat pomocí CNF formule).[3]

Praktické SAT solvery si ale umí poradit s instancemi obsahujícími mnoho, dokonce až miliony, výrokových proměnných a klauzulí. V této kapitole si nejprve ukážeme praktickou aplikaci SAT solveru na problém 'ze života', potom dva fragmenty problému SAT, tzv. *2-SAT* a *Horn-SAT*, pro které existují polynomiální algoritmy, a na závěr si ukážeme také algoritmus DPLL, který je základem (téměř?) všech SAT solverů. (V pozdější kapitole uvidíme také souvislost s *rezoluční metodou*.)

3.1 (draft) SAT solvery

[TODO]

Praktická ukázka použití řešiče SAT na konkrétní problém.

- Problém SAT: Je daná výroková formule splnitelná?
- Příklad *Lze šachovnici bez dvou protilehlých rohů* perfektně *pokrýt kostkami domina?* Snadno vytvoříme výrokovou formuli, která je splnitelná, právě když to lze. Pak ji můžeme zkusit ověřit pomocí nějakého SAT řešiče.
- Nejlepší řešiče pro SAT: `www.satcompetition.org`.
- Řešič v ukázce: `Glucose`, formát pro CNF soubory: `DIMACS`.

[1] Z anglického 'Boolean satisfiability problem'.
[2] Např. DIMACS-CNF formát, viz Wikipedia.
[3] Viz předmět NTIN090 Základy složitosti a vyčíslitelnosti.

- Obecnější otázka: *Lze celou matematiku převést do logických formulí?*

  AI, strojové dokazování, Peano: *Formulario* (1895-1908), Mizar system

- *Proč to lidé (většinou) nedělají?*

  Jak vyřešíme uvedený příklad *elegantněji*? V čem náš postup spočívá?

3.2 2-SAT a implikační graf

Výrok φ je v k-CNF, pokud je v CNF a každá klauzule má nejvýše k literálů. Problému k-SAT se ptá, zda je daný k-CNF formule splnitelná. Pro $k \geq 3$ je k-SAT nadále NP-úplný, každou CNF formuli lze zakódovat do 3-CNF formule:

Cvičení 3.1. Ukažte, že pro každý výrok φ v CNF existuje *ekvisplnitelný* výrok v φ' 3-CNF (tj. φ je splnitelný, právě když φ' je splnitelný), který lze zkonstruovat v lineárním čase.

Pro problém 2-SAT ale existuje polynomiální (dokonce lineární) algoritmus, který si nyní představíme. Algoritmus využívá tzv. *implikačního grafu*. Ukážeme si postup na příkladě:

Příklad 3.2.1. Mějme následující 2-CNF výrok φ:

$$(\neg p_1 \vee p_2) \wedge (\neg p_2 \vee \neg p_3) \wedge (p_1 \vee p_3) \wedge (p_3 \vee \neg p_4) \wedge (\neg p_1 \vee p_5) \wedge (p_2 \vee p_5) \wedge p_1 \wedge \neg p_4$$

Implikační graf

Implikační graf 2-CNF výroku φ je založený na myšlence, že 2-klauzuli $\ell_1 \vee \ell_2$ (kde ℓ_1, ℓ_2 jsou literály) lze chápat jako dvojici implikací: $\overline{\ell_1} \to \ell_2$ a $\overline{\ell_2} \to \ell_1$.[4] Například, z klauzule $\neg p_1 \vee p_2$ vzniknou implikace $p_1 \to p_2$ a také , $\neg p_2 \to \neg p_1$. Tedy pokud p_1 platí v nějakém modelu, musí platit i p_2, a pokud p_2 neplatí, nesmí platit ani p_1. Jednotkovou klauzuli ℓ můžeme také vyjádřit pomocí implikace jako $\overline{\ell} \to \ell$, např. z p_1 dostáváme $\neg p_1 \to p_1$.

Implikační graf $\mathcal{G}_\varphi$ je tedy orientovaný graf, jehož vrcholy jsou všechny literály (proměnné z $\mathrm{Var}(\varphi)$ a jejich negace) a hrany jsou dané implikacemi popsanými výše:

- $V(\mathcal{G}_\varphi) = \{p, \neg p \mid p \in \mathrm{Var}(\varphi)\}$,
- $E(\mathcal{G}_\varphi) = \{(\overline{\ell_1}, \ell_2), (\overline{\ell_2}, \ell_1) \mid \ell_1 \vee \ell_2 \text{ je klauzule } \varphi\} \cup \{(\overline{\ell}, \ell) \mid \ell \text{ je jednotková klauzule } \varphi\}$

V našem příkladě máme množinu vrcholů

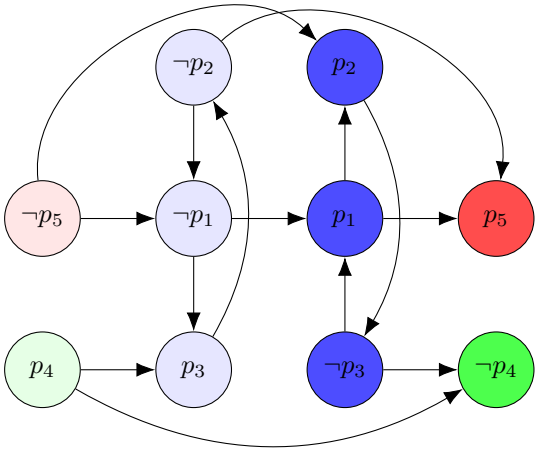
$$V(\mathcal{G}_\varphi) = \{p_1, p_2, p_3, p_4, p_5, \neg p_1, \neg p_2, \neg p_3, \neg p_4, \neg p_5\}$$

a hrany jsou:

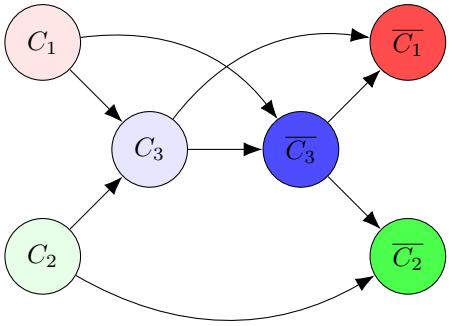
$$E(\mathcal{G}_\varphi) = \{(p_1, p_2), (\neg p_2, \neg p_1), (p_2, \neg p_3), (p_3, \neg p_2), (\neg p_1, p_3), (\neg p_3, p_1), (\neg p_3, \neg p_4), \\ (p_4, p_3), (p_1, p_5), (\neg p_5, \neg p_1), (\neg p_2, p_5), (\neg p_5, p_2), (\neg p_1, p_1), (p_4, \neg p_4)\}$$

Výsledný graf je znázorněný na Obrázku 3.1.

[4] V předchozí kapitole jsme vyjadřovali $p_1 \to p_2$ jako $\neg p_1 \vee p_2$, zde provádíme opačný postup.



Obrázek 3.1: Implikační graf $\mathcal{G}_\varphi$. Komponenty silné souvislosti jsou odlišeny barevně.



Obrázek 3.2: Implikační graf $\mathcal{G}_\varphi$. Graf silně souvislých komponent $\mathcal{G}^*_\varphi$.

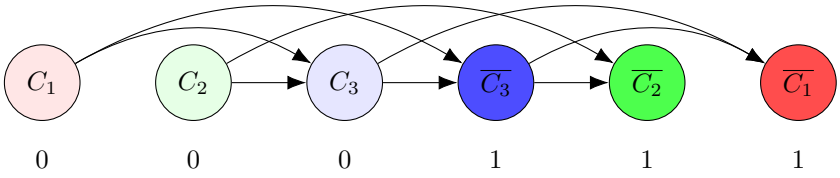
3.2.1 Silně souvislé komponenty

Nyní musíme najít komponenty silné souvislosti[5] tohoto grafu. V našem příkladě dostáváme následující komponenty: $C_1 = \{p_4\}$, $C_2 = \{\neg p_5\}$, $C_3 = \{\neg p_1, \neg p_2, p_3\}$, $\overline{C_3} = \{p_1, p_2, \neg p_3\}$, $\overline{C_2} = \{p_5\}$, $\overline{C_1} = \{\neg p_4\}$.

Všechny literály v jedné komponentě musí být ohodnoceny stejně. Pokud bychom tedy našli dvojici opačných literálů v jedné komponentě, znamená to, že výrok je nesplnitelný. V opačném případě vždy můžeme najít splňující ohodnocení, jak si dokážeme v Tvrzení 3.2.2. Potřebujeme zajistit, aby z žádné komponenty ohodnocené 1 nevedla hrana do komponenty ohodnocené 1. Provedeme-li kontrakci komponent, výsledný graf $\mathcal{G}^*_\varphi$ je acyklický (každý cyklus byl uvnitř nějaké komponenty), a můžeme ho tedy nakreslit v *topologickém uspořádání* (tj. uspořádání na přímce, kde hrany vedou jen doprava), viz Obrázek 3.2.

Při hledání splňujícího ohodnocení (pokud nám nestačí informace, že výrok je splnitelný) potom postupujeme tak, že vezmeme nejlevější dosud neohodnocenou komponentu, ohodnotíme ji 0, opačnou komponentu ohodnotíme 1, a postup opakujeme dokud zbývá nějaká

[5] *Silná souvislost* znamená, že existuje orientovaná cesta z u do v i z v do u, neboli každé dva vrcholy v jedné komponentě leží v orientovaném cyklu. A naopak, každý orientovaný cyklus leží uvnitř nějaké komponenty.



Obrázek 3.3: Implikační graf $\mathcal{G}_\varphi$. Topologické uspořádání grafu $\mathcal{G}_\varphi^*$ a splňující ohodnocení komponent.

neohodnocená komponenta. Například, topologické uspořádání na Obrázku 3.3 odpovídá modelu $v = (1, 1, 0, 0, 1)$.

Na závěr shrneme naše úvahy do následujícího tvrzení:

Tvrzení 3.2.2. *Výrok φ je splnitelný, právě když žádná silně souvislá komponenta v $\mathcal{G}_\varphi$ neobsahuje dvojici opačných literálů $\ell, \overline{\ell}$.*

Důkaz. Každý model, neboli splňující ohodnocení, musí ohodnotit všechny literály ze stejné komponenty stejnou hodnotou. (V opačném případě by nutně existovala implikace $\ell_1 \to \ell_2$, kde ℓ_1 v modelu platí ale ℓ_2 neplatí.) V jedné komponentě tedy nemohou být opačné literály.

Naopak předpokládejme, že žádná komponenta neobsahuje dvojici opačných literálů, a ukažme, že potom existuje model. Označme $\mathcal{G}_\varphi^*$ graf vzniklý z $\mathcal{G}_\varphi$ kontrakcí silně souvislých komponent. Tento graf je acyklický, zvolme nějaké topologické uspořádání. Model zkonstruujeme tak, že zvolíme první dosud neohodnocenou komponentu v našem topologickém uspořádání, všechny literály v ní obsažené ohodnotíme 0, a opačné literály ohodnotíme 1. Takto pokračujeme dokud nejsou všechny komponenty ohodnoceny.

Proč v takto získaném modelu platí výrok φ? Kdyby ne, neplatila by některá z klauzulí. Jednotková klauzule ℓ musí platit, neboť v grafu $\mathcal{G}_\varphi$ máme hranu $\overline{\ell} \to \ell$. Stejná hrana je i v grafu komponent, tedy $\overline{\ell}$ předchází v topologickém uspořádání komponentu obsahující ℓ. Při konstrukci modelu jsme museli ohodnotit $\overline{\ell}$ dříve než ℓ, tedy $\overline{\ell} = 0$ a $\ell = 1$. Podobně, 2-klauzule $\ell_1 \vee \ell_2$ také musí platit: máme hrany $\overline{\ell_1} \to \ell_2$ a $\overline{\ell_2} \to \ell_1$. Pokud jsme ℓ_1 ohodnotili dříve než ℓ_2, museli jsme kvůli hraně $\overline{\ell_1} \to \ell_2$ ohodnotit $\overline{\ell_1} = 0$, tedy ℓ_1 platí. Podobně pokud jsme ohodnotili nejdříve ℓ_2, musí být $\overline{\ell_2} = 0$ a $\ell_2 = 1$. □

Důsledek 3.2.3. *Problém 2-SAT je řešitelný v lineárním čase. V lineárním čase můžeme také zkonstruovat model, pokud existuje.*

Důkaz. Komponenty silné souvislosti lze snadno nalézt v v čase $\mathcal{O}(|V| + |E|)$, topologické uspořádání můžeme také zkonstruovat v čase $\mathcal{O}(|V| + |E|)$. □

Cvičení 3.2. Najděte nějaký nesplnitelný 2-CNF výrok, sestrojte jeho implikační graf, a přesvědčete se, že existuje dvojice opačných literálů ve stejné komponentě silné souvislosti.

Cvičení 3.3. Najděte všechna topologická uspořádání grafu $\mathcal{G}_\varphi^*$ z příkladu výše a jim odpovídající modely. Rozmyslete si, proč takto získáme právě všechny modely výroku φ.

Cvičení 3.4. Rozmyslete si, proč lze komponenty i topologické uspořádání nalézt v čase $\mathcal{O}(|V| + |E|)$.

3.3 Horn-SAT a jednotková propagace

Nyní si ukážeme další fragment SATu řešitelný v polynomiálním čase, tzv. *Horn-SAT* neboli problém splnitelnosti *hornovských výroků*. Výrok je v *hornovský (v Hornově tvaru)*[6], pokud je konjunkcí *hornovských klauzulí*, tj. klauzulí obsahujících *nejvýše jeden *pozitivní* literál*. Význam Hornovských klauzulí vyplývá z ekvivalentního vyjádření ve formě implikace:

$$\neg p_1 \vee \neg p_2 \vee \cdots \vee \neg p_n \vee q \;\sim\; (p_1 \wedge p_2 \wedge \cdots \wedge p_n) \to q$$

Hornovské formule tedy dobře modelují systémy, kde splnění určitých podmínek zaručuje splnění jiné podmínky. Upozorněme, že jednotková klauzule ℓ je také hornovská. V kontextu logického programování se jí říká *fakt*, pokud je literál pozitivní, a *cíl* pokud je negativní.[7] Hornovské formule s alespoň jedním pozitivním a alespoň jedním negativním literálem jsou *pravidla*.

Příklad 3.3.1. Příkladem výroku, který je v CNF, ale není hornovský, je třeba $(p_1 \vee p_2 \vee \neg p_3) \wedge (\neg p_1 \vee p_3)$. Jako příklad, na kterém budeme ilustrovat algoritmus, nám poslouží následující hornovský výrok:

$$\varphi = (\neg p_1 \vee p_2) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (\neg_3 \vee \neg p_4) \wedge (\neg p_5 \vee \neg p_4) \wedge p_4$$

Polynomiální algoritmus pro řešení problému Horn-SAT je založený na jednoduché myšlence *jednotkové propagace*: Pokud náš výrok obsahuje *jednotkovou* klauzuli, víme, jak musí být ohodnocenena výroková proměnná obsažená v této klauzuli. A tuto znalost můžeme *propagovat*—využít k zjednodušení výroku.

Náš výrok φ obsahuje jednotkovou klauzuli p_4. Víme tedy, že v každém jeho modelu $v \in \mathrm{M}(\varphi)$ musí platit $v(p_4) = 1$. To ale znamená, že v libovolném modelu výroku φ

- každá klauzule obsahující pozitivní literál p_4 je splněna, můžeme ji tedy z výroku odstranit,
- negativní literál $\neg p_4$ nemůže být splněn, můžeme ho tedy odstranit ze všech klauzulí, které ho obsahují.

Tomu kroku se říká *jednotková propagace*. Výsledkem je následující zjednodušený výrok, který označíme φ^{p_4} (obecně φ^{ℓ} máme-li jednotkovou klauzuli ℓ):

$$\varphi^{p_4} = (\neg p_1 \vee p_2) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (\neg_3 \vee \neg p_4) \wedge \neg p_5$$

Pozorování 3.3.2. *Všimněte si, že φ^{ℓ} už neobsahuje literál ℓ ani $\overline{\ell}$, a zřejmě platí, že modely φ jsou právě modely $\{\varphi^{\ell}, \ell\}$, neboli modely φ^{ℓ} v původním jazyce $\mathbb{P}$, ve kterých platí ℓ.*

Jednotkovou propagací jsme získali ve výroku φ^{p_4} novou jednotkovou klauzuli $\neg p_5$, můžeme tedy pokračovat nastavením $v(p_5) = 0$ a další jednotkovou propagací:

$$(\varphi^{p_4})^{\neg p_5} = (\neg p_1 \vee p_2) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_3 \vee \neg p_4)$$

[6] Matematik Alfred Horn objevil význam tohoto tvaru logických formulí (a položil tak základ logickému programování) v roce 1951.

[7] Neboť dokazujeme sporem, více v pozdější kapitole o rezoluci a Prologu.

Výsledný výrok už neobsahuje jednotkovou klauzuli. To ale znamená, že každá klauzule obsahuje alespoň dva literály, a nejvýše jeden z nich může být pozitivní! (Zde potřebujeme hornovskost výroku.) Protože každá klauzule obsahuje negativní literál, stačí ohodnotit všechny zbývající proměnné 0, a výrok bude splněn: $v(p_1) = v(p_2) = v(p_3) = 0$. Dostáváme tedy model $v = (0, 0, 0, 1, 1)$.

Příklad 3.3.3. Co by se stalo, pokud by výrok nebyl splnitelný? Podívejme se na výrok

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

a provádějme jednotkovou propagaci jako v předchozím příkladě: máme $v(p) = 1$ a $\psi^p = q \wedge (\neg q \vee r) \vee \neg r$, dále $v(q) = 1$ a $(\psi^p)^q = r \vee \neg r$. Tento výrok je nesplnitelný, neboť obsahuje dvojici opačných jednotkových klauzulí. [8]

Shrňme si nyní algoritmus pro řešení problému Horn-SAT:

Algoritmus (Horn-SAT)**. vstup**: Vstup: výrok φ v Hornově tvaru, výstup: model φ nebo informace, že φ není splnitelný

1. Pokud φ obsahuje dvojici opačných jednotkových klauzulí $\ell, \overline{\ell}$, není splnitelný.
2. Pokud φ neobsahuje žádnou jednotkovou klauzuli, je splnitelný, ohodnoť všechny zbývající proměnné 0.
3. Pokud φ obsahuje jednotkovou klauzuli ℓ, ohodnoť literál ℓ hodnotou 1, proveď jednotkovou propagaci, nahraď φ výrokem φ^ℓ, a vrať se na začátek.

Tvrzení 3.3.4. *Algoritmus je korektní.*

Důkaz. Korektnost plyne z Pozorování a z předchozí diskuze. □

Důsledek 3.3.5. *Horn-SAT lze řešit v lineárním čase.*

Důkaz. V každém kroku stačí projít výrok jednou, a jednotková propagace výrok vždy zkrátí. Z toho snadno plyne kvadratický horní odhad, ale při vhodné implementaci lze dosáhnout lineárního času vzhledem k délce φ. □

Cvičení 3.5. Navrhněte implementaci algoritmu pro Horn-SAT v lineárním čase.

Cvičení 3.6. Navrhněte modifikaci algoritmu pro Horn-SAT, která najde všechny modely.

3.4 DPLL algoritmus pro řešení problému SAT

Na závěr kapitoly o problému splnitelnosti si představíme zdaleka nejpoužívanější algoritmus pro řešení obecného problému SAT, algoritmus DPLL.[9] Ačkoliv v nejhorším případě má exponenciální složitost, v praxi funguje velmi efektivně.

Algoritmus používá jednotkovou propagaci spolu s následujícím pozorováním: Řekneme, že literál ℓ má *čistý výskyt* v φ, pokud se vyskytuje ve φ, ale opačný literál $\overline{\ell}$ se ve φ nevyskytuje. Máme-li literál s čistým výskytem, můžeme jeho hodnotu nastavit na 1, a splnit (a odstranit) tak všechny klauzule, které ho obsahují. Pokud výrok neumíme takto zjednodušit, rozvětvíme výpočet dosazením obou možných hodnot pro vybranou výrokovou proměnnou.

[8] Jinými slovy, v dalším kroku bychom provedli jednotkovou propagaci r, odstranili jednotkovou klauzuli r, a ze zbývající jednotkové klauzule $\neg r$ bychom odstranili literál $\neg r$, čímž by vznikla *prázdná klauzule*, která je nesplnitelná.

[9] Pojmenovaný po svých tvůrcích, Davis-Putnam-Logemann-Loveland, pochází z roku 1961.

Algoritmus (DPLL)**. vstup**: Vstup: výrok φ v CNF, výstup: model φ nebo informace, že φ není splnitelný

1. Dokud φ obsahuje jednotkovou klauzuli ℓ, ohodnoť literál ℓ hodnotou 1, proveď jednotkovou propagaci, a nahraď φ výrokem φ^ℓ.
2. Dokud existuje literál ℓ, který má ve φ čistý výskyt, ohodnoť ℓ hodnotou 1, a odstraň klauzule obsahující ℓ.
3. Pokud φ neobsahuje žádnou klauzuli, je splnitelný.
4. Pokud φ obsahuje prázdnou klauzuli, není splnitelný.
5. Jinak zvol dosud neohodnocenou výrokovou proměnnou p, a zavolej algoritmus rekurzivně na $\varphi \wedge p$ a na $\varphi \wedge \neg p$.

To, že je algoritmus v nejhorším případě exponenciální, lze snadno nahlédnout na příkladě jediné klauzule $p_1 \vee p_2 \vee \cdots \vee p_n$. Korektnost není těžké ověřit.

Tvrzení 3.4.1. *Algoritmus DPLL řeší problém SAT.*

Příklad 3.4.2. Ukážeme si běh algoritmu na následujícím příkladě:

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

Výrok nemá žádnou jednotkovou klauzuli. Literál $\neg r$ má čistý výskyt, nastavíme $v(r) = 0$ a odstraníme klauzule obsahující $\neg r$:

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

Žádný další literál nemá čistý výskyt. Spustíme proto rekurzivně algoritmus:

(p=1) Přidáme jednotkovou klauzuli p:

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

Nastavíme $v(p) = 1$ a provedeme jednotkovou propagaci: $(\neg q \vee \neg s) \wedge (q \vee s)$. Nyní rozvětvíme na proměnné q:

(q=1) $(\neg q \vee \neg s) \wedge (q \vee s) \wedge q$. Po nastavení $v(q) = 1$ a jednotkové propagaci dostáváme s, po nastavení $v(s) = 1$ a jednotkové propagaci dostáváme výrok neobsahující žádnou klauzuli, je tedy splnitelný ohodnocením $(1, 1, 0, *, *)$. Odpověď na problém splnitelnosti už máme, pokud chceme znát všechny modely, můžeme dokončit ostatní větve výpočtu.

(q=0) $(\neg q \vee \neg s) \wedge (q \vee s) \wedge \neg q$. Dostáváme modely $(1, 0, 0, *, *)$.

(p=0) Přidáme jednotkovou klauzuli $\neg p$:

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

Po provedení jednotkové propagace $\neg p$ máme $s \wedge \neg s \wedge (q \vee s)$. Po provedení jednotkové propagace s máme $\square \wedge q$, kde $\square$ je prázdná klauzule. Výrok je tedy nesplnitelný a v této větvi nedostaneme žádné modely.

Zjistili jsme, že původní výrok je splnitelný, má 8 modelů, konkrétně: $\mathrm{M}_\varphi = \{(1, a, 0, b, c) \mid a, b, c \in \{0, 1\}, \}$.[10]

[10] To znamená, že je ekvivalentní výroku $p \wedge \neg r$.