Třetí přednáška

NAIL062 Výroková a predikátová logika

Jakub Bulín (KTIML MFF UK)

Zimní semestr 2024

Třetí přednáška

Program

- problém splnitelnosti, SAT solvery
- 2-SAT a implikační graf
- Horn-SAT a jednotková propagace
- algoritmus DPLL
- úvod do tablo metody

Materiály

Zápisky z přednášky, Kapitola 3, Sekce 4.1-4.2 z Kapitoly 4

Kapitola 3: Problém splnitelnosti

Problém splnitelnosti Booleovských formulí

Problém SAT:

- vstup: výrok φ v CNF
- otázka: je φ splnitelný?

Problém splnitelnosti Booleovských formulí

Problém SAT:

- vstup: výrok φ v CNF
- otázka: je φ splnitelný?

univerzální problém: každou teorii nad konečným jazykem lze převést do CNF

Problém splnitelnosti Booleovských formulí

Problém SAT:

- vstup: výrok φ v CNF
- otázka: je φ splnitelný?

univerzální problém: každou teorii nad konečným jazykem lze převést do CNF

Cook-Levinova věta: SAT je NP-úplný (důkaz: formalizuj výpočet nedeterministického Turingova stroje ve výrokové logice)

Problém splnitelnosti Booleovských formulí

Problém SAT:

- vstup: výrok φ v CNF
- otázka: je φ splnitelný?

univerzální problém: každou teorii nad konečným jazykem lze převést do CNF

Cook-Levinova věta: SAT je NP-úplný (důkaz: formalizuj výpočet nedeterministického Turingova stroje ve výrokové logice)

ale některé *fragmenty* jsou v P, efektivně řešitelné, např. 2-SAT a Horn-SAT (viz Sekce 3.2 a 3.3)

Problém splnitelnosti Booleovských formulí

Problém SAT:

- vstup: výrok φ v CNF
- otázka: je φ splnitelný?

univerzální problém: každou teorii nad konečným jazykem lze převést do CNF

Cook-Levinova věta: SAT je NP-úplný (důkaz: formalizuj výpočet nedeterministického Turingova stroje ve výrokové logice)

ale některé *fragmenty* jsou v P, efektivně řešitelné, např. 2-SAT a Horn-SAT (viz Sekce 3.2 a 3.3)

praktický problém: moderní *SAT solvery* (viz Sekce 3.1) se používají v řadě odvětví aplikované informatiky, poradí si s obrovskými instancemi

3.1 SAT solvery

SAT solvery

- existují od 60. let 20. století, v 21. století dramatický rozvoj dnes až 10^8 proměnných, viz www.satcompetition.org.

SAT solvery

- existují od 60. let 20. století, v 21. století dramatický rozvoj dnes až 10^8 proměnných, viz www.satcompetition.org.
- nejčastěji založeny na jednoduchém algoritmu DPLL (viz Sekce 3.4), umí i najít řešení (model)

SAT solvery

- existují od 60. let 20. století, v 21. století dramatický rozvoj dnes až 10^8 proměnných, viz www.satcompetition.org.
- nejčastěji založeny na jednoduchém algoritmu DPLL (viz Sekce 3.4), umí i najít řešení (model)
- různá rozšíření, zejména Conflict-driven clause learning (CDCL)

SAT solvery

- existují od 60. let 20. století, v 21. století dramatický rozvoj dnes až 10^8 proměnných, viz www.satcompetition.org.
- nejčastěji založeny na jednoduchém algoritmu DPLL (viz Sekce 3.4), umí i najít řešení (model)
- různá rozšíření, zejména Conflict-driven clause learning (CDCL)
- řada technologií pro efektivnější řešení instancí pocházejících z různých aplikačních domén, heuristiky pro řízení prohledávání (za použití ML, NN) — desítky tisíc řádků kódu

SAT solvery

- existují od 60. let 20. století, v 21. století dramatický rozvoj dnes až 10^8 proměnných, viz www.satcompetition.org.
- nejčastěji založeny na jednoduchém algoritmu DPLL (viz Sekce 3.4), umí i najít řešení (model)
- různá rozšíření, zejména Conflict-driven clause learning (CDCL)
- řada technologií pro efektivnější řešení instancí pocházejících z různých aplikačních domén, heuristiky pro řízení prohledávání (za použití ML, NN) — desítky tisíc řádků kódu

Praktická ukázka: boardomino

Lze pokrýt šachovnici s chybějícími dvěma protilehlými rohy perfektně pokrýt kostkami domina?

SAT solvery

- existují od 60. let 20. století, v 21. století dramatický rozvoj dnes až 10^8 proměnných, viz www.satcompetition.org.
- nejčastěji založeny na jednoduchém algoritmu DPLL (viz Sekce 3.4), umí i najít řešení (model)
- různá rozšíření, zejména Conflict-driven clause learning (CDCL)
- řada technologií pro efektivnější řešení instancí pocházejících z různých aplikačních domén, heuristiky pro řízení prohledávání (za použití ML, NN) — desítky tisíc řádků kódu

Praktická ukázka: boardomino

Lze pokrýt šachovnici s chybějícími dvěma protilehlými rohy perfektně pokrýt kostkami domina?

těžká instance SATu (proč?), jak zakódovat?

SAT solvery

- existují od 60. let 20. století, v 21. století dramatický rozvoj dnes až 10^8 proměnných, viz www.satcompetition.org.
- nejčastěji založeny na jednoduchém algoritmu DPLL (viz Sekce 3.4), umí i najít řešení (model)
- různá rozšíření, zejména Conflict-driven clause learning (CDCL)
- řada technologií pro efektivnější řešení instancí pocházejících z různých aplikačních domén, heuristiky pro řízení prohledávání (za použití ML, NN) — desítky tisíc řádků kódu

Praktická ukázka: boardomino

Lze pokrýt šachovnici s chybějícími dvěma protilehlými rohy perfektně pokrýt kostkami domina?

těžká instance SATu (proč?), jak zakódovat?

řešič Glucose, formát vstupu: DIMACS CNF

3.2 2-SAT a implikační graf

2-SAT vs. 3-SAT

- k-CNF: CNF a každá klauzule nejvýše k literálů

2-SAT vs. 3-SAT

- k-CNF: CNF a každá klauzule nejvýše k literálů
- k-SAT: je daný k-CNF výrok splnitelný?

2-SAT vs. 3-SAT

- k-CNF: CNF a každá klauzule nejvýše k literálů
- k-SAT: je daný k-CNF výrok splnitelný?
- k-SAT je NP-úplný pro $k \geq 3$ (ke každému výroku lze sestrojit ekvisplnitelný 3-CNF výrok)

2-SAT vs. 3-SAT

- k-CNF: CNF a každá klauzule nejvýše k literálů
- k-SAT: je daný k-CNF výrok splnitelný?
- k-SAT je NP-úplný pro $k \geq 3$ (ke každému výroku lze sestrojit ekvisplnitelný 3-CNF výrok)
- ale 2-SAT je v P, dokonce řešitelný v lineárním čase

2-SAT vs. 3-SAT

- k-CNF: CNF a každá klauzule nejvýše k literálů
- k-SAT: je daný k-CNF výrok splnitelný?
- k-SAT je NP-úplný pro $k \geq 3$ (ke každému výroku lze sestrojit ekvisplnitelný 3-CNF výrok)
- ale 2-SAT je v P, dokonce řešitelný v lineárním čase
- algoritmus využívá tzv. implikační graf:

2-SAT vs. 3-SAT

- k-CNF: CNF a každá klauzule nejvýše k literálů
- k-SAT: je daný k-CNF výrok splnitelný?
- k-SAT je NP-úplný pro $k \geq 3$ (ke každému výroku lze sestrojit ekvisplnitelný 3-CNF výrok)
- ale 2-SAT je v P, dokonce řešitelný v lineárním čase
- algoritmus využívá tzv. implikační graf:
  - 2-klauzule $p \vee q$ je ekvivalentní $\neg p \to q$ a také $\neg q \to p$

2-SAT vs. 3-SAT

- k-CNF: CNF a každá klauzule nejvýše k literálů
- k-SAT: je daný k-CNF výrok splnitelný?
- k-SAT je NP-úplný pro $k \geq 3$ (ke každému výroku lze sestrojit ekvisplnitelný 3-CNF výrok)
- ale 2-SAT je v P, dokonce řešitelný v lineárním čase
- algoritmus využívá tzv. implikační graf:
  - 2-klauzule $p \vee q$ je ekvivalentní $\neg p \to q$ a také $\neg q \to p$
  - $p \sim p \vee p$ je ekvivalentní $\neg p \to p$

2-SAT vs. 3-SAT

- k-CNF: CNF a každá klauzule nejvýše k literálů
- k-SAT: je daný k-CNF výrok splnitelný?
- k-SAT je NP-úplný pro $k \geq 3$ (ke každému výroku lze sestrojit ekvisplnitelný 3-CNF výrok)
- ale 2-SAT je v P, dokonce řešitelný v lineárním čase
- algoritmus využívá tzv. implikační graf:
  - 2-klauzule $p \vee q$ je ekvivalentní $\neg p \to q$ a také $\neg q \to p$
  - $p \sim p \vee p$ je ekvivalentní $\neg p \to p$
  - vrcholy jsou literály

2-SAT vs. 3-SAT

- k-CNF: CNF a každá klauzule nejvýše k literálů
- k-SAT: je daný k-CNF výrok splnitelný?
- k-SAT je NP-úplný pro $k \geq 3$ (ke každému výroku lze sestrojit ekvisplnitelný 3-CNF výrok)
- ale 2-SAT je v P, dokonce řešitelný v lineárním čase
- algoritmus využívá tzv. implikační graf:
  - 2-klauzule $p \vee q$ je ekvivalentní $\neg p \to q$ a také $\neg q \to p$
  - $p \sim p \vee p$ je ekvivalentní $\neg p \to p$
  - vrcholy jsou literály
  - hrany dané implikacemi

2-SAT vs. 3-SAT

- k-CNF: CNF a každá klauzule nejvýše k literálů
- k-SAT: je daný k-CNF výrok splnitelný?
- k-SAT je NP-úplný pro $k \geq 3$ (ke každému výroku lze sestrojit ekvisplnitelný 3-CNF výrok)
- ale 2-SAT je v P, dokonce řešitelný v lineárním čase
- algoritmus využívá tzv. implikační graf:
  - 2-klauzule $p \vee q$ je ekvivalentní $\neg p \to q$ a také $\neg q \to p$
  - $p \sim p \vee p$ je ekvivalentní $\neg p \to p$
  - vrcholy jsou literály
  - hrany dané implikacemi
  - **myšlenka**: ohodnotíme-li vrchol 1, všude kam se dostaneme po hranách (komponenta silné souvislosti) musí být také 1

Implikační graf

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

Implikační graf

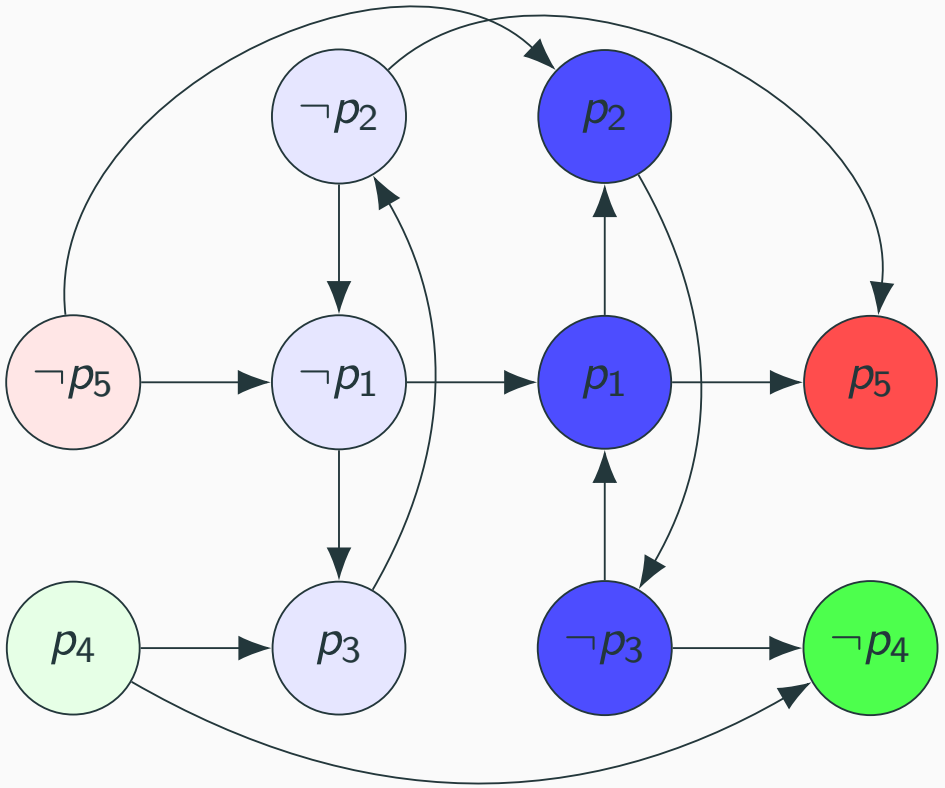
$$
\begin{aligned}
V(\mathcal{G}_\varphi) =& \{p, \neg p \mid p \in \mathrm{Var}(\varphi)\},\\
E(\mathcal{G}_\varphi) =& \{(\overline{\ell_1}, \ell_2), (\overline{\ell_2}, \ell_1) \mid \ell_1 \vee \ell_2 \text{ je klauzule } \varphi\} \cup\\
& \{(\overline{\ell}, \ell) \mid \ell \text{ je jednotková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

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

Implikační graf

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

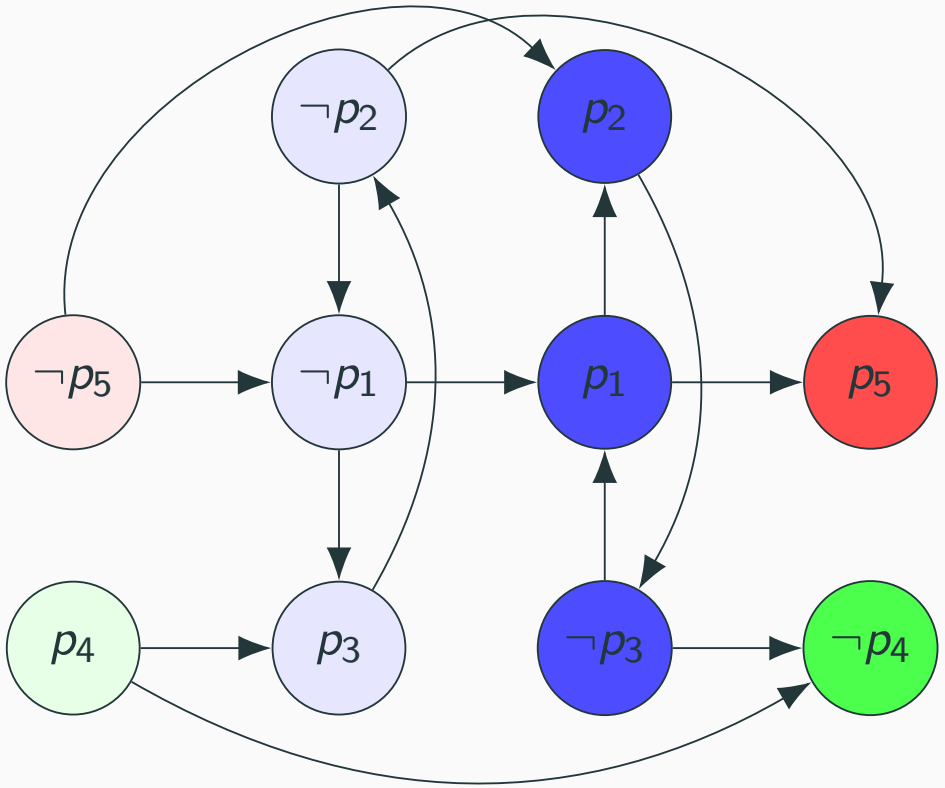
$(\neg p_1 \vee p_2) \wedge (\neg p_2 \vee \neg p_3) \wedge (p_1 \vee p_3) \wedge (p_3 \vee \neg p_4) \wedge (\neg p_1 \vee p_5) \wedge (p_2 \vee p_5) \wedge p_1 \wedge \neg p_4$



Implikační graf

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

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

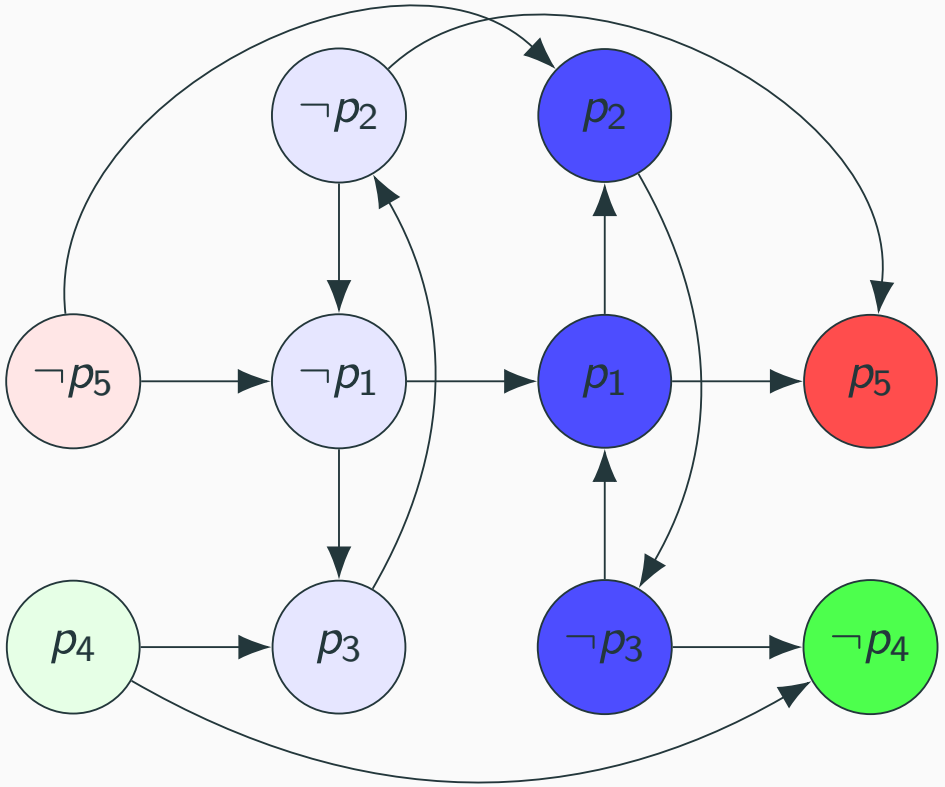


- najdeme komponenty silné souvislosti

Implikační graf

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

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

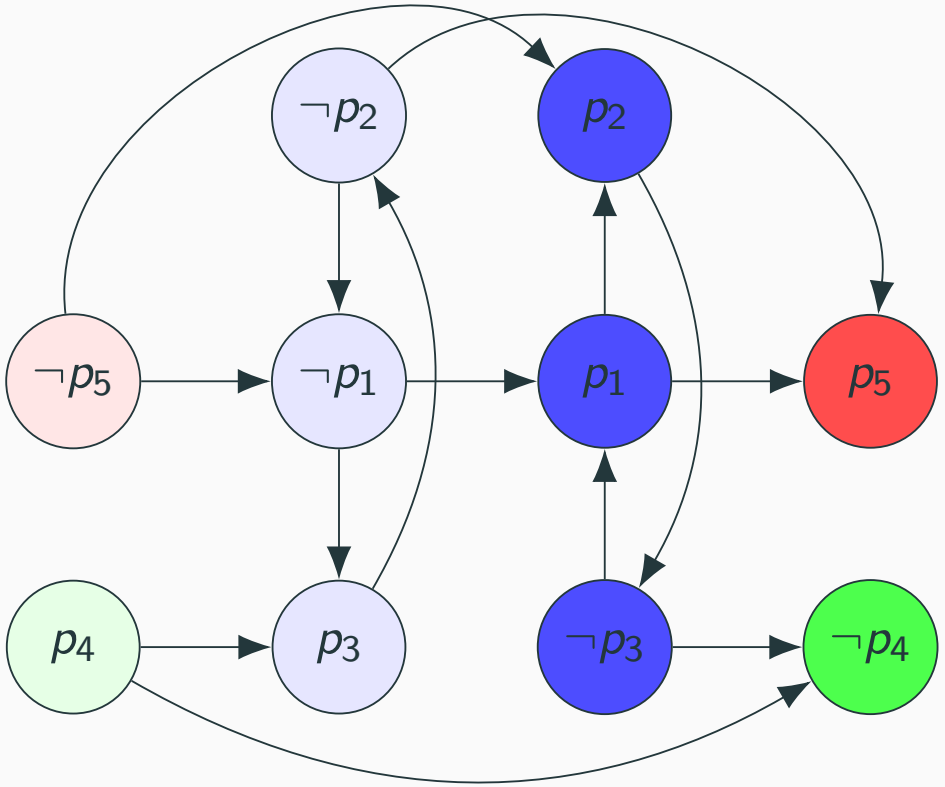


- najdeme komponenty silné souvislosti
- literály v komponentě musí být ohodnoceny stejně (jinak "$1 \to 0$")

Implikační graf

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

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

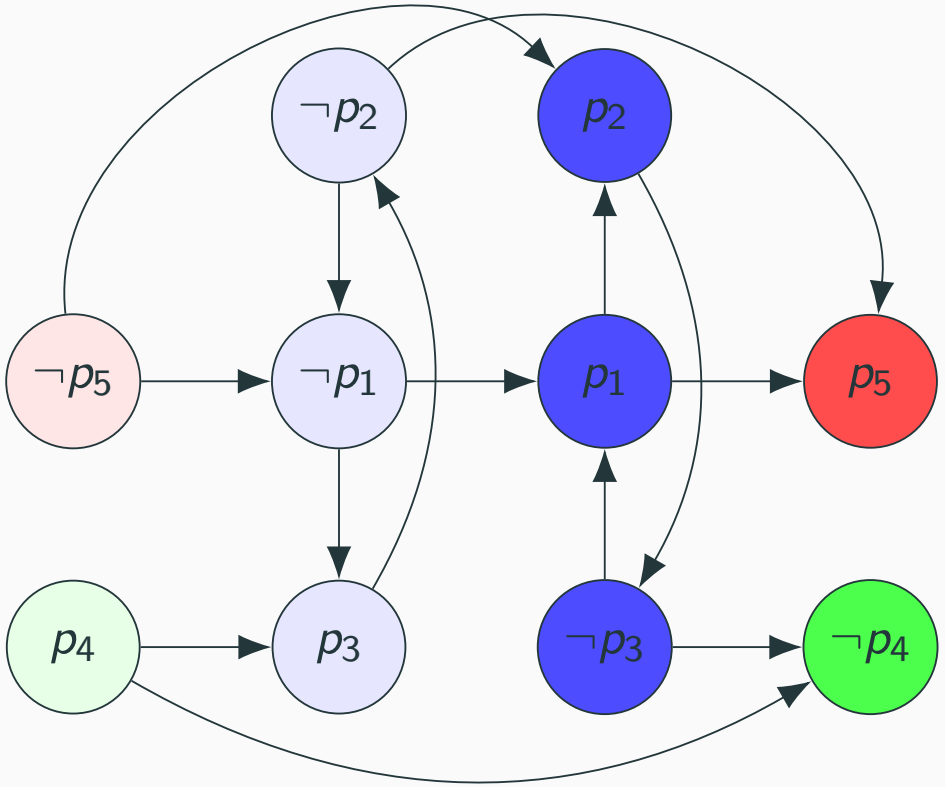


- najdeme komponenty silné souvislosti
- literály v komponentě musí být ohodnoceny stejně (jinak "$1 \to 0$")
- pokud má nějaká komponenta opačné literály, je φ nesplnitelný

Implikační graf

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

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



- najdeme komponenty silné souvislosti
- literály v komponentě musí být ohodnoceny stejně (jinak "$1 \to 0$")
- pokud má nějaká komponenta opačné literály, je φ nesplnitelný
- jinak sestrojíme model

Konstrukce modelu

Všimněte si: stačí, aby z žádné komponenty ohodnocené 1 nevedla hrana do komponenty ohodnocené 0

Konstrukce modelu

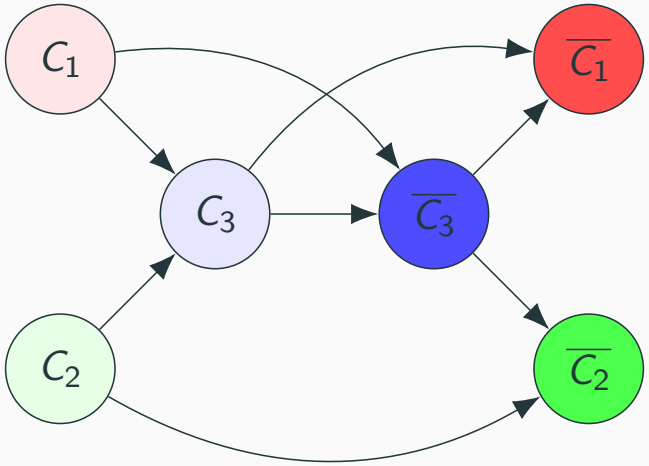
Všimněte si: stačí, aby z žádné komponenty ohodnocené 1 nevedla hrana do komponenty ohodnocené 0

provedeme kontrakci komponent, výsledný graf $\mathcal{G}_\varphi^*$ je acyklický

Konstrukce modelu

Všimněte si: stačí, aby z žádné komponenty ohodnocené 1 nevedla hrana do komponenty ohodnocené 0

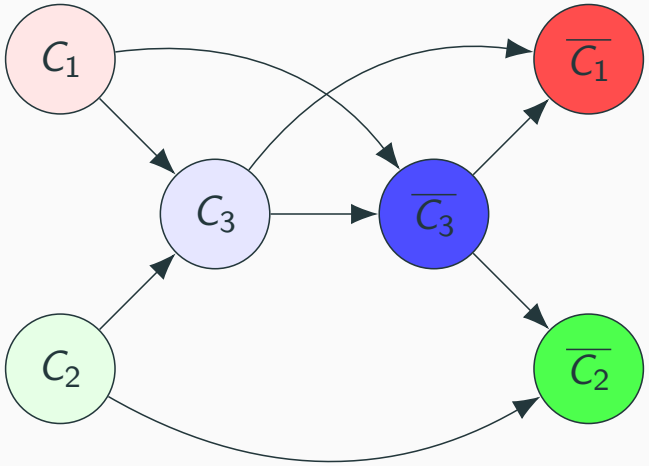
provedeme kontrakci komponent, výsledný graf $\mathcal{G}^*_\varphi$ je acyklický



Konstrukce modelu

Všimněte si: stačí, aby z žádné komponenty ohodnocené 1 nevedla hrana do komponenty ohodnocené 0

provedeme kontrakci komponent, výsledný graf $\mathcal{G}^*_\varphi$ je acyklický

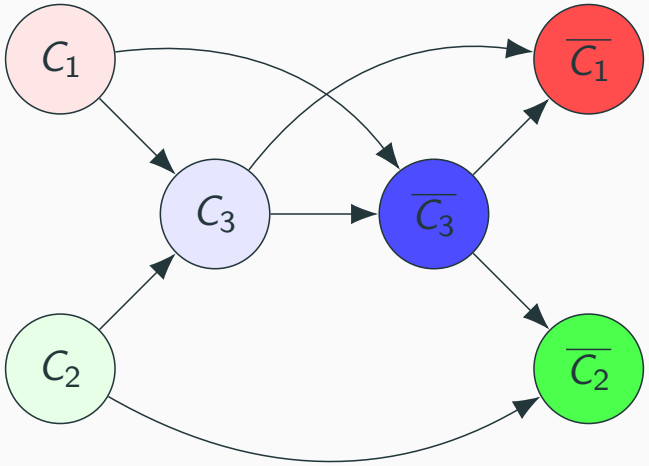


najdeme nějaké topologické uspořádání; v něm najdeme nejlevější dosud neohodnocenou komponentu, ohodnotíme ji 0, opačnou komponentu ohodnotíme 1, a opakujeme

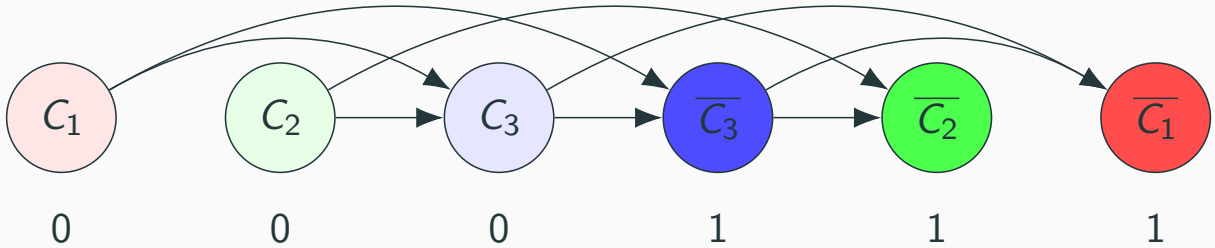
Konstrukce modelu

Všimněte si: stačí, aby z žádné komponenty ohodnocené 1 nevedla hrana do komponenty ohodnocené 0

provedeme kontrakci komponent, výsledný graf $\mathcal{G}^*_\varphi$ je acyklický



najdeme nějaké topologické uspořádání; v něm najdeme nejlevější dosud neohodnocenou komponentu, ohodnotíme ji 0, opačnou komponentu ohodnotíme 1, a opakujeme



Shrnutí

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

Shrnutí

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

Důkaz: $\Rightarrow$ literály v komponentě musí být ohodnoceny stejně

Shrnutí

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

Důkaz: $\Rightarrow$ literály v komponentě musí být ohodnoceny stejně

$\Leftarrow$ ohodnocení zkonstruované výše je model φ:

Shrnutí

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

Důkaz: $\Rightarrow$ literály v komponentě musí být ohodnoceny stejně

$\Leftarrow$ ohodnocení zkonstruované výše je model φ:

- jednotková klauzule ℓ platí kvůli hraně $\overline{\ell} \to \ell$, komponenta s $\overline{\ell}$ byla ohodnocena dříve, a to 0, takže $v(\ell) = 1$

Shrnutí

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

Důkaz: $\Rightarrow$ literály v komponentě musí být ohodnoceny stejně

$\Leftarrow$ ohodnocení zkonstruované výše je model φ:

- jednotková klauzule ℓ platí kvůli hraně $\overline{\ell} \to \ell$, komponenta s $\overline{\ell}$ byla ohodnocena dříve, a to 0, takže $v(\ell) = 1$
- podobně pro 2-klauzuli $\ell_1 \vee \ell_2$, máme hrany $\overline{\ell_1} \to \ell_2$, $\overline{\ell_2} \to \ell_1$

Shrnutí

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

Důkaz: $\Rightarrow$ literály v komponentě musí být ohodnoceny stejně

$\Leftarrow$ ohodnocení zkonstruované výše je model φ:

- jednotková klauzule ℓ platí kvůli hraně $\overline{\ell} \to \ell$, komponenta s $\overline{\ell}$ byla ohodnocena dříve, a to 0, takže $v(\ell) = 1$
- podobně pro 2-klauzuli $\ell_1 \vee \ell_2$, máme hrany $\overline{\ell_1} \to \ell_2$, $\overline{\ell_2} \to \ell_1$ pokud jsme ℓ_1 ohodnotili dříve než ℓ_2, museli jsme jako první narazit na komponentu s $\overline{\ell_1}$ a ohodnotit ji 0, tedy ℓ_1 platí; v opačném případě symetricky platí ℓ_2 $\square$

Shrnutí

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

Důkaz: $\Rightarrow$ literály v komponentě musí být ohodnoceny stejně

$\Leftarrow$ ohodnocení zkonstruované výše je model φ:

- jednotková klauzule ℓ platí kvůli hraně $\overline{\ell} \to \ell$, komponenta s $\overline{\ell}$ byla ohodnocena dříve, a to 0, takže $v(\ell) = 1$
- podobně pro 2-klauzuli $\ell_1 \vee \ell_2$, máme hrany $\overline{\ell_1} \to \ell_2$, $\overline{\ell_2} \to \ell_1$ pokud jsme ℓ_1 ohodnotili dříve než ℓ_2, museli jsme jako první narazit na komponentu s $\overline{\ell_1}$ a ohodnotit ji 0, tedy ℓ_1 platí; v opačném případě symetricky platí ℓ_2 □

Důsledek: 2-SAT je řešitelný v lineárním čase, včetně konstrukce modelu (pokud existuje).

Shrnutí

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

Důkaz: $\Rightarrow$ literály v komponentě musí být ohodnoceny stejně

$\Leftarrow$ ohodnocení zkonstruované výše je model φ:

- jednotková klauzule ℓ platí kvůli hraně $\overline{\ell} \to \ell$, komponenta s $\overline{\ell}$ byla ohodnocena dříve, a to 0, takže $v(\ell) = 1$
- podobně pro 2-klauzuli $\ell_1 \vee \ell_2$, máme hrany $\overline{\ell_1} \to \ell_2$, $\overline{\ell_2} \to \ell_1$ pokud jsme ℓ_1 ohodnotili dříve než ℓ_2, museli jsme jako první narazit na komponentu s $\overline{\ell_1}$ a ohodnotit ji 0, tedy ℓ_1 platí; v opačném případě symetricky platí ℓ_2 □

Důsledek: 2-SAT je řešitelný v lineárním čase, včetně konstrukce modelu (pokud existuje).

Důkaz: Komponenty silné souvislosti i topologické uspořádání najdeme v čase $\mathcal{O}(|V| + |E|)$, stačí je projít jednou □

3.3 Horn-SAT a jednotková propagace

Horn-SAT

- hornovská klauzule: *nejvýše jeden *pozitivní* literál*

Horn-SAT

- hornovská klauzule: *nejvýše jeden *pozitivní* literál*

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

Horn-SAT

- hornovská klauzule: *nejvýše jeden *pozitivní* literál*

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

základ logického programování (Prolog `q:-p1,p2,...,pn.`)

Horn-SAT

- hornovská klauzule: *nejvýše jeden *pozitivní* literál*

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

  základ logického programování (Prolog `q:-p1,p2,...,pn.`)
- Horn-SAT, tj. splnitelnost hornovského výroku (konjunkce hornovských klauzulí) je opět v P, v lineárním čase

Horn-SAT

- hornovská klauzule: *nejvýše jeden *pozitivní* literál*

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

  základ logického programování (Prolog `q:-p1,p2,...,pn.`)
- Horn-SAT, tj. splnitelnost hornovského výroku (konjunkce hornovských klauzulí) je opět v P, v lineárním čase
- algoritmus využívá tzv. jednotkovou propagaci:
  - jednotková klauzule vynucuje hodnotu výrokové proměnné

Horn-SAT

- hornovská klauzule: *nejvýše jeden *pozitivní* literál*

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

  základ logického programování (Prolog `q:-p1,p2,...,pn.`)
- Horn-SAT, tj. splnitelnost hornovského výroku (konjunkce hornovských klauzulí) je opět v P, v lineárním čase
- algoritmus využívá tzv. jednotkovou propagaci:
  - jednotková klauzule vynucuje hodnotu výrokové proměnné
  - tím můžeme výrok zjednodušit, např. pro $\neg p$ ($p = 0$):
    odstraníme klauzule s literálem $\neg p$, už jsou splněné
    odstraníme literál p (nemůže být splněný)

Horn-SAT

- hornovská klauzule: *nejvýše jeden *pozitivní* literál*

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

  základ logického programování (Prolog `q:-p1,p2,...,pn.`)
- Horn-SAT, tj. splnitelnost hornovského výroku (konjunkce hornovských klauzulí) je opět v P, v lineárním čase
- algoritmus využívá tzv. jednotkovou propagaci:
  - jednotková klauzule vynucuje hodnotu výrokové proměnné
  - tím můžeme výrok zjednodušit, např. pro $\neg p$ ($p = 0$): odstraníme klauzule s literálem $\neg p$, už jsou splněné odstraníme literál p (nemůže být splněný)
  - žádná jednotková klauzule $\Rightarrow$ každá klauzule má aspoň jeden negativní literál $\Rightarrow$ vše nastavíme na 0

Jednotková propagace

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

Jednotková propagace

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

- nastav $v(p_4) = 1$, odstraň klauzule obsahující literál p_4, z ostatních klauzulí odstraň $\neg p_4$

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

Jednotková propagace

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

- nastav $v(p_4) = 1$, odstraň klauzule obsahující literál p_4, z ostatních klauzulí odstraň $\neg p_4$

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

- nastav $v(p_5) = 0$, proveď jednotkovou propagaci $\neg p_5$

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

Jednotková propagace

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

- nastav $v(p_4) = 1$, odstraň klauzule obsahující literál p_4, z ostatních klauzulí odstraň $\neg p_4$

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

- nastav $v(p_5) = 0$, proveď jednotkovou propagaci $\neg p_5$

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

- už žádná jednotková klauzule, v každé klauzuli alespoň dva literály ale nejvýše jeden pozitivní, tj. alespoň jeden negativní: $v(p_1) = v(p_2) = v(p_3) = 0$, model $v = (0, 0, 0, 1, 0)$

Jednotková propagace

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

- nastav $v(p_4) = 1$, odstraň klauzule obsahující literál p_4, z ostatních klauzulí odstraň $\neg p_4$

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

- nastav $v(p_5) = 0$, proveď jednotkovou propagaci $\neg p_5$

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

- už žádná jednotková klauzule, v každé klauzuli alespoň dva literály ale nejvýše jeden pozitivní, tj. alespoň jeden negativní: $v(p_1) = v(p_2) = v(p_3) = 0$, model $v = (0, 0, 0, 1, 0)$

Jednotková propagace

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

- nastav $v(p_4) = 1$, odstraň klauzule obsahující literál p_4, z ostatních klauzulí odstraň $\neg p_4$

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

- nastav $v(p_5) = 0$, proveď jednotkovou propagaci $\neg p_5$

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

- už žádná jednotková klauzule, v každé klauzuli alespoň dva literály ale nejvýše jeden pozitivní, tj. alespoň jeden negativní: $v(p_1) = v(p_2) = v(p_3) = 0$, model $v = (0, 0, 0, 1, 0)$

$$\varphi^{\ell} = \{C \setminus \{\bar{\ell}\} \mid C \in \varphi, \ell \notin C\} \qquad \text{(množinový zápis)}$$

Jednotková propagace

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

- nastav $v(p_4) = 1$, odstraň klauzule obsahující literál p_4, z ostatních klauzulí odstraň $\neg p_4$

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

- nastav $v(p_5) = 0$, proveď jednotkovou propagaci $\neg p_5$

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

- už žádná jednotková klauzule, v každé klauzuli alespoň dva literály ale nejvýše jeden pozitivní, tj. alespoň jeden negativní: $v(p_1) = v(p_2) = v(p_3) = 0$, model $v = (0, 0, 0, 1, 0)$

$$\varphi^{\ell} = \{C \setminus \{\bar{\ell}\} \mid C \in \varphi, \ell \notin C\} \qquad \text{(množinový zápis)}$$

Pozorování: φ^{ℓ} neobsahuje ℓ ani $\bar{\ell}$, modely $=$ modely φ splňující ℓ

Jednotková propagace

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

- nastav $v(p_4) = 1$, odstraň klauzule obsahující literál p_4, z ostatních klauzulí odstraň $\neg p_4$

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

- nastav $v(p_5) = 0$, proveď jednotkovou propagaci $\neg p_5$

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

- už žádná jednotková klauzule, v každé klauzuli alespoň dva literály ale nejvýše jeden pozitivní, tj. alespoň jeden negativní: $v(p_1) = v(p_2) = v(p_3) = 0$, model $v = (0, 0, 0, 1, 0)$

$$\varphi^{\ell} = \{C \setminus \{\bar{\ell}\} \mid C \in \varphi, \ell \notin C\} \qquad \text{(množinový zápis)}$$

Pozorování: φ^{ℓ} neobsahuje ℓ ani $\bar{\ell}$, modely $=$ modely φ splňující ℓ

$\psi = p \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge \neg r$ je nesplnitelný, co se stane?

Algoritmus pro Horn-SAT

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.

Algoritmus pro Horn-SAT

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: Algoritmus je korektní.

Důsledek: Horn-SAT lze řešit v lineárním čase.

Algoritmus pro Horn-SAT

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: Algoritmus je korektní.

Důsledek: Horn-SAT lze řešit v lineárním čase.

Důkaz: Korektnost plyne z pozorování a z diskuze. V každém kroku stačí projít, výrok zkrátíme (kvadratický horní odhad, ale při vhodné implementaci lineární) □

3.4 DPLL algoritmus pro řešení problému SAT

Algoritmus DPLL (Davis-Putnam-Logemann-Loveland, 1961)

myšlenka: čistý výskyt p buď jen v pozitivních nebo jen v negativních literálech $\Rightarrow$ lze mu nastavit příslušnou hodnotu!

Algoritmus DPLL (Davis-Putnam-Logemann-Loveland, 1961)

myšlenka: čistý výskyt p buď jen v pozitivních nebo jen v negativních literálech $\Rightarrow$ lze mu nastavit příslušnou hodnotu!

DPLL = jednotková propagace + čistý výskyt + větvení (rekurze)

Algoritmus DPLL (Davis-Putnam-Logemann-Loveland, 1961)

myšlenka: čistý výskyt p buď jen v pozitivních nebo jen v negativních literálech $\Rightarrow$ lze mu nastavit příslušnou hodnotu!

DPLL = jednotková propagace + čistý výskyt + větvení (rekurze)

vstup: výrok φ v CNF,

výstup: model φ nebo informace, že φ není splnitelný

Algoritmus DPLL (Davis-Putnam-Logemann-Loveland, 1961)

myšlenka: čistý výskyt p buď jen v pozitivních nebo jen v negativních literálech $\Rightarrow$ lze mu nastavit příslušnou hodnotu!

DPLL = jednotková propagace + čistý výskyt + větvení (rekurze)

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, nahraď φ výrokem φ^{ℓ}.

Algoritmus DPLL (Davis-Putnam-Logemann-Loveland, 1961)

myšlenka: čistý výskyt p buď jen v pozitivních nebo jen v negativních literálech $\Rightarrow$ lze mu nastavit příslušnou hodnotu!

DPLL = jednotková propagace + čistý výskyt + větvení (rekurze)

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, 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.

Algoritmus DPLL (Davis-Putnam-Logemann-Loveland, 1961)

myšlenka: čistý výskyt p buď jen v pozitivních nebo jen v negativních literálech $\Rightarrow$ lze mu nastavit příslušnou hodnotu!

DPLL = jednotková propagace + čistý výskyt + větvení (rekurze)

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, 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.

Algoritmus DPLL (Davis-Putnam-Logemann-Loveland, 1961)

myšlenka: čistý výskyt p buď jen v pozitivních nebo jen v negativních literálech $\Rightarrow$ lze mu nastavit příslušnou hodnotu!

DPLL = jednotková propagace + čistý výskyt + větvení (rekurze)

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, 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.

Algoritmus DPLL (Davis-Putnam-Logemann-Loveland, 1961)

myšlenka: čistý výskyt p buď jen v pozitivních nebo jen v negativních literálech $\Rightarrow$ lze mu nastavit příslušnou hodnotu!

DPLL = jednotková propagace + čistý výskyt + větvení (rekurze)

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, 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 $\varphi \wedge \neg p$.

Ukázkový běh

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

Ukázkový běh

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

žádná jednotková klauzule, $\neg r$ má čistý výskyt: nastav $v(r) = 0$ a odstraň klauzule obsahující $\neg r$:

Ukázkový běh

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

žádná jednotková klauzule, $\neg r$ má čistý výskyt: nastav $v(r) = 0$ a odstraň klauzule obsahující $\neg r$:

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

Ukázkový běh

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

žádná jednotková klauzule, $\neg r$ má čistý výskyt: nastav $v(r) = 0$ a odstraň klauzule obsahující $\neg r$:

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

už žádný čistý výskyt, rekurzivně zavolej na:

1. $(\neg p \vee \neg q \vee \neg s) \wedge (p \vee s) \wedge (p \vee \neg s) \wedge (q \vee s) \wedge p$
2. $(\neg p \vee \neg q \vee \neg s) \wedge (p \vee s) \wedge (p \vee \neg s) \wedge (q \vee s) \wedge \neg p$

Ukázkový běh

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

žádná jednotková klauzule, $\neg r$ má čistý výskyt: nastav $v(r) = 0$ a odstraň klauzule obsahující $\neg r$:

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

už žádný čistý výskyt, rekurzivně zavolej na:

1. $(\neg p \vee \neg q \vee \neg s) \wedge (p \vee s) \wedge (p \vee \neg s) \wedge (q \vee s) \wedge p$
2. $(\neg p \vee \neg q \vee \neg s) \wedge (p \vee s) \wedge (p \vee \neg s) \wedge (q \vee s) \wedge \neg p$

a pokračuj dále v obou větvích výpočtu

Ukázkový běh

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

žádná jednotková klauzule, $\neg r$ má čistý výskyt: nastav $v(r) = 0$ a odstraň klauzule obsahující $\neg r$:

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

už žádný čistý výskyt, rekurzivně zavolej na:

1. $(\neg p \vee \neg q \vee \neg s) \wedge (p \vee s) \wedge (p \vee \neg s) \wedge (q \vee s) \wedge p$
2. $(\neg p \vee \neg q \vee \neg s) \wedge (p \vee s) \wedge (p \vee \neg s) \wedge (q \vee s) \wedge \neg p$

a pokračuj dále v obou větvích výpočtu

$$\vdots$$

1. větev dává $(1, 0, 0, 1)$ a $(1, 1, 0, 0)$, 2. je sporná. Modelem je také $(1, 1, 1, 0)$, ten ztratíme nastavením $v(r) = 0$. Odstranění čistého výskytu zachová splnitelnost, ne všechny modely.

Kapitola 4: Metoda analytického tabla

4.1 Formální dokazovací systémy

Formální dokazovací systém

chceme zjistit, zda výrok platí [$T \models \varphi$], a to čistě syntakticky, aniž bychom se zabývali sémantikou: najít (formální) důkaz [$T \vdash \varphi$]

Formální dokazovací systém

chceme zjistit, zda výrok platí [$T \models \varphi$], a to čistě syntakticky, aniž bychom se zabývali sémantikou: najít (formální) důkaz [$T \vdash \varphi$]

důkaz je konečný syntaktický objekt vycházející z φ a axiomů T

Formální dokazovací systém

chceme zjistit, zda výrok platí [$T \models \varphi$], a to čistě syntakticky, aniž bychom se zabývali sémantikou: najít (formální) důkaz [$T \vdash \varphi$]

důkaz je konečný syntaktický objekt vycházející z φ a axiomů T

dokazování lze dělat algoritmicky (pokud máme algoritmický přístup k axiomům T, která může být nekonečná), a lze rychle algoritmicky ověřit, zda je daný objekt opravdu korektní důkaz

Formální dokazovací systém

chceme zjistit, zda výrok platí [$T \models \varphi$], a to čistě syntakticky, aniž bychom se zabývali sémantikou: najít (formální) důkaz [$T \vdash \varphi$]

důkaz je konečný syntaktický objekt vycházející z φ a axiomů T

dokazování lze dělat algoritmicky (pokud máme algoritmický přístup k axiomům T, která může být nekonečná), a lze rychle algoritmicky ověřit, zda je daný objekt opravdu korektní důkaz

- korektnost: "co dokážu, platí" $T \vdash \varphi \Rightarrow T \models \varphi$
- úplnost: "dokážu vše, co platí" $T \models \varphi \Rightarrow T \vdash \varphi$

Formální dokazovací systém

chceme zjistit, zda výrok platí [$T \models \varphi$], a to čistě syntakticky, aniž bychom se zabývali sémantikou: najít (formální) důkaz [$T \vdash \varphi$]

důkaz je konečný syntaktický objekt vycházející z φ a axiomů T

dokazování lze dělat algoritmicky (pokud máme algoritmický přístup k axiomům T, která může být nekonečná), a lze rychle algoritmicky ověřit, zda je daný objekt opravdu korektní důkaz

- korektnost: "co dokážu, platí" $T \vdash \varphi \Rightarrow T \models \varphi$
- úplnost: "dokážu vše, co platí" $T \models \varphi \Rightarrow T \vdash \varphi$

(korektnost je nutná, úplnost ne: rychlý dokazovací systém může být praktický i když není úplný)

Formální dokazovací systém

chceme zjistit, zda výrok platí [$T \models \varphi$], a to čistě syntakticky, aniž bychom se zabývali sémantikou: najít (formální) důkaz [$T \vdash \varphi$]

důkaz je konečný syntaktický objekt vycházející z φ a axiomů T

dokazování lze dělat algoritmicky (pokud máme algoritmický přístup k axiomům T, která může být nekonečná), a lze rychle algoritmicky ověřit, zda je daný objekt opravdu korektní důkaz

- korektnost: "co dokážu, platí" $T \vdash \varphi \Rightarrow T \models \varphi$
- úplnost: "dokážu vše, co platí" $T \models \varphi \Rightarrow T \vdash \varphi$

(korektnost je nutná, úplnost ne: rychlý dokazovací systém může být praktický i když není úplný)

ukážeme si: *tablo metodu*, *hilbertovský kalkulus*, *rezoluční metodu*

Formální dokazovací systém

chceme zjistit, zda výrok platí [$T \models \varphi$], a to čistě syntakticky, aniž bychom se zabývali sémantikou: najít (formální) důkaz [$T \vdash \varphi$]

důkaz je konečný syntaktický objekt vycházející z φ a axiomů T

dokazování lze dělat algoritmicky (pokud máme algoritmický přístup k axiomům T, která může být nekonečná), a lze rychle algoritmicky ověřit, zda je daný objekt opravdu korektní důkaz

- korektnost: "co dokážu, platí" $T \vdash \varphi \Rightarrow T \models \varphi$
- úplnost: "dokážu vše, co platí" $T \models \varphi \Rightarrow T \vdash \varphi$

(korektnost je nutná, úplnost ne: rychlý dokazovací systém může být praktický i když není úplný)

ukážeme si: *tablo metodu*, *hilbertovský kalkulus*, *rezoluční metodu*

nutný předpoklad: jazyk musí být spočetný (potom i T je spočetná)

4.2 Úvod do tablo metody

Tablo metoda neformálně

nejprve případ $T = \emptyset$, tedy dokazujeme, že φ platí *v logice*

Tablo metoda neformálně

nejprve případ $T = \emptyset$, tedy dokazujeme, že φ platí *v logice*

tablo je strom představující hledání protipříkladu (modelu $v \not\models \varphi$),

Tablo metoda neformálně

nejprve případ $T = \emptyset$, tedy dokazujeme, že φ platí *v logice*

tablo je strom představující hledání protipříkladu (modelu $v \not\models \varphi$), když všechny větve selžou, máme důkaz (sporem)

Tablo metoda neformálně

nejprve případ $T = \emptyset$, tedy dokazujeme, že φ platí *v logice*

tablo je strom představující hledání protipříkladu (modelu $v \not\models \varphi$), když všechny větve selžou, máme důkaz (sporem)

labely: položky $\mathrm{T}\psi, \mathrm{F}\psi$ (určují, zda na dané větvi platí výrok ψ)

Tablo metoda neformálně

nejprve případ $T = \emptyset$, tedy dokazujeme, že φ platí *v logice*

tablo je strom představující hledání protipříkladu (modelu $v \not\models \varphi$), když všechny větve selžou, máme důkaz (sporem)

labely: položky $\mathrm{T}\psi, \mathrm{F}\psi$ (určují, zda na dané větvi platí výrok ψ)

kořen $\mathrm{F}\varphi$, dále rozvíjíme redukcí položek (podle struktury výroků v nich), aby platil invariant:

Tablo metoda neformálně

nejprve případ $T = \emptyset$, tedy dokazujeme, že φ platí *v logice*

tablo je strom představující hledání protipříkladu (modelu $v \not\models \varphi$), když všechny větve selžou, máme důkaz (sporem)

labely: položky $\mathrm{T}\psi, \mathrm{F}\psi$ (určují, zda na dané větvi platí výrok ψ)

kořen $\mathrm{F}\varphi$, dále rozvíjíme redukcí položek (podle struktury výroků v nich), aby platil invariant:

Každý model, který se *shoduje* s položkou v kořeni (tj. ve kterém neplatí φ), se musí *shodovat* i s některou větví tabla (tj. splňovat všechny požadavky vyjádřené položkami na této větvi).

Tablo metoda neformálně

nejprve případ $T = \emptyset$, tedy dokazujeme, že φ platí *v logice*

tablo je strom představující hledání protipříkladu (modelu $v \not\models \varphi$), když všechny větve selžou, máme důkaz (sporem)

labely: položky $\mathrm{T}\psi, \mathrm{F}\psi$ (určují, zda na dané větvi platí výrok ψ)

kořen $\mathrm{F}\varphi$, dále rozvíjíme redukcí položek (podle struktury výroků v nich), aby platil invariant:

Každý model, který se *shoduje* s položkou v kořeni (tj. ve kterém neplatí φ), se musí *shodovat* i s některou větví tabla (tj. splňovat všechny požadavky vyjádřené položkami na této větvi).

je-li na větvi $\mathrm{T}\psi$ a zároveň $\mathrm{F}\psi$, potom selhala (je sporná), pokud všechny větve selhaly, je tablo sporné, je to důkaz $T \vdash \varphi$

Tablo metoda neformálně

nejprve případ $T = \emptyset$, tedy dokazujeme, že φ platí *v logice*

tablo je strom představující hledání protipříkladu (modelu $v \not\models \varphi$), když všechny větve selžou, máme důkaz (sporem)

labely: položky $\mathrm{T}\psi, \mathrm{F}\psi$ (určují, zda na dané větvi platí výrok ψ)

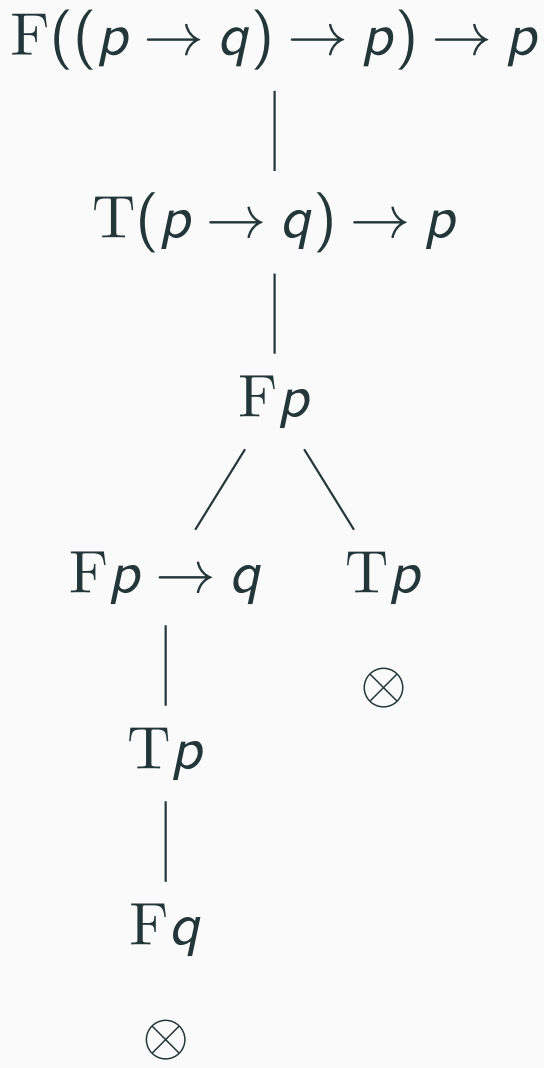
kořen $\mathrm{F}\varphi$, dále rozvíjíme redukcí položek (podle struktury výroků v nich), aby platil invariant:

Každý model, který se *shoduje* s položkou v kořeni (tj. ve kterém neplatí φ), se musí *shodovat* i s některou větví tabla (tj. splňovat všechny požadavky vyjádřené položkami na této větvi).

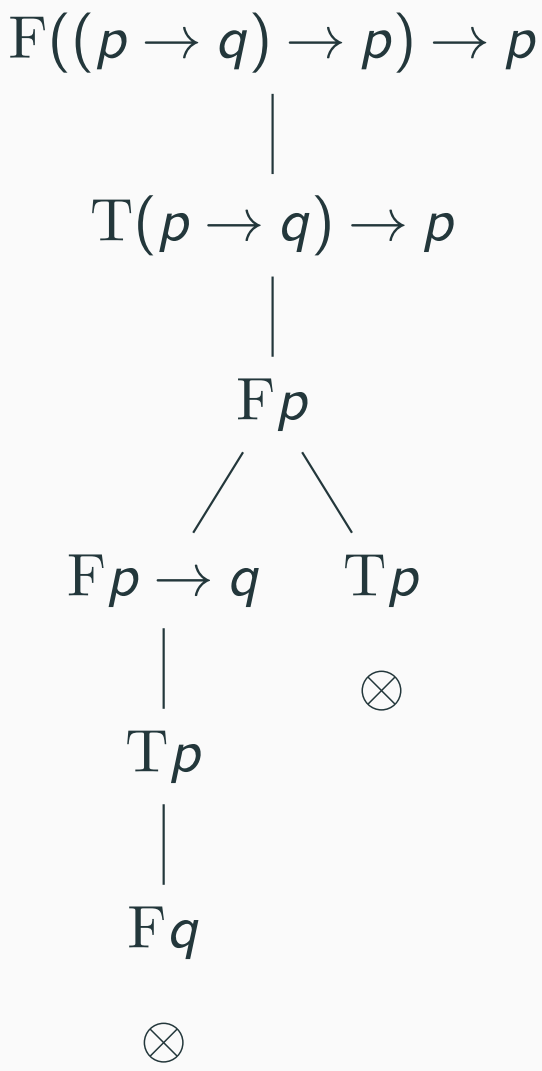
je-li na větvi $\mathrm{T}\psi$ a zároveň $\mathrm{F}\psi$, potom selhala (je sporná), pokud všechny větve selhaly, je tablo sporné, je to důkaz $T \vdash \varphi$

pokud nějaká větev neselhala a je dokončená (vše na ní zredukované), lze z ní zkonstruovat model, ve kterém φ neplatí

Příklad: tablo důkaz $((p \to q) \to p) \to p$

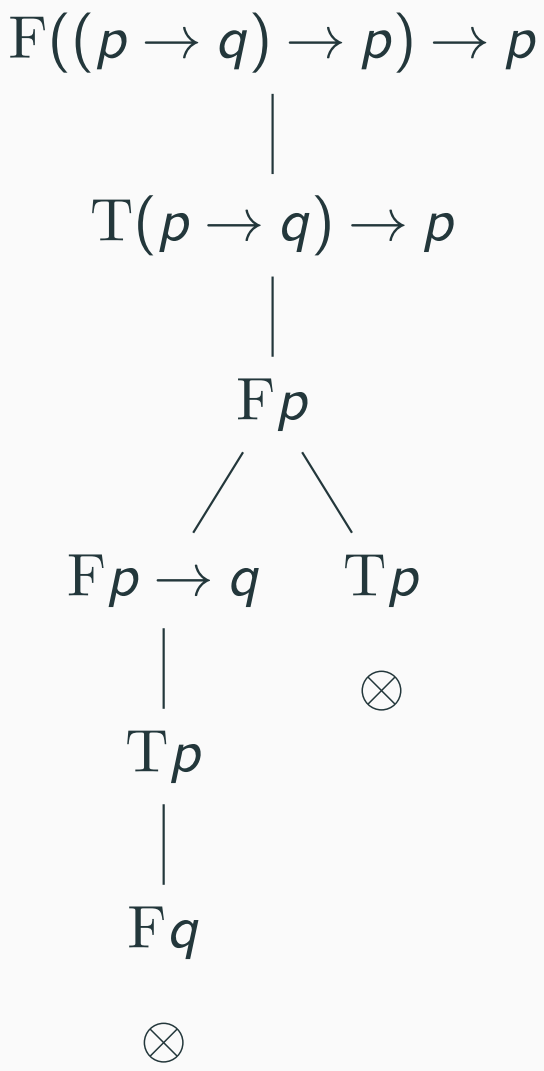


Příklad: tablo důkaz $((p \to q) \to p) \to p$



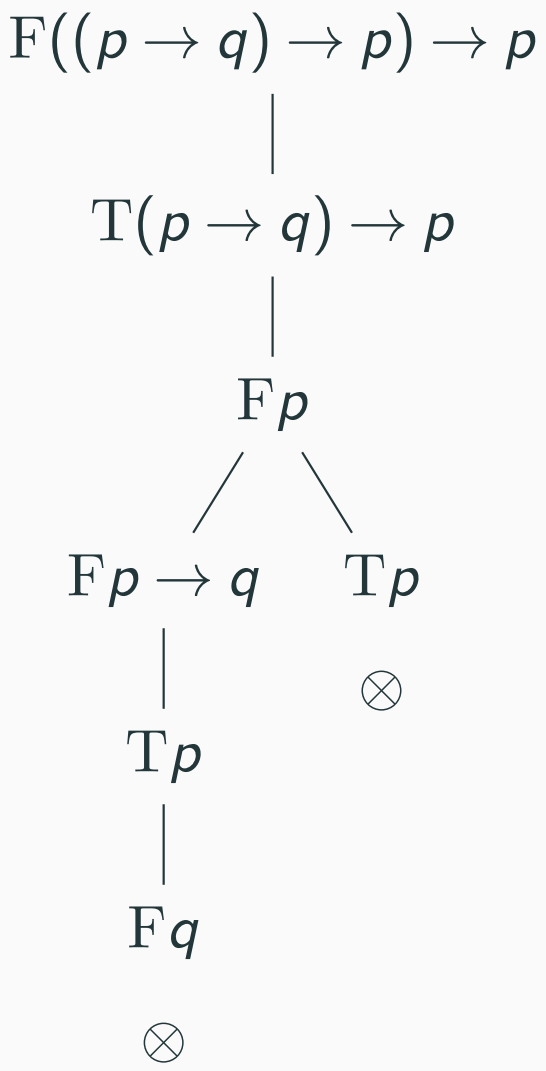
- důkaz sporem: v kořeni příznak F

Příklad: tablo důkaz $((p \to q) \to p) \to p$



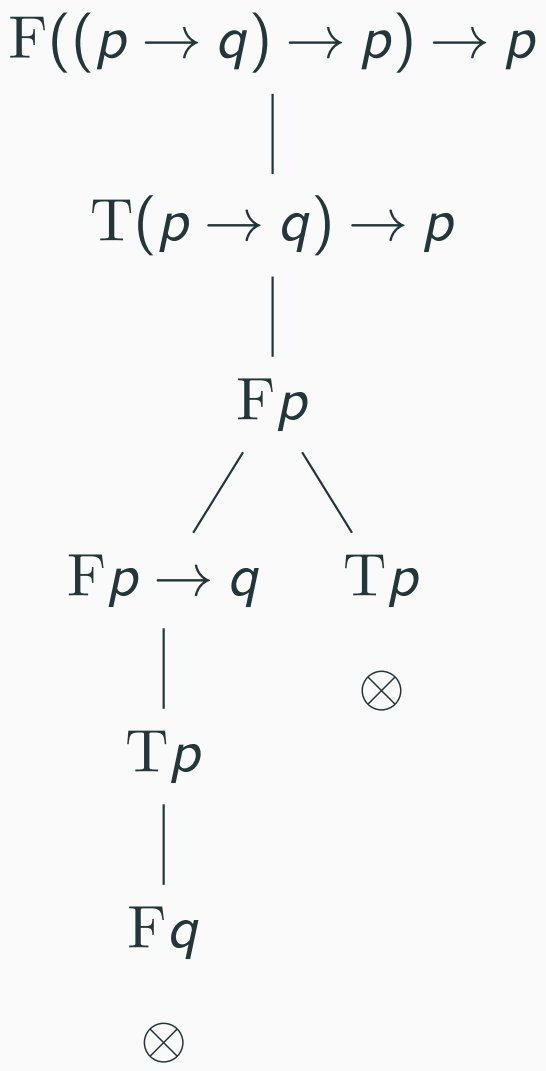
- důkaz sporem: v kořeni příznak F
- redukujeme položku tvaru $\mathrm{F}\varphi_1 \to \varphi_2$:

Příklad: tablo důkaz $((p \to q) \to p) \to p$



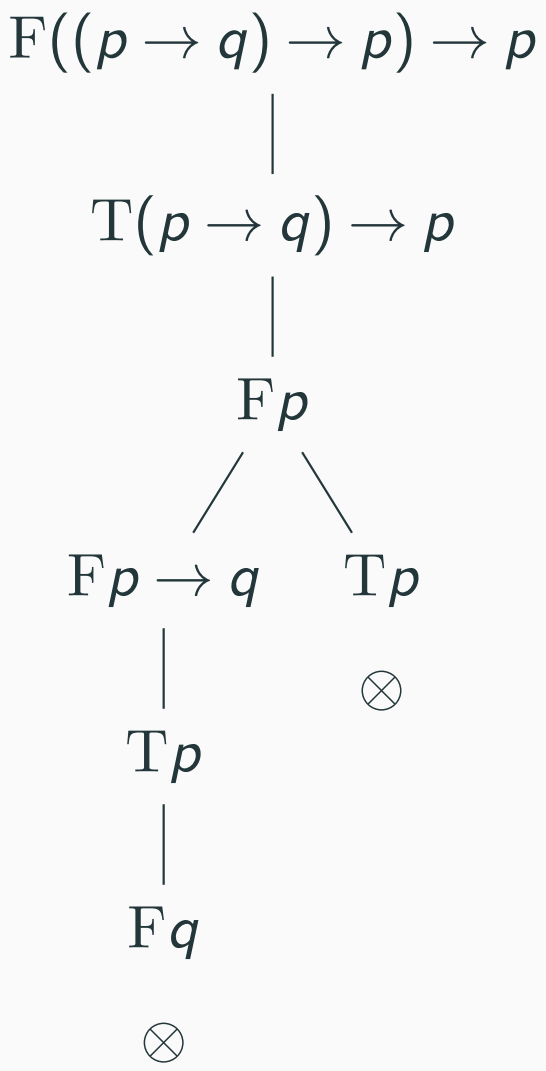
- důkaz sporem: v kořeni příznak F
- redukujeme položku tvaru $\mathrm{F}\varphi_1 \to \varphi_2$:
- pokud $v \not\models \varphi_1 \to \varphi_2$, nutně $v \models \varphi_1$ a zároveň $v \not\models \varphi_2$

Příklad: tablo důkaz $((p \to q) \to p) \to p$



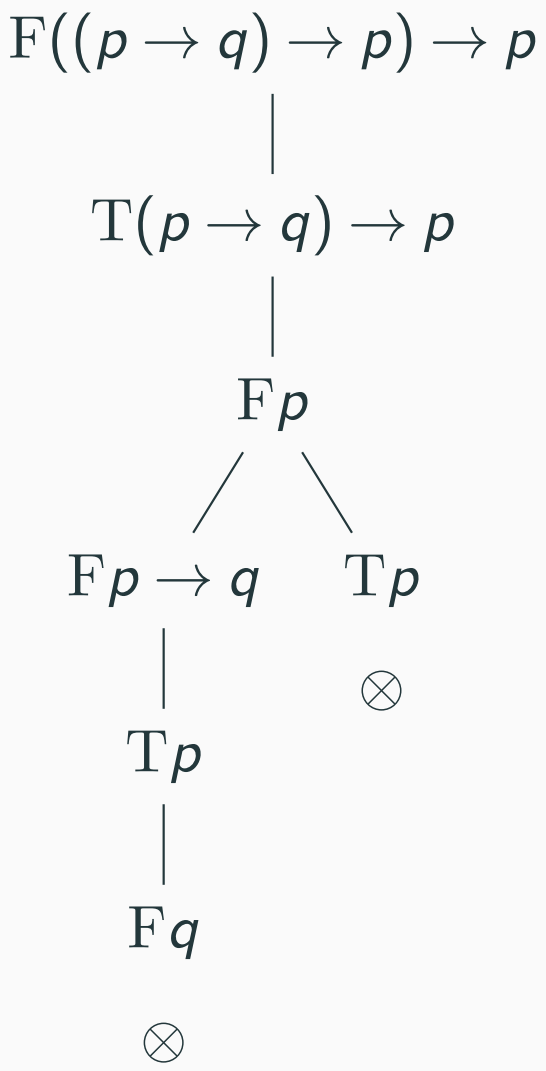
- důkaz sporem: v kořeni příznak F
- redukujeme položku tvaru $\mathrm{F}\varphi_1 \to \varphi_2$:
- pokud $v \not\models \varphi_1 \to \varphi_2$, nutně $v \models \varphi_1$ a zároveň $v \not\models \varphi_2$
- proto na větev připojíme položky $\mathrm{T}(p \to q) \to p$ a $\mathrm{F}p$, invariant platí

Příklad: tablo důkaz $((p \to q) \to p) \to p$



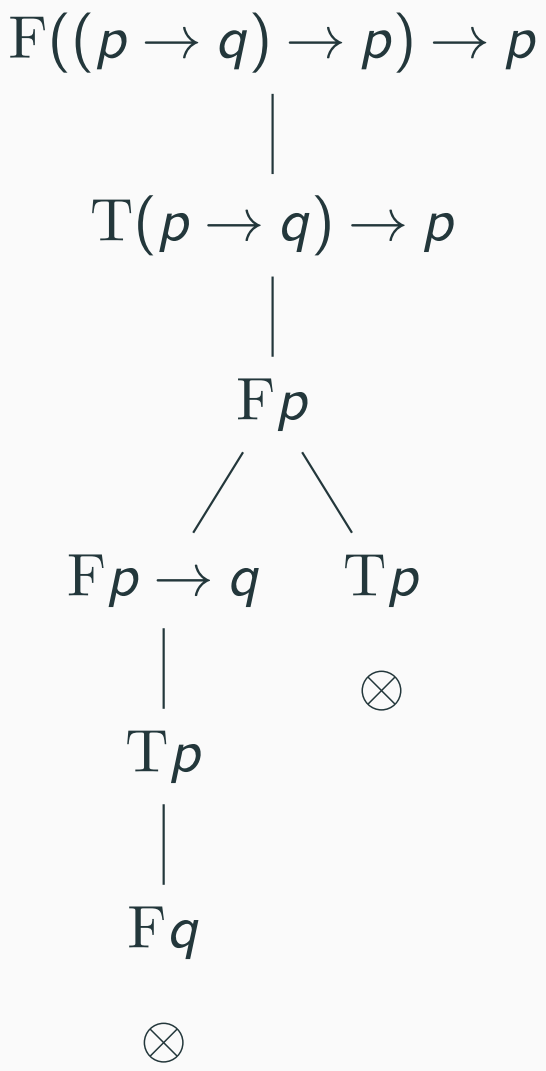
- důkaz sporem: v kořeni příznak F
- redukujeme položku tvaru $\mathrm{F}\varphi_1 \to \varphi_2$:
- pokud $v \not\models \varphi_1 \to \varphi_2$, nutně $v \models \varphi_1$ a zároveň $v \not\models \varphi_2$
- proto na větev připojíme položky $\mathrm{T}(p \to q) \to p$ a $\mathrm{F}p$, invariant platí
- redukce položky $\mathrm{T}(p \to q) \to p$: model se shoduje s $\mathrm{F}(p \to q)$ nebo s $\mathrm{T}p$, rozvětvi!

Příklad: tablo důkaz $((p \to q) \to p) \to p$



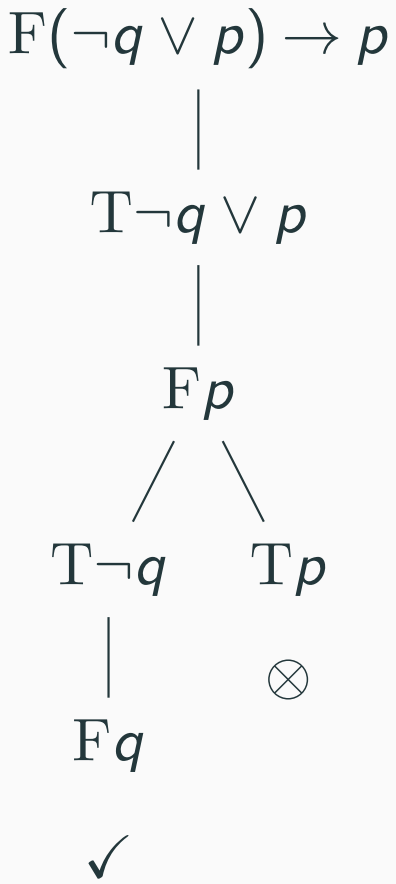
- důkaz sporem: v kořeni příznak F
- redukujeme položku tvaru $\mathrm{F}\varphi_1 \to \varphi_2$:
- pokud $v \not\models \varphi_1 \to \varphi_2$, nutně $v \models \varphi_1$ a zároveň $v \not\models \varphi_2$
- proto na větev připojíme položky $\mathrm{T}(p \to q) \to p$ a $\mathrm{F}p$, invariant platí
- redukce položky $\mathrm{T}(p \to q) \to p$: model se shoduje s $\mathrm{F}(p \to q)$ nebo s $\mathrm{T}p$, rozvětvi!
- redukce $\mathrm{F}(p \to q)$: připoj $\mathrm{T}p$ a $\mathrm{F}q$

Příklad: tablo důkaz $((p \to q) \to p) \to p$

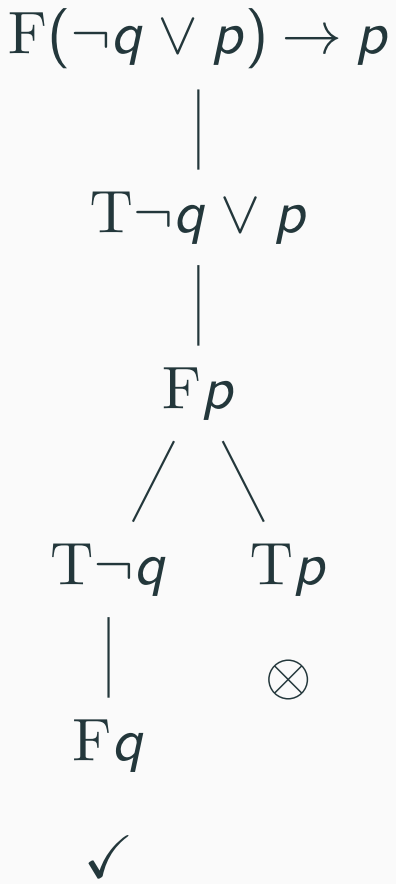


- důkaz sporem: v kořeni příznak F
- redukujeme položku tvaru $\mathrm{F}\varphi_1 \to \varphi_2$:
- pokud $v \not\models \varphi_1 \to \varphi_2$, nutně $v \models \varphi_1$ a zároveň $v \not\models \varphi_2$
- proto na větev připojíme položky $\mathrm{T}(p \to q) \to p$ a $\mathrm{F}p$, invariant platí
- redukce položky $\mathrm{T}(p \to q) \to p$: model se shoduje s $\mathrm{F}(p \to q)$ nebo s $\mathrm{T}p$, rozvětvi!
- redukce $\mathrm{F}(p \to q)$: připoj $\mathrm{T}p$ a $\mathrm{F}q$
- všechny větve sporné, protipříklad neexistuje, tedy máme tablo důkaz, píšeme: $\vdash ((p \to q) \to p) \to p$

Příklad: tablo pro $\mathrm{F}(\neg q \vee p) \to p$

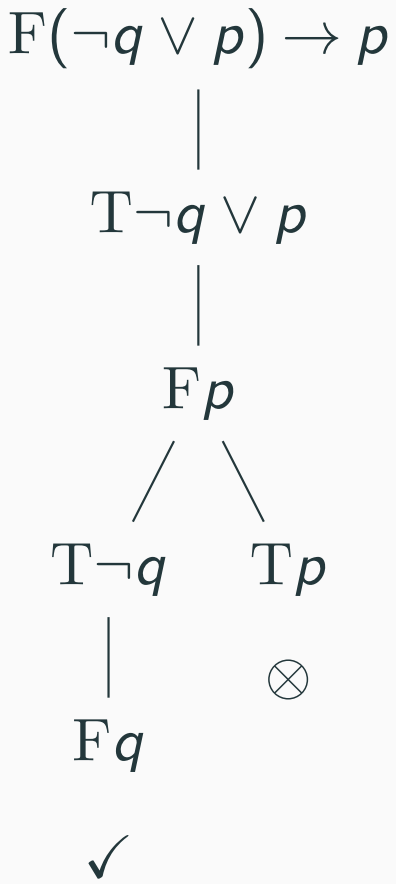


Příklad: tablo pro $\mathrm{F}(\neg q \vee p) \to p$



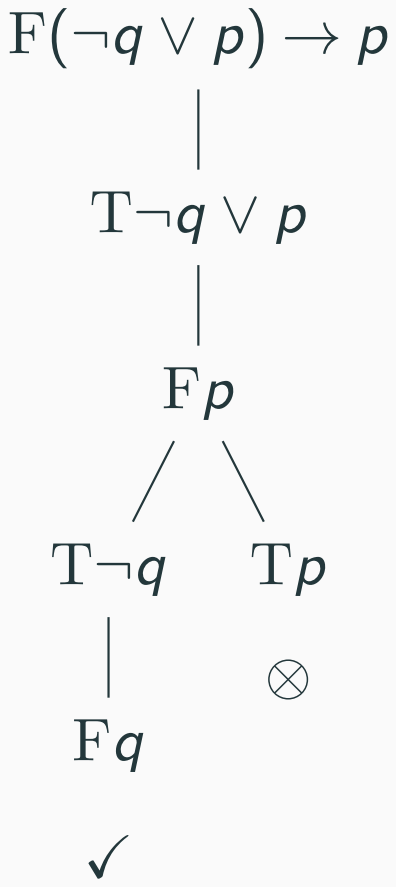
- tablo je dokončené, ale není sporné

Příklad: tablo pro $\mathrm{F}(\neg q \vee p) \to p$



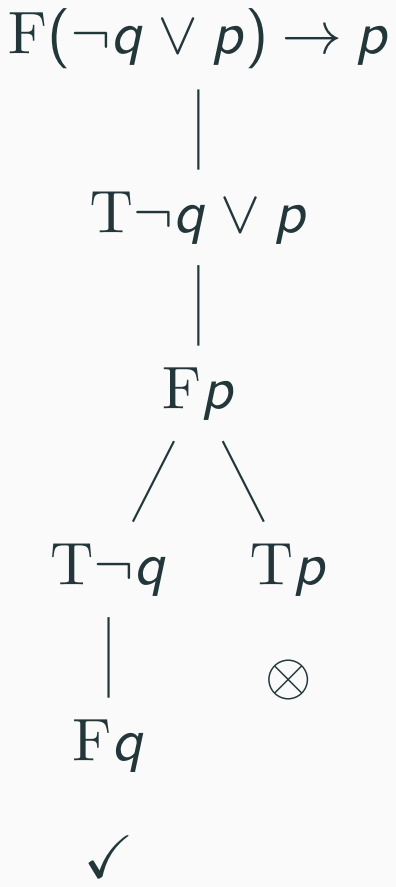
- tablo je dokončené, ale není sporné
- tedy nejde o důkaz

Příklad: tablo pro $\mathrm{F}(\neg q \vee p) \to p$



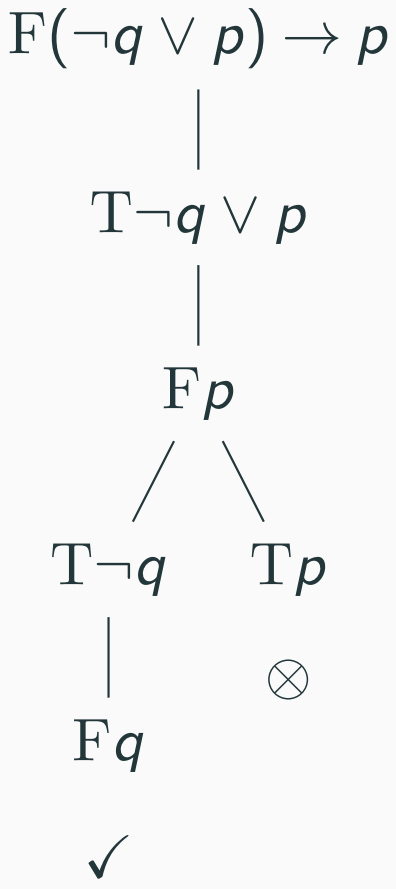
- tablo je dokončené, ale není sporné
- tedy nejde o důkaz
- levá větev dává protipříklad: model $v = (0, 0)$ ve kterém výrok neplatí

Příklad: tablo pro $\mathrm{F}(\neg q \vee p) \to p$



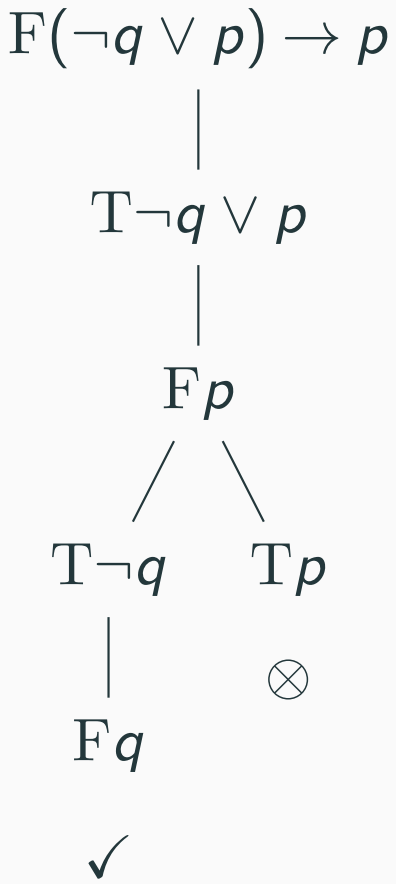
- tablo je dokončené, ale není sporné
- tedy nejde o důkaz
- levá větev dává protipříklad: model $v = (0, 0)$ ve kterém výrok neplatí
- invariant říká, že existuje-li protipříklad, shoduje se s některou větví

Příklad: tablo pro $\mathrm{F}(\neg q \vee p) \to p$



- tablo je dokončené, ale není sporné
- tedy nejde o důkaz
- levá větev dává protipříklad: model $v = (0, 0)$ ve kterém výrok neplatí
- invariant říká, že existuje-li protipříklad, shoduje se s některou větví
- tato větev nemůže být sporná

Příklad: tablo pro $\mathrm{F}(\neg q \vee p) \to p$



- tablo je dokončené, ale není sporné
- tedy nejde o důkaz
- levá větev dává protipříklad: model $v = (0, 0)$ ve kterém výrok neplatí
- invariant říká, že existuje-li protipříklad, shoduje se s některou větví
- tato větev nemůže být sporná
- tak se dokáže korektnost tablo metody

Poznámky

- Jak redukujeme položky?

Poznámky

- Jak redukujeme položky?
  - Připojíme příslušné atomické tablo (viz následující slide) na konec všech bezesporných větví procházejících vrcholem.

Poznámky

- Jak redukujeme položky?
  - Připojíme příslušné atomické tablo (viz následující slide) na konec všech bezesporných větví procházejících vrcholem.
- Co když dokazujeme v nějaké teorii T?

Poznámky

- Jak redukujeme položky?
  - Připojíme příslušné atomické tablo (viz následující slide) na konec všech bezesporných větví procházejících vrcholem.
- Co když dokazujeme v nějaké teorii T?
  - Připojíme položky $\mathrm{T}\alpha$ pro (všechny) axiomy $\alpha \in T$.

Poznámky

- Jak redukujeme položky?
  - Připojíme příslušné atomické tablo (viz následující slide) na konec všech bezesporných větví procházejících vrcholem.
- Co když dokazujeme v nějaké teorii T?
  - Připojíme položky $\mathrm{T}\alpha$ pro (všechny) axiomy $\alpha \in T$.
- Co když je T nekonečná?

Poznámky

- Jak redukujeme položky?
  - Připojíme příslušné atomické tablo (viz následující slide) na konec všech bezesporných větví procházejících vrcholem.
- Co když dokazujeme v nějaké teorii T?
  - Připojíme položky $\mathrm{T}\alpha$ pro (všechny) axiomy $\alpha \in T$.
- Co když je T nekonečná?
  - Tablo může být nekonečné.

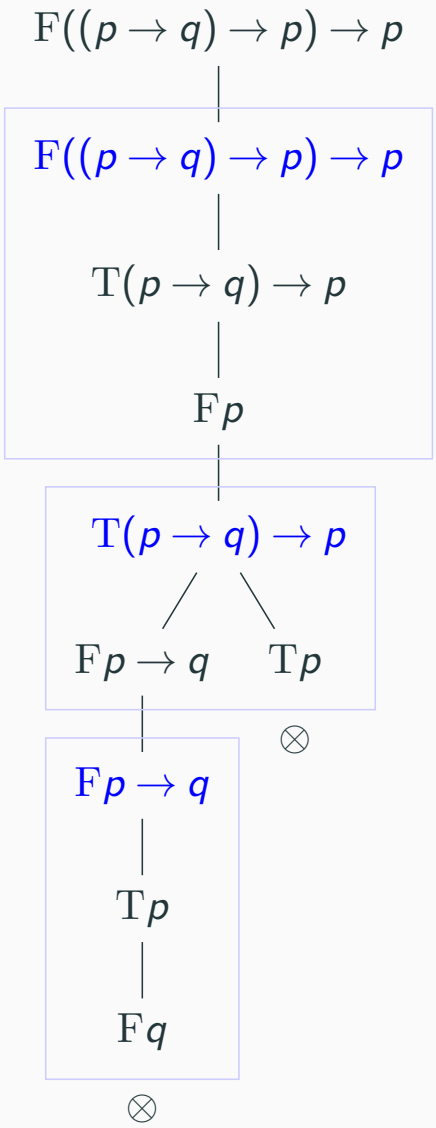
Poznámky

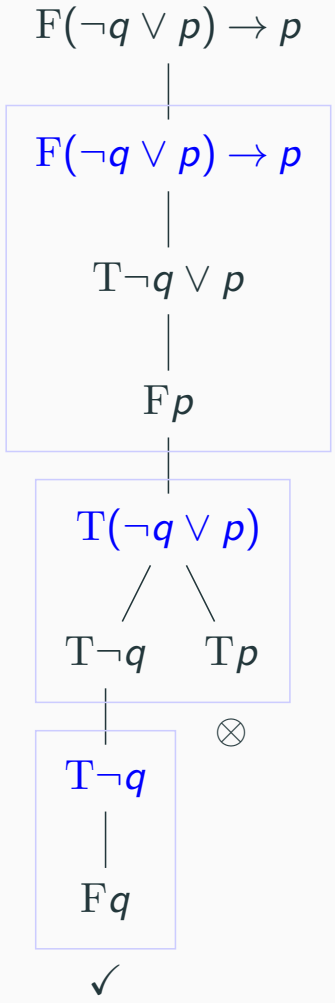
- Jak redukujeme položky?
  - Připojíme příslušné atomické tablo (viz následující slide) na konec všech bezesporných větví procházejících vrcholem.
- Co když dokazujeme v nějaké teorii T?
  - Připojíme položky $\mathrm{T}\alpha$ pro (všechny) axiomy $\alpha \in T$.
- Co když je T nekonečná?
  - Tablo může být nekonečné.
  - Ale vyjde-li sporné, lze sestrojit jiné, které je konečné a také sporné. (“Existuje-li důkaz, existuje konečný důkaz.”)

Atomická tabla

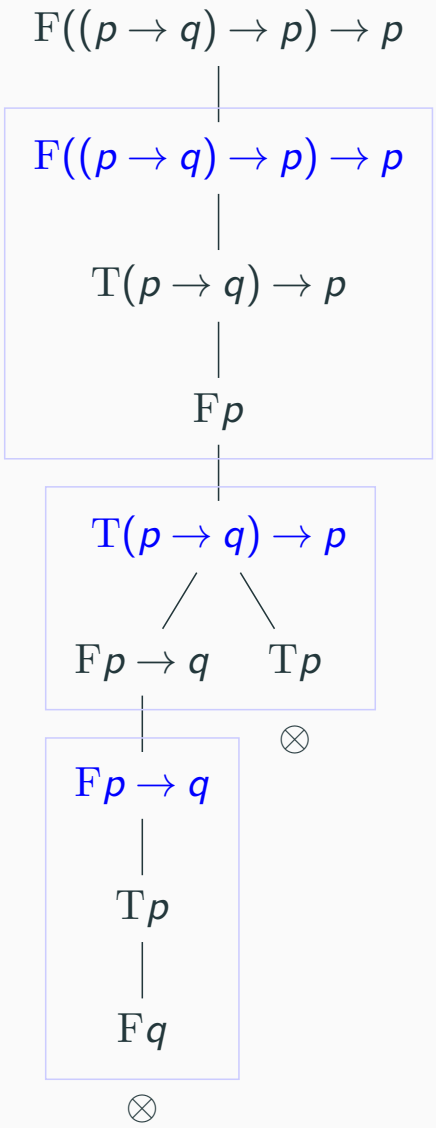
	$\neg$	$\wedge$	$\vee$	$\rightarrow$	$\leftrightarrow$
True	$\mathrm{T}\neg\varphi$ | $\mathrm{F}\varphi$	$\mathrm{T}\varphi \wedge \psi$ | $\mathrm{T}\varphi$ | $\mathrm{T}\psi$	$\mathrm{T}\varphi \vee \psi$ / \\ $\mathrm{T}\varphi$ $\mathrm{T}\psi$	$\mathrm{T}\varphi \rightarrow \psi$ / \\ $\mathrm{F}\varphi$ $\mathrm{T}\psi$	$\mathrm{T}\varphi \leftrightarrow \psi$ / \\ $\mathrm{T}\varphi$ $\mathrm{F}\varphi$ | | $\mathrm{T}\psi$ $\mathrm{F}\psi$
False	$\mathrm{F}\neg\varphi$ | $\mathrm{T}\varphi$	$\mathrm{F}\varphi \wedge \psi$ / \\ $\mathrm{F}\varphi$ $\mathrm{F}\psi$	$\mathrm{F}\varphi \vee \psi$ | $\mathrm{F}\varphi$ | $\mathrm{F}\psi$	$\mathrm{F}\varphi \rightarrow \psi$ | $\mathrm{T}\varphi$ | $\mathrm{F}\psi$	$\mathrm{F}\varphi \leftrightarrow \psi$ / \\ $\mathrm{T}\varphi$ $\mathrm{F}\varphi$ | | $\mathrm{F}\psi$ $\mathrm{T}\psi$

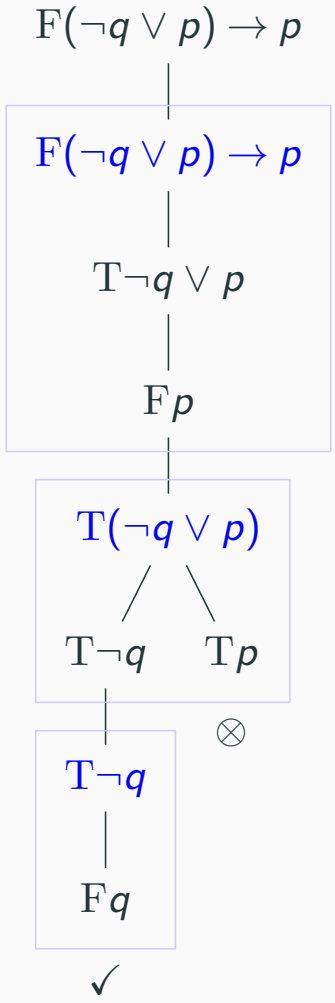
Konstrukce tabel z příkladů





Konstrukce tabel z příkladů





konvence: kořeny atomických tabel (modře) nezakreslujeme

O stromech

- strom je $T \neq \emptyset$ s částečným uspořádáním $<_T$, které má nejmenší prvek (kořen) a množina předků libovolného vrcholu je dobře uspořádaná

O stromech

- strom je $T \neq \emptyset$ s částečným uspořádáním $<_T$, které má nejmenší prvek (kořen) a množina předků libovolného vrcholu je dobře uspořádaná (každá její neprázdná podmnožina má nejmenší prvek, to zakáže nekonečné klesající řetězce předků)

O stromech

- strom je $T \neq \emptyset$ s částečným uspořádáním $<_T$, které má nejmenší prvek (kořen) a množina předků libovolného vrcholu je dobře uspořádaná (každá její neprázdná podmnožina má nejmenší prvek, to zakáže nekonečné klesající řetězce předků)
- větev je maximální lineárně uspořádaná podmnožina T.

O stromech

- strom je $T \neq \emptyset$ s částečným uspořádáním $<_T$, které má nejmenší prvek (kořen) a množina předků libovolného vrcholu je dobře uspořádaná (každá její neprázdná podmnožina má nejmenší prvek, to zakáže nekonečné klesající řetězce předků)
- větev je maximální lineárně uspořádaná podmnožina T.
- uspořádaný strom má navíc lineární uspořádání $<_L$ množiny synů každého vrcholu (říkáme mu pravolevé, $<_T$ je stromové)

O stromech

- strom je $T \neq \emptyset$ s částečným uspořádáním $<_T$, které má nejmenší prvek (kořen) a množina předků libovolného vrcholu je dobře uspořádaná (každá její neprázdná podmnožina má nejmenší prvek, to zakáže nekonečné klesající řetězce předků)
- větev je maximální lineárně uspořádaná podmnožina T.
- uspořádaný strom má navíc lineární uspořádání $<_L$ množiny synů každého vrcholu (říkáme mu pravolevé, $<_T$ je stromové)
- označkovaný strom má navíc funkci $\mathrm{label}\colon T \to \mathrm{Labels}$

O stromech

- strom je $T \neq \emptyset$ s částečným uspořádáním $<_T$, které má nejmenší prvek (kořen) a množina předků libovolného vrcholu je dobře uspořádaná (každá její neprázdná podmnožina má nejmenší prvek, to zakáže nekonečné klesající řetězce předků)
- větev je maximální lineárně uspořádaná podmnožina T.
- uspořádaný strom má navíc lineární uspořádání $<_L$ množiny synů každého vrcholu (říkáme mu pravolevé, $<_T$ je stromové)
- označkovaný strom má navíc funkci $\mathrm{label}\colon T \to \mathrm{Labels}$

Königovo lemma: Nekonečný, konečně větvící strom má nekonečnou větev.