1 Převody do normálních forem

Příklad 1.1: Vyjádřete následující formule v DNF pomocí pravdivostní tabulky a pomocí převodu logických spojek.

a) $(A \Rightarrow B) \Rightarrow C$

b) $(A \Leftrightarrow B) \vee \neg C$

c) $(A \Leftrightarrow B) \Rightarrow (C \vee D)$

Formule je v disjunktivní normální formě (DNF), pokud má tvar $\alpha_1 \vee \ldots \vee \alpha_n$, *kde* $\alpha_i = A_{i1} \wedge \ldots \wedge A_{ij_i}$ *a každé* A_{ij} *je výroková proměnná nebo její negace.*

Řešení 1.1:

a) Pro formuli $(A \Rightarrow B) \Rightarrow C$ vytvoříme pravdivostní tabulku.

A	B	C	$A \Rightarrow B$	$(A \Rightarrow B) \Rightarrow C$
0	0	0	1	0
0	0	1	1	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Pro každou interpretaci, ve které je formule pravdivá, přidáme do vytvořené formule novou formuli, která bude obsahovat konjunkci všech výrokových symbolů z původní formule, následovně: Pokud je výrokovému symbolu v dáné interpretaci přiřazena pravdivostní hodnota 0, přidáme do formule negaci tohoto symbolu. V opačném případě do formule přidáme symbol samotný.

Výsledkem je pak formule v tzv. *úplné disjunktivní normální formě*:

$(\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C)$

Při přímem převodu postupujeme následovně: Ekvivalence a implikace nahrazujeme disjunkcemi, konjunkcemi a negacemi, pak uplatníme de Morganova pravidla a associativitu a distributivitu.

$$\begin{aligned} (A \Rightarrow B) \Rightarrow C &\Leftrightarrow (\neg A \vee B) \Rightarrow C \Leftrightarrow \\ &\Leftrightarrow \neg(\neg A \vee B) \vee C \Leftrightarrow \\ &\Leftrightarrow (A \wedge \neg B) \vee C \end{aligned}$$

b) Formuli $(A \Leftrightarrow B) \vee \neg C$ převedeme do DNF následovně:

$$\begin{aligned} (A \Leftrightarrow B) \vee \neg C &\Leftrightarrow ((A \Rightarrow B) \wedge (B \Rightarrow A)) \vee \neg C \Leftrightarrow \\ &\Leftrightarrow ((\neg A \vee B) \wedge (\neg B \vee A)) \vee \neg C \Leftrightarrow \\ &\Leftrightarrow (\neg A \wedge \neg B) \vee (\neg A \wedge A) \vee (B \wedge \neg B) \vee (B \wedge A) \vee \neg C \\ &\Leftrightarrow (\neg A \wedge \neg B) \vee (B \wedge A) \vee \neg C \end{aligned}$$

c) Přímým použitím převodních vztahů dostaneme:

$$\begin{aligned}(A \Leftrightarrow B) \Rightarrow (C \vee D) \quad &\Leftrightarrow \quad ((A \Rightarrow B) \wedge (B \Rightarrow A)) \Rightarrow (C \vee D) \Leftrightarrow \\ &\Leftrightarrow \quad \neg((A \Rightarrow B) \wedge (B \Rightarrow A)) \vee (C \vee D) \Leftrightarrow \\ &\Leftrightarrow \quad \neg((\neg A \vee B) \wedge (\neg B \vee A)) \vee (C \vee D) \Leftrightarrow \\ &\Leftrightarrow \quad (A \wedge \neg B) \vee (B \wedge \neg A) \vee C \vee D\end{aligned}$$

□

Příklad 1.2: Vyjádřete následující formule v CNF, a to pomocí pravdivostní tabulky a pomocí převodu logických spojek.

a) $(A \Leftrightarrow B) \Rightarrow (\neg A \wedge C)$

b) $(A \Rightarrow B) \Leftrightarrow (A \Rightarrow C)$

Formule je v konjunktivní normální formě (CNF), pokud má tvar $\alpha_1 \wedge \ldots \wedge \alpha_n$, *kde* $\alpha_i = A_{i1} \vee \ldots \vee A_{ij_i}$ *a* A_{ij} *je výroková proměnná nebo její negace.*

Řešení 1.2:

a) Z pravdivostní tabulky pro formuli $(A \Leftrightarrow B) \Rightarrow (\neg A \wedge C)$ vytvoříme ekvivalentní formuli v CNF takto:

A	B	C	$A \Leftrightarrow B$	$\neg A \wedge C$	$(A \Leftrightarrow B) \Rightarrow (\neg A \wedge C)$
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	0	0	1
1	0	1	0	0	1
1	1	0	1	0	0
1	1	1	1	0	0

Pro každou interpretaci, ve které je formule nepravdivá, přidáme do vytvářené formule novou formuli, která bude obsahovat disjunkci všech výrokových symbolů z původní formule, následovně: Pokud je výrokovému symbolu v dáné interpretaci přiřazena pravdivostní hodnota 1, přidáme do formule negaci tohoto symbolu. V opačném případě do formule přidáme symbol samotný.

Výsledkem je tentokrát formule v tzv. *úplné konjunktivní normální formě*:

$$(A \vee B \vee C) \wedge (\neg A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee \neg C)$$

Přímým užitím převodních vztahů pak dostaneme:

$$\begin{aligned}(A \Leftrightarrow B) \Rightarrow (\neg A \wedge C) \quad &\Leftrightarrow \quad ((A \Rightarrow B) \wedge (B \Rightarrow A)) \Rightarrow (\neg A \wedge C) \Leftrightarrow \\ &\Leftrightarrow \quad ((\neg A \vee B) \wedge (\neg B \vee A)) \Rightarrow (\neg A \wedge C) \Leftrightarrow \\ &\Leftrightarrow \quad \neg((\neg A \vee B) \wedge (\neg B \vee A)) \vee (\neg A \wedge C) \Leftrightarrow \\ &\Leftrightarrow \quad (\neg(\neg A \vee B) \vee \neg(\neg B \vee A)) \vee (\neg A \wedge C) \Leftrightarrow \\ &\Leftrightarrow \quad ((A \wedge \neg B) \vee (B \wedge \neg A)) \vee (\neg A \wedge C) \Leftrightarrow \\ &\Leftrightarrow \quad \ldots\end{aligned}$$

Uplatněním distributivních zákonů dostaneme požadovaný tvar.

b) Přímým převodem dostaneme:

$$\begin{aligned}
& (A \Rightarrow B) \Leftrightarrow (A \Rightarrow C) \Leftrightarrow \\
\Leftrightarrow\ & ((A \Rightarrow B) \Rightarrow (A \Rightarrow C)) \wedge ((A \Rightarrow C) \Rightarrow (A \Rightarrow B)) \Leftrightarrow \\
\Leftrightarrow\ & (\neg(A \Rightarrow B) \vee (A \Rightarrow C)) \wedge (\neg(A \Rightarrow C) \vee (A \Rightarrow B)) \Leftrightarrow \\
\Leftrightarrow\ & ((A \wedge \neg B) \vee (\neg A \vee C)) \wedge ((A \wedge \neg C) \vee (\neg A \vee B)) \Leftrightarrow \\
\Leftrightarrow\ & (((A \vee \neg A) \wedge (\neg B \vee \neg A)) \vee C) \wedge \\
& \wedge(((A \vee \neg A) \wedge (\neg C \vee \neg A)) \vee B) \Leftrightarrow \\
\Leftrightarrow\ & (\neg B \vee \neg A \vee C) \wedge (\neg C \vee \neg A \vee B)
\end{aligned}$$

Poznámka: *Některé formule jsou zároveň v disjunktivní i konjunktivní normální formě, například* $A \vee B \vee \neg C$ *nebo* $\neg A \wedge B$ *nebo třeba* $\neg C$.

Poznámka: *Každá formule má nějakou odpovídající DNF a CNF, tyto normální formy však nejsou určeny jednoznačně. Například CNF formule* $(P \Rightarrow Q) \wedge (Q \Rightarrow R) \wedge (R \Rightarrow P)$ *může být* $(\neg P \vee Q) \wedge (\neg Q \vee R) \wedge (\neg R \vee P)$*, ale stejně tak i* $(\neg P \vee R) \wedge (\neg R \vee Q) \wedge (\neg Q \vee P)$. □

Příklad 1.3: Víme, že můžeme provádět následující chemické reakce:

$$\begin{aligned}
\mathrm{MgO} + \mathrm{H_2} &\rightarrow \mathrm{Mg} + \mathrm{H_2O} \\
\mathrm{C} + \mathrm{O_2} &\rightarrow \mathrm{CO_2} \\
\mathrm{CO_2} + \mathrm{H_2O} &\rightarrow \mathrm{H_2CO_3}
\end{aligned}$$

K dispozici máme dále sloučeniny C, H_2, O_2 a MgO. Rezolucí dokažte, že H_2CO_3 logicky vyplývá ze zadaných reakcí.

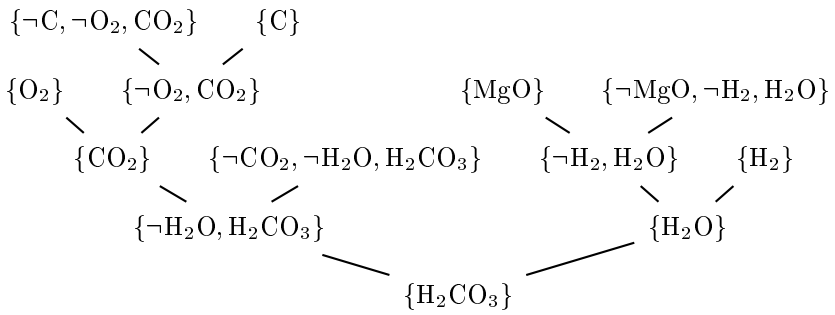
Řešení 1.3: Reakcím odpovídají následující formule[1]:

$$\begin{aligned}
(\mathrm{MgO} \wedge \mathrm{H_2}) &\Rightarrow (\mathrm{Mg} \wedge \mathrm{H_2O}) \\
(\mathrm{C} \wedge \mathrm{O_2}) &\Rightarrow \mathrm{CO_2} \\
(\mathrm{CO_2} \wedge \mathrm{H_2O}) &\Rightarrow \mathrm{H_2CO_3}
\end{aligned}$$

Převodem formulí do CNF obdržíme množinu klauzulí:

$$\begin{aligned}
S = \{\ & \{\neg\mathrm{MgO}, \neg\mathrm{H_2}, \mathrm{Mg}\}, \{\neg\mathrm{MgO}, \neg\mathrm{H_2}, \mathrm{H_2O}\}, \{\neg\mathrm{C}, \neg\mathrm{O_2}, \mathrm{CO_2}\}, \\
& \{\neg\mathrm{CO_2}, \neg\mathrm{H_2O}, \mathrm{H_2CO_3}\}, \{\mathrm{C}\}, \{\mathrm{H_2}\}, \{\mathrm{O_2}\}, \{\mathrm{MgO}\}\}
\end{aligned}$$

Z množiny klauzulí S se můžeme pokusit přímo odvodit rezolventu H_2CO_3:



[1] Předpokládáme, že symbol + má v rovnicích vyšší prioritu než →.

Další možností je pokusit se nalézt rezoluční vyvracení množiny $S \cup \{\neg H_2CO_3\}$. Protože množina S obsahuje pouze Hornovy klauzule, můžeme použít SLD rezoluce. Nejdříve převedeme klauzule na uspořádané klauzule (definite clauses). Obdržíme tak následující množinu klauzulí:

$$S = \{\ [Mg, \neg MgO, \neg H_2], [H_2O, \neg MgO, \neg H_2], [CO_2, \neg C, \neg O_2], \\ [H_2CO_3, \neg CO_2, \neg H_2O], [C], [H_2], [O_2], [MgO]\}$$

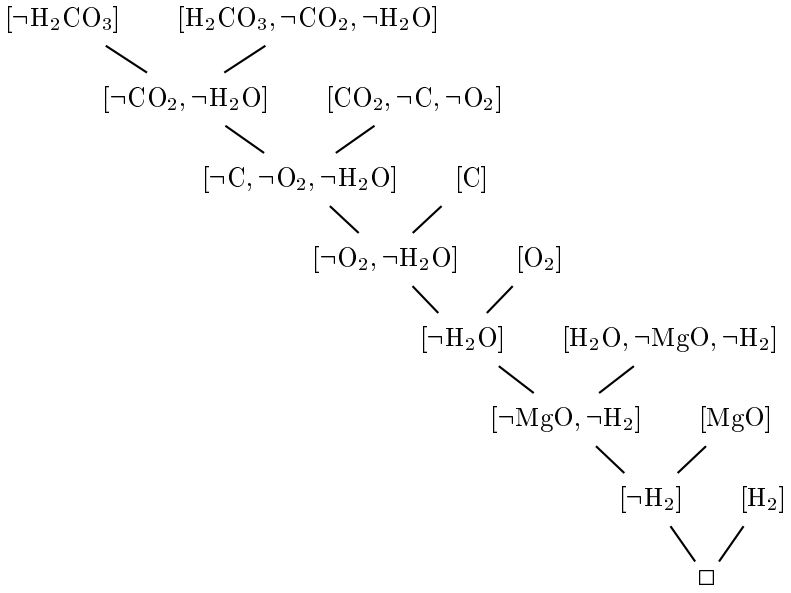
Rezoluční pravidlo upravíme tak, že rezolventou klauzulí

$$\begin{aligned} G &= [\neg p_1, \neg p_2, \ldots, \neg p_n] \text{ a} \\ H &= [q, \neg q_1, \neg q_2, \ldots, \neg q_m], \end{aligned}$$

kde $p_1 = \neg q$, bude uspořádaná klauzule

$$[\neg q_1, \neg q_2, \ldots, \neg q_m, \neg p_2, \ldots, \neg p_n].$$

Důkaz SLD-rezolučním vyvracením začneme dotazem $[\neg H_2CO_3]$. Literál, podle kterého budeme dále rezolvovat vybereme dle tzv. selekčního (výběrového) pravidla[2]. Selekční pravidlo je tedy definováno tak, že vybírá nejlevější literál.



Z množiny S a negace závěru (dotazu nebo též cíle $\neg H_2CO_3$) se nám podařilo odvodit prázdnou klauzuli $\square$. Tím jsme dokázali platnost $S \models H_2CO_3$.

$\square$

Příklad 1.4: Převeďte na prenexovou normální formu formule:

a) $\forall y(\exists x P(x,y) \Rightarrow Q(y,z)) \wedge \exists y(\forall x R(x,y) \vee Q(x,y))$

b) $\exists x R(x,y) \Leftrightarrow \forall y P(x,y)$

[2] Odtud pochází název metody – SLD, jedná se o lineární rezoluci se selekčním pravidlem.

c) $(\forall x \exists y Q(x,y) \vee \exists x \forall y P(x,y)) \wedge \neg \exists x \exists y P(x,y)$

d) $\neg(\forall x \exists y P(x,y) \Rightarrow \exists x \exists y R(x,y)) \wedge \forall x(\neg \exists y Q(x,y))$

Formule je v prenexové normání formě (PNF), pokud jsou všechny kvatifikátory na začátku formule, tj. formule má tvar $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi$, kde $Q_1, \ldots, Q_n \in \{\forall, \exists\}$ a φ je formule bez kvantifikátorů (tzv. otevřená formule).

Řešení 1.4: Pro převod do prenexové formy používáme tato pravidla:

$$\overrightarrow{Q}x\ \neg \forall y \varphi \quad \Leftrightarrow \quad \overrightarrow{Q}x\ \exists y \neg \varphi \tag{1}$$

$$\overrightarrow{Q}x\ \neg \exists y \varphi \quad \Leftrightarrow \quad \overrightarrow{Q}x\ \forall y \neg \varphi \tag{2}$$

$$\overrightarrow{Q}x\ (\forall y \varphi \oplus \psi) \quad \Leftrightarrow \quad \overrightarrow{Q}x\ \forall z(\varphi(y/z) \oplus \psi) \tag{3}$$

$$\overrightarrow{Q}x\ (\exists y \varphi \oplus \psi) \quad \Leftrightarrow \quad \overrightarrow{Q}x\ \exists z(\varphi(y/z) \oplus \psi) \tag{4}$$

Symbol $\overrightarrow{Qx}$ označuje vektor kvantifikátorů, které již splňují požadavky na PNF. Symbol $\oplus$ v rovnicích (3) a (4) zastupuje logickou spojku $\wedge$ nebo $\vee$. Výrazem (y/z) rozumíme substituci proměnné y za proměnnou z (proměnná y je ve formuli φ nahrazena proměnnou z), která se nevyskytuje nikde ve formuli.

a) užitím uvedených pravidel dostaneme z

$$\forall y(\exists x P(x,y) \Rightarrow Q(y,z)) \wedge \exists y(\forall x R(x,y) \vee Q(x,y))$$

formuli v PNF takto[3]:

$$\begin{aligned}
& \forall y(\exists x P(x,y) \Rightarrow Q(y,z)) \wedge \exists y(\forall x R(x,y) \vee Q(x,y)) \Leftrightarrow \\
\Leftrightarrow\ & \forall y(\neg(\exists x P(x,y)) \vee Q(y,z)) \wedge \exists y(\forall x R(x,y) \vee Q(x,y)) \Leftrightarrow \\
\Leftrightarrow\ & \forall y(\forall x \neg P(x,y) \vee Q(y,z)) \wedge \exists y(\forall x(R(x,y) \vee Q(x,y)) \Leftrightarrow \\
\Leftrightarrow\ & \forall y_1[(\forall x \neg P(x,y_1) \vee Q(y_1,z)) \wedge \exists y(\forall x R(x,y) \vee Q(x,y))] \Leftrightarrow \\
\Leftrightarrow\ & \forall y_1 \forall x_1[(\neg P(x_1,y_1) \vee Q(y_1,z)) \wedge \exists y(\forall x R(x,y) \vee Q(x,y))] \Leftrightarrow \\
\Leftrightarrow\ & \forall y_1 \forall x_1 \exists y_2[(\neg P(x_1,y_1) \vee Q(y_1,z)) \wedge (\forall x R(x,y_2) \vee Q(x,y_2))] \Leftrightarrow \\
\Leftrightarrow\ & \forall y_1 \forall x_1 \exists y_2[(\neg P(x_1,y_1) \vee Q(y_1,z)) \wedge \forall x_2(R(x_2,y_2) \vee Q(x,y_2))] \Leftrightarrow \\
\Leftrightarrow\ & \forall y_1 \forall x_1 \exists y_2 \forall x_2[(\neg P(x_1,y_1) \vee Q(y_1,z)) \wedge (R(x_2,y_2) \vee Q(x,y_2))]
\end{aligned}$$

b) formuli $\exists x R(x,y) \Leftrightarrow \forall y P(x,y)$ převedeme následovně:

$$\begin{aligned}
& \exists x R(x,y) \Leftrightarrow \forall y P(x,y) \Leftrightarrow \\
\Leftrightarrow\ & (\exists x R(x,y) \Rightarrow \forall y P(x,y)) \wedge (\forall y P(x,y) \Rightarrow \exists x R(x,y)) \Leftrightarrow \\
\Leftrightarrow\ & (\neg \exists x R(x,y) \vee \forall y P(x,y)) \wedge (\neg \forall y P(x,y) \vee \exists x R(x,y)) \Leftrightarrow \\
\Leftrightarrow\ & (\forall x \neg R(x,y) \vee \forall y P(x,y)) \wedge (\exists y \neg P(x,y) \vee \exists x R(x,y)) \Leftrightarrow \\
\Leftrightarrow\ & \forall x_1[(\neg R(x_1,y) \vee \forall y P(x,y)) \wedge (\exists y \neg P(x,y) \vee \exists x R(x,y))] \Leftrightarrow \\
\Leftrightarrow\ & \forall x_1 \forall y_1[(\neg R(x_1,y) \vee P(x,y_1)) \wedge (\exists y \neg P(x,y) \vee \exists x R(x,y))] \Leftrightarrow \\
\Leftrightarrow\ & \forall x_1 \forall y_1[(\neg R(x_1,y) \vee P(x,y_1)) \wedge \exists y_2(\neg P(x,y_2) \vee \exists x R(x,y))] \Leftrightarrow
\end{aligned}$$

[3] Při převodu pro zjednodušení vynecháme krok, který vysune kvantifikátory v levé části formule mezi vnější (značíme symbolem "[") a vnitřní závorku.

$$\Leftrightarrow \quad \forall x_1 \forall y_1 \exists y_2 [(\neg R(x_1, y) \vee P(x, y_1)) \wedge (\neg P(x, y_2) \vee \exists x R(x, y))] \Leftrightarrow$$
$$\Leftrightarrow \quad \forall x_1 \forall y_1 \exists y_2 [(\neg R(x_1, y) \vee P(x, y_1)) \wedge \exists x_2 (\neg P(x, y_2) \vee R(x_2, y))] \Leftrightarrow$$
$$\Leftrightarrow \quad \forall x_1 \forall y_1 \exists y_2 \exists x_2 [(\neg R(x_1, y) \vee P(x, y_1)) \wedge (\neg P(x, y_2) \vee R(x_2, y))]$$

□

2 Skolemizace a unifikace

Začneme připomenutím následujících definic.

- Formle bez volných proměnných se nazývá *sentence*.
- Nechť $\varphi(x_1, \ldots, x_n)$ je formule s volnými proměnnými $x_1, \ldots, x_n$, kde $n \geq 0$. Jejím *univerzálním uzávěrem* rozumíme formuli $\forall x_1 \ldots \forall x_n \varphi(x_1, \ldots, x_n)$.
- Formule $\varphi(x_1, \ldots, x_n)$ je *pravdivá v interpretaci* I právě tehdy, když je v této interpretaci pravdivý její univerzální uzávěr, tj. pokud $\varphi(x_1, \ldots, x_n)$ je pravdivá v interpretaci I pro všechny valuace.
- Podobně, formule $\varphi(x_1, \ldots, x_n)$ je *splnitelná*, je-li splnitelný její univerzální uzávěr, tj. pokud existuje interpretace I taková, že $\varphi(x_1, \ldots, x_n)$ je pravdivá pro všechny valuace.
- Formule φ, ψ jsou *equisatisfiable* (ekvivalentní vzhledem ke splnitelnosti), pokud jsou obě splnitelné nebo obě nesplnitelné.
- Nechť T je množina formulí. Formule φ je *logickým důsledkem* T (nebo φ *logicky vyplývá* z T, píšeme $T \models \varphi$,), pokud je φ pravdivá v každé interpretaci I, ve které jsou pravdivé všechny formule z T.

Nyní si ukážeme, jak lze převést úlohu *Dokažte, že φ je logickým důsledkem T* do klauzulární formy (tj. konjunktivní normální formy), kde již může být dořešena rezoluční metodou.

a) Nejprve nahradíme všechny formule s volnými proměnnými v T jejich univerzálními uzávěry. Vzniklou množinu sentencí označíme T'. Má-li formule φ volné proměnné $x_1, \ldots, x_n$, nahradíme ji jejím univerzálním uzávěrem $\forall x_1 \ldots \forall x_n \varphi(x_1, \ldots, x_n)$. Z výše uvedených definic plyne, že $T \models \varphi$ právě tehdy, když $T' \models \forall x_1 \ldots \forall x_n \varphi(x_1, \ldots, x_n)$.

b) $T' \models \forall x_1 \ldots \forall x_n \varphi(x_1, \ldots, x_n)$ právě tehdy, když je množina sentencí $T' \cup \{\neg \forall x_1 \ldots \forall x_n \varphi(x_1, \ldots, x_n)\} = T' \cup \{\exists x_1 \ldots \exists x_n \neg \varphi(x_1, \ldots, x_n)\}$ nesplnitelná.

c) Všechny formule z množiny $T' \cup \{\exists x_1 \ldots \exists x_n \neg \varphi(x_1, \ldots, x_n)\}$ převedeme do prenexního normální formy (PNF) a skolemizujeme (skolemizace vytvoří formuli, která je equisatisfiable původní formuli). Získanou množinu sentencí v PNF a bez existenčních kvantifikátorů označíme T''.

d) Nyní z každé formule z T'' odstraníme část s kvantifikátory (korektnost tohoto kroku opět vyplývá z výše uvedených definic) a zbylou část formule převedeme do konjunktivní normální formy. Konjunkci všech takto získaných formulí nazveme ψ. Platí $T \models \varphi$ právě tehdy, když ψ je nesplnitelná. Formuli ψ přepíšeme na množinu klauzulí a každou klauzuli

nahradíme množinou jejích literálů. K důkazu nesplnitelnosti takto vytvořené množiny klauzulí můžeme použít rezoluční metodu.

Pořadí některých částí výše uvedeného postupu lze zaměnit, např. převod do konjunktivní normální formy lze provést zaráz s převodem do PNF.

Příklad 2.1: Proveďte skolemizaci následujících formulí v PNF:

a) $\forall y_1 \forall x_1 \exists y_2 \forall x_2 [(\neg P(x_1, y_1) \vee Q(y_1, a)) \wedge (R(x_2, y_2) \vee Q(x_1, y_2))]$

b) $\forall x_1 \forall y_1 \exists y_2 \exists x_2 [(\neg R(x_1, y_2) \vee P(b, y_1)) \wedge (\neg P(x_1, y_2) \vee R(x_2, b))]$

c) $\exists x_1 \forall y_1 \exists x_2 (S(y_1) \vee R(x_1, x_2))$

Řešení 2.1: Každý výskyt proměnné x, která je ve formuli kvantifikovaná existenčně, nahradíme termem $f(y_1, \ldots, y_n)$, kde f je nový funkční symbol a $y_1, \ldots, y_n$ jsou všechny univerzálně kvantifikované proměnné, které se vyskytují v sekvenci kvantifikátorů před proměnnou x. Pokud se před x žádné takové proměnné nevyskytují, nahradíme x nulárním funkčním symbolem, tj. konstantou.

Skolemizace se aplikuje na formule bez volných proměnných. Formuli s volnými proměnnými je tedy třeba před skolemizací nahradit jejím univerzálním uzávěrem (univerzální uzávěr i skolemizace zachovávají splnitelnost formule).

a) První formuli upravíme následovně:

$$\begin{array}{rrll} & \forall y_1 \forall x_1 \exists y_2 \forall x_2 & [(\neg P(x_1, y_1) \vee Q(y_1, a)) \wedge & \\ & & \wedge (R(x_2, y_2) \vee Q(x_1, y_2))] & \rightarrow \\ \rightarrow & \forall y_1 \forall x_1 \forall x_2 & [(\neg P(x_1, y_1) \vee Q(y_1, a)) \wedge & \\ & & \wedge (R(x_2, f(y_1, x_1)) \vee Q(x_1, f(y_1, x_1)))] & \end{array}$$

b) Stejným způsobem upravíme formuli:

$$\begin{array}{rrll} & \forall x_1 \forall y_1 \exists y_2 \exists x_2 & [(\neg R(x_1, y_2) \vee P(b, y_1)) \wedge & \\ & & \wedge (\neg P(x_1, y_2) \vee R(x_2, b))] & \rightarrow \\ \rightarrow & \forall x_1 \forall y_1 \exists x_2 & [(\neg R(x_1, f(x_1, y_1)) \vee P(b, y_1)) \wedge & \\ & & \wedge (\neg P(x_1, f(x_1, y_1)) \vee R(x_2, b))] & \rightarrow \\ \rightarrow & \forall x_1 \forall y_1 & [(\neg R(x_1, f(x_1, y_1)) \vee P(b, y_1)) \wedge & \\ & & \wedge (\neg P(x_1, f(x_1, y_1)) \vee R(g(x_1, y_1), b))] & \end{array}$$

c) Skolemizace třetí formule ukazuje náhradu proměnné novou konstantou:

$$\begin{array}{rrll} & \exists x_1 \forall y_1 \exists x_2 & (S(y_1) \vee R(x_1, x_2)) & \rightarrow \\ \rightarrow & \forall y_1 \exists x_2 & (S(y_1) \vee R(a, x_2)) & \rightarrow \\ \rightarrow & \forall y_1 & (S(y_1) \vee R(a, f(y_1))) & \end{array}$$

□

Příklad 2.2: Najděte nejobecnější unifikátory (angl. *most general unifiers*, *mgu*) následujících množin literálů:

a) $S = \{P(x, f(y), z), P(g(a), f(w), u), P(v, f(b), c)\}$

b) $T = \{Q(h(x, y), w), Q(h(g(v), a), f(v)), Q(h(g(v), a), f(b))\}$

Řešení 2.2: Při hledání *mgu* hledáme rozdíly mezi výrazy a substituujeme volné proměnné tak dlouho, dokud vstupní množina neobsahuje právě jeden výraz.[4]

Rozdíl $D(S)$ mezi výrazy z množiny S definujeme jako množinu podvýrazů všech $E \in S$ začínajících na první (nejlevější) pozici, na které mají výrazy E různou hodnotu.

a) Nejobecnější unifikátor pro množinu

$$S = \{P(x, f(y), z), P(g(a), f(w), u), P(v, f(b), c)\}$$

zkonstruujeme následovně.

1. $S_0 = S$, $|S_0| \neq 1$ a je tedy co unifikovat. $D(S) = \{x, g(a), v\}$. Máme čtyři možnosti jak substituovat ($[x/g(a)]$, $[x/v]$, $[v/g(a)]$ a $[v/x]$). Zvolíme první možnost, tedy $\phi_1 = [x/g(a)]$ a aplikujeme ϕ_1 na S_0, čímž obdržíme množinu S_1:

$$S_1 = S_0\phi_1 = \{P(g(a), f(y), z), P(g(a), f(w), u), P(v, f(b), c)\}$$

2. $D(S_1) = \{g(a), v\}$, $\phi_2 = [v/g(a)]$.

$$S_2 = S_1\phi_2 = \{P(g(a), f(y), z), P(g(a), f(w), u), P(g(a), f(b), c)\}$$

3. $D(S_2) = \{y, w, b\}$, $\phi_3 = [y/w]$.

$$S_3 = S_2\phi_2 = \{P(g(a), f(w), z), P(g(a), f(w), u), P(g(a), f(b), c)\}$$

4. $D(S_3) = \{w, b\}$, $\phi_3 = [w/b]$.

$$S_4 = S_3\phi_3 = \{P(g(a), f(b), z), P(g(a), f(b), u), P(g(a), f(b), c)\}$$

5. $D(S_4) = \{z, u, c\}$, $\phi_3 = [z/u]$.

$$S_5 = S_4\phi_4 = \{P(g(a), f(b), u), P(g(a), f(b), u), P(g(a), f(b), c)\}$$

6. $D(S_5) = \{u, c\}$, $\phi_3 = [u/c]$.

$$S_6 = S_5\phi_5 = \{P(g(a), f(b), c)\}$$

7. $|S_6| = 1$, takže algoritmus pro nalezení unifikátoru ukončí výpočet. Nejobecnějším unifikátorem je posloupnost substitucí:

$$\begin{aligned} mgu(S) &= \phi_1\phi_2\phi_3\phi_4\phi_5 = \\ &= [x/g(a)][v/g(a)][y/w][w/b][z/u][u/c] \end{aligned}$$

b) Aplikací popsaného postupu dostaneme pro množinu T tento nejobecnější unifikátor:

$$mgu(T) = [x/g(v)][y/a][w/f(v)][v/b]$$

□

[4] Viz materiály k předmětu *IB101 Úvod do logiky a logického programování* (šestá přednáška).

Příklad 2.3: Najděte všechny rezolventy následujících dvojic klauzulí:

a) $C_1 = \{P(x,y), P(y,z)\}, C_2 = \{\neg P(u, f(u))\}$

b) $C_1 = \{P(x,x), \neg R(x, f(x))\}, C_2 = \{R(x,y), Q(y,z)\}$

c) $C_1 = \{P(x,y), \neg P(x,x), Q(x, f(x), z)\}, C_2 = \{\neg Q(f(x), x, z), P(x,z)\}$

Řešení 2.3: Před rezolucí je třeba přejmenovat proměnné v klauzulích tak, aby obě klauzule používaly různé proměnné. Rezoluce v predikátové logice probíhá obdobně jako v logice propoziční. Rezoluční pravidlo je definováno následovně:[5] Mějme klauzule C_1 a C_2 bez společných proměnných ve tvaru:

$$\begin{aligned} C_1 &= C_1' \cup \{P(\vec{x_1}), \ldots, P(\vec{x_n})\} \\ C_2 &= C_2' \cup \{\neg P(\vec{y_1}), \ldots, \neg P(\vec{y_m})\}. \end{aligned}$$

Je-li substituce ϕ nejobecnějším unifikátorem množiny

$$\{P(\vec{x_1}), \ldots, P(\vec{x_n}), P(\vec{y_1}), \ldots, P(\vec{y_m})\},$$

pak rezolventou C_1 a C_2 je $C_1'\phi \cup C_2'\phi$.

Poznámka:

- *Přejmenování proměnných je nutné:* $\{\{P(x)\}, \{\neg P(f(x))\}\}$ *je nesplnitelná množina klauzulí, ovšem literály* $P(x)$ *a* $P(f(x))$ *nejsou unifikovatelné.*
- *Rezolvování na více literálech je nutné:* $\{\{\neg P(x), \neg P(y)\}, \{P(x), P(y)\}\}$ *je nesplnitelná množina klauzulí, ovšem rezolvováním na jednom literálu bychom se nikdy nedostali k prázdné klauzuli.*

Na zadané klauzule aplikujeme rezoluční pravidlo a pokusíme se nálézt všechny rezolventy, které lze odvodit.

a) Přejmenování proměnných není v tomto případě nutné, protože klauzule neobsahují společné proměnné.

   1. nejdříve zkusíme odvodit prázdnou klauzuli $\square$. Protože však nelze unifikovat množinu $\{P(x,y), P(y,z), P(u, f(u))\}$, není $\square$ rezolventou C_1 a C_2.
   2. zkusíme tedy unifikovat množinu $\{P(x,y), P(u, f(u))\}$. Zjistíme, že *mgu* této množiny je posloupnost substitucí $\phi = [x/u][y/f(u)]$ a můžeme tedy odvodit rezolventu

$$\begin{aligned} C_1'\phi \cup C_2'\phi &= \\ &= (C_1 \setminus \{P(x,y)\})\phi \cup (C_2 \setminus \{\neg P(u, f(u))\})\phi = \\ &= \{P(y,z)\}\phi \cup \emptyset = \\ &= \{P(f(u), z)\}. \end{aligned}$$

   3. nakonec můžeme unifikovat množinu $\{P(y,z), P(u, f(u))\}$ pomocí *mgu* $\phi = [y/u][z/f(u)]$. Rezolventou je poté klauzule $\{P(x,u)\}$.

$\square$

[5] Viz materiály k předmětu *IB101 Úvod do logiky a logického programování* (sedmá přednáška).

3 Rezoluce

Příklad 3.1: Dokažte, že platí následující vyplývání:

a) $\{\forall x P(x,x), \forall x\forall y\forall z((P(x,y) \wedge P(y,z)) \Rightarrow P(z,x))\} \models \forall x\forall y(P(x,y) \Rightarrow P(y,x))$

b) $\{\forall x\forall y\forall z((P(x,y) \wedge P(y,z)) \Rightarrow P(x,z)), \forall x\forall y(P(x,y) \Rightarrow P(y,x))\} \models \forall x\forall y\forall z((P(x,y) \wedge P(z,y)) \Rightarrow P(x,z))$

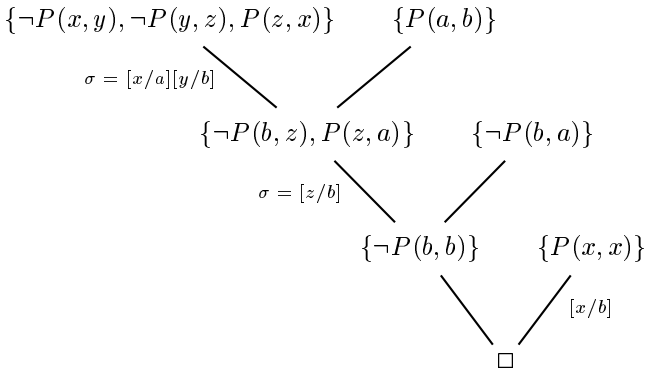
Řešení 3.1: Stejně jako ve výrokové logice provedeme důkaz sporem pomocí rezoluční metody. Protože jsou všechny formule v zadání tohoto příkladu uzavřené, stačí nalézt vyvrácení množiny $S \cup \{\neg\varphi\}$, kde S je množina formulí z předpokladů a φ je formule ze závěru logického vyplývání.

Množinu formulí $S \cup \{\neg\varphi\}$ tedy převedeme na množinu klauzulí[6] a pomocí rezolučního pravidla pro predikátovou logiku se pokusíme odvodit prázdnou klauzuli.

a) Znegováním závěru získáme formuli $\neg\varphi = \exists x\exists y\neg(P(x,y) \Rightarrow P(y,x))$. Převodem této formule a množiny předpokladů do PNF, odstraněním kvantifikací a převodem do CNF obdržíme množinu klauzulí

$$\{\{P(x,x)\}, \{\neg P(x,y), \neg P(y,z), P(z,x)\}, \{P(a,b)\}, \{\neg P(b,a)\}\}.$$

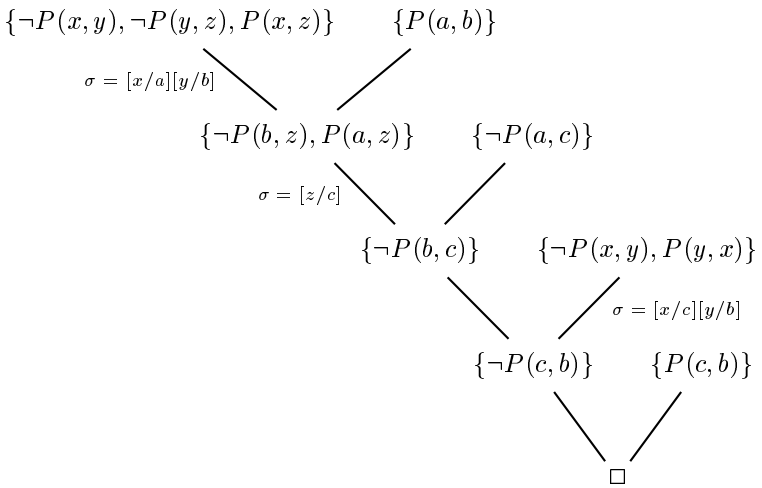
Z ní můžeme vytvořit např. tento strom rezolučního vyvracení:



b) Obdobně jako v předchozím případě vyvrátíme množinu klauzulí:

$$\{\{\neg P(x,y), \neg P(y,z), P(x,z)\}, \{\neg P(x,y), P(y,x)\},$$
$$\{P(a,b)\}, \{P(c,b)\}, \{\neg P(a,c)\}\}.$$

[6] Podrobný obecný postup transformace zadání na množinu klauzulí je popsán v materiálech ke druhému cvičení.



V obou případech se nám podařilo odvodit z množiny $S \cup \{\neg\varphi\}$ prazdnou klauzuli a tím jsme dokázali platnost logického vyplývání $S \models \varphi$. □

Příklad 3.2: Převeďte následující tvrzení v přirozeném jazyce na formule predikátové logiky a dokažte jejich platnost.

a) Předpokládejte, že platí následující tři tvrzení:

- Existuje drak (označme $D/1$).
- Draci spí ($S/1$) nebo loví ($L/1$).
- Když jsou draci hladoví ($H/1$), tak nespí.

Důsledek: *Když jsou draci hladoví, tak loví.*

b) Předpokládejte, že platí následující dvě tvrzení:

- Všichni holiči ($B/1$) holí ($S/2$) každého, kdo se neholí sám.
- Žádný holič neholí někoho, kdo se holí sám.

Důsledek: *Holiči neexistují.*

Řešení 3.2: Převedeme věty přirozeného jazyka na formule predikátové logiky a rezolučním principem dokážeme platnost tvrzení.

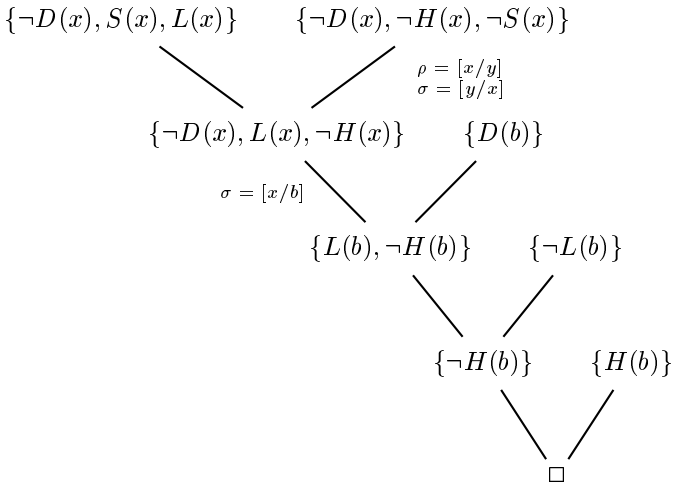
a) Přepisem obdržíme následující množinu formulí:

$$\begin{aligned} S &= \{\exists x D(x), \\ &\quad \forall x(D(x) \Rightarrow (S(x) \vee L(x))), \\ &\quad \forall x((D(x) \wedge H(x)) \Rightarrow \neg S(x))\} \\ \varphi &= \forall x((D(x) \wedge H(x)) \Rightarrow L(x)) \end{aligned}$$

Všechny předpoklady z S a negaci závěru φ převedeme na klauzulární tvar a dokazujeme nesplnitelnost takto vzniklé množiny klauzulí:

$$\begin{aligned} S' &= \{\{D(a)\}, \{\neg D(x), S(x), L(x)\}, \{\neg D(x), \neg H(x), \neg S(x)\}, \\ &\quad \{D(b)\}, \{H(b)\}, \{\neg L(b)\}\} \end{aligned}$$

V dalším textu budeme při zápisu rezoluce obvykle používat ρ pro označení přejmenování proměnných v klauzulích a σ pro označení *mgu* substituce.

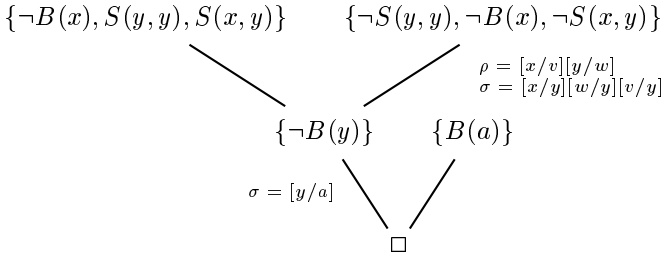


b) Přepisem obdržíme:

$$\begin{aligned} S &= \{\forall x \forall y((B(x) \wedge \neg S(y,y)) \Rightarrow S(x,y)), \\ &\quad \forall y(S(y,y) \Rightarrow \neg \exists x(B(x) \wedge S(x,y)))\} \\ \varphi &= \neg \exists x B(x) \end{aligned}$$

Množinu formulí $S \cup \{\neg\varphi\}$ převedeme na množinu klauzulí S', ke které pak zkonstruujeme rezoluční vyvrácení.

$$S' = \{\{\neg B(x), S(y,y), S(x,y)\}, \{\neg S(y,y), \neg B(x), \neg S(x,y)\}, \{B(a)\}\}$$



□

Příklad 3.3: Vyvraťte následující množinu klauzulí

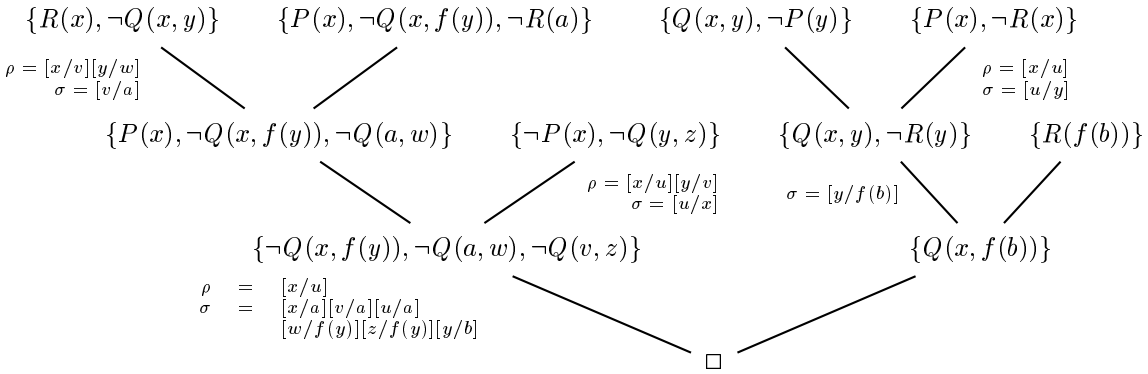
$$\begin{aligned} S &= \{\{P(x), \neg Q(x, f(y)), \neg R(a)\}, \{R(x), \neg Q(x,y)\}, \{\neg P(x), \neg Q(y,z)\}, \\ &\quad \{P(x), \neg R(x)\}, \{R(f(b))\}, \{Q(x,y), \neg P(y)\}\} \end{aligned}$$

pomocí obecné rezoluce, lineární rezoluce, LI rezoluce, LD rezoluce a SLD rezoluce[7].

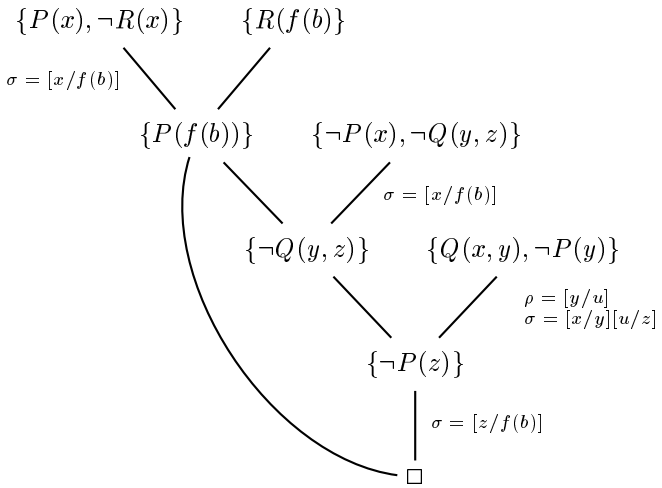
[7] Viz materiály k předmětu *IB101 Úvod do logiky a logického programování* (sedmá přednáška).

Řešení 3.3:

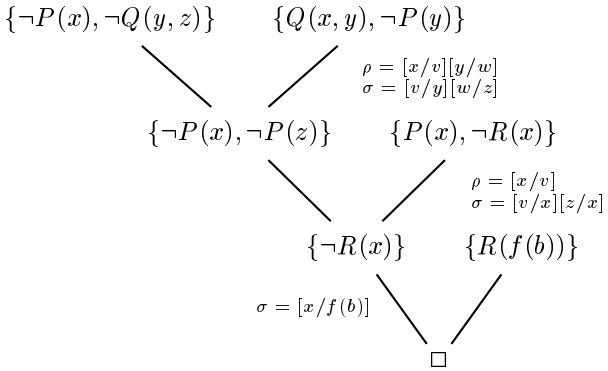
a) Obecná rezoluce



b) Lineární rezoluce – rezolvujeme vždy předchozí rezolventu s klauzulí z vyvracené množiny nebo dříve odvozenou rezolventou. Důkaz má lineární strukturu. Lineární rezoluce je úplná.



c) LI rezoluce (lineární vstupní rezoluce) – lineární vstupní rezoluce začíná cílovou klauzulí (klauzulí, která neobsahuje žádný pozitivní literál) a rezolvuje vždy předchozí rezolventou s klauzulí z vyvracené množiny. Jedná se o zjemnění lineární rezoluce, které není obecně úplné. Aplikací na množinu Hornových klauzulí dostaneme úplnou metodu.



d) LD rezoluce – zjemnění LI rezoluce, které pracuje výhradně s množinou Hornových klauzulí. Klauzule jsou nahrazeny uspořádanými seznamy stejně jako ve výrokové logice. Z původní množiny S obdržíme množinu

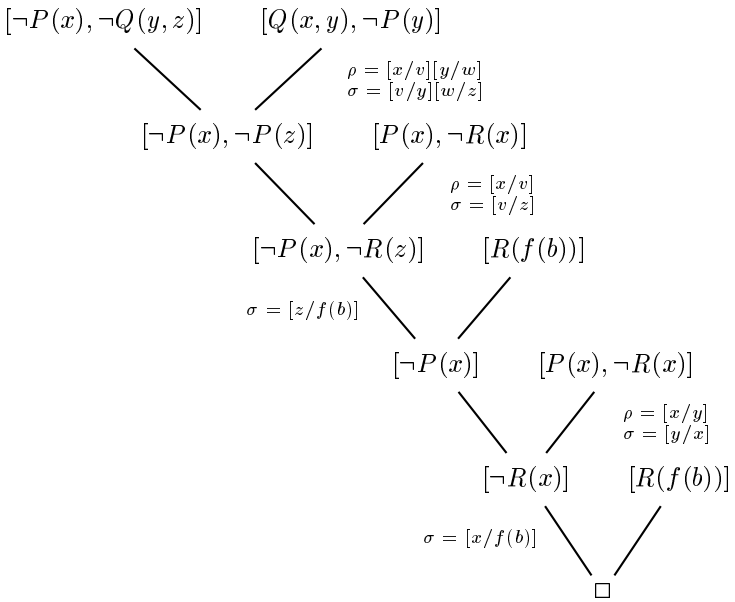
$$\begin{aligned} S' \quad = \quad & \{[P(x), \neg Q(x, f(y)), \neg R(a)], [R(x), \neg Q(x, y)], [\neg P(x), \neg Q(y, z)], \\ & [P(x), \neg R(x)], [R(f(b))], [Q(x, y), \neg P(y)]\} \end{aligned}$$

Rezoluční pravidlo je pak definováno následovně: Mějme uspořádané klauzule

$$\begin{aligned} G \quad = \quad & [\neg A_1, \neg A_2, \ldots, \neg A_n] \text{ a} \\ H \quad = \quad & [B_0, \neg B_1, \neg B_2, \ldots, \neg B_m]. \end{aligned}$$

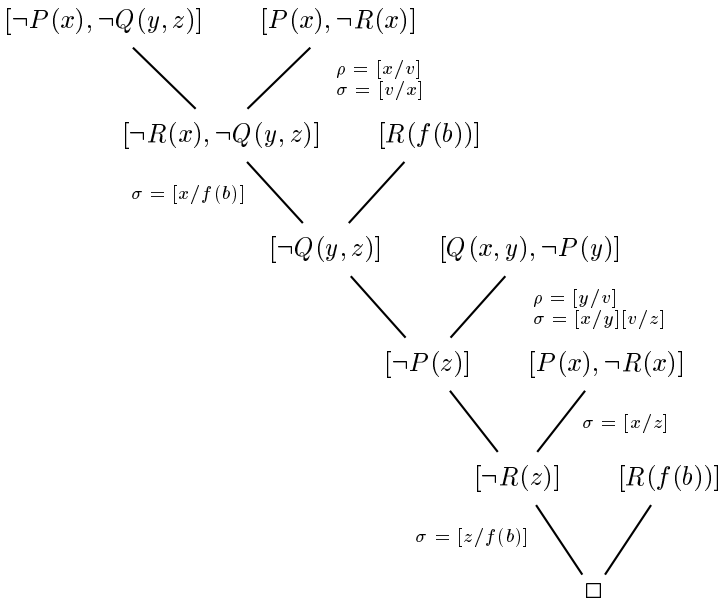
Rezolventou G a H pro $\phi = mgu(B_0, A_i)$ bude uspořádaná klauzule

$$[\neg A_1\phi, \neg A_2\phi, \ldots, \neg A_{i-1}\phi, \neg B_1\phi, \neg B_2\phi, \ldots, \neg B_m\phi, \neg A_{i+1}\phi, \ldots, \neg A_n\phi].$$



e) SLD rezoluce (LD rezoluce se selekčním pravidlem) – jedná se o speciální případ LD rezoluce. K výběru literálu z předchozí rezolventy, podle kterého

se bude rezolvovat dále, slouží tzv. výběrové pravidlo. Toto pravidlo je definováno tak, že volí vždy první literál z předchozí rezolventy.



$\square$

4 SLD-stromy a rezoluce v Prologu

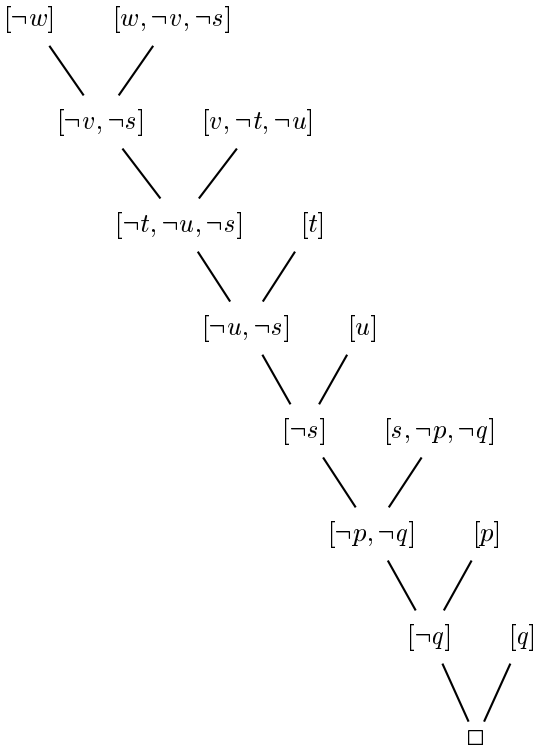
Příklad 4.1: Nalezněte rezoluční vyvracení množiny klauzulí zadaných jako program a dotaz v Prologu.

```
1. r :- p, q.        5. t.
2. s :- p, q.        6. q.
3. v :- t, u.        7. u.
4. w :- v, s.        8. p.

?- w.
```

Řešení 4.1: Program v jazyce Prolog nejprve převedeme na formuli výrokové logiky v konjunktivní normální formě. Přitom postupujeme tak, že každou programovou klauzuli tvaru `p :- q`$_1$`,...,q`$_n$`.` převedeme na uspořádanou klauzuli tvaru $[p, \neg q_1, \ldots, \neg q_n]$, fakt `q.` na klauzuli $[q]$ a dotaz `?- p`$_1$`,...,p`$_n$`.` na cílovou klauzuli $[\neg p_1, \ldots, \neg p_n]$. Získáme tak formuli (resp. množinu klauzulí) v konjunktivní normální formě, která obsahuje pouze Hornovy klauzule. K vyvracení této množiny použijeme SLD-rezoluci.

$$\begin{aligned} S \quad = \quad & \{[r, \neg p, \neg q], [s, \neg p, \neg q], [v, \neg t, \neg u], \\ & [w, \neg v, \neg s], [t], [q], [u], [p], [\neg w]\} \end{aligned}$$

Aplikací popsaného postupu na program ze zadání obdržíme množinu klauzulí S a následující strom rezolučního vyvracení.



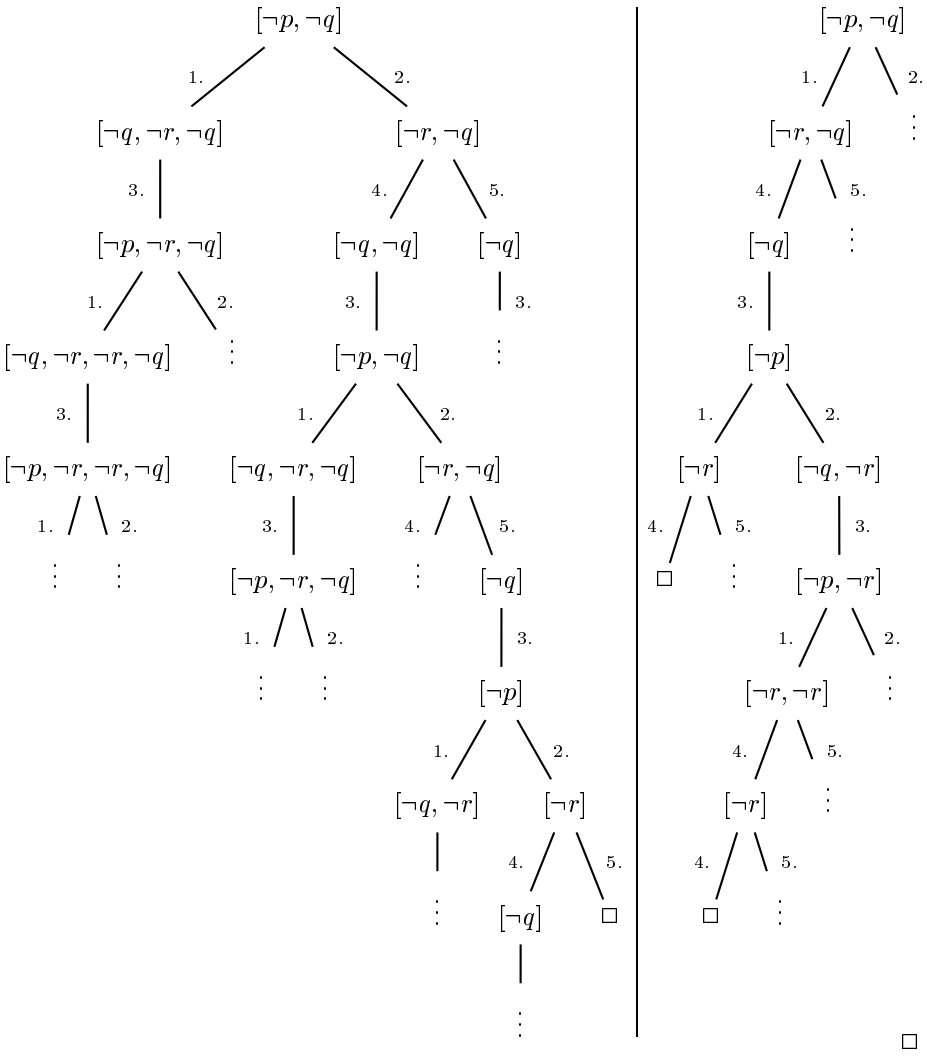
$\square$

Příklad 4.2: Vytvořte SLD-strom pro následující program (`Program 1.`) a dotaz v Prologu a zjistěte, jak se projeví změna pořadí klauzulí (`Program 2.`) v definici programu na výsledné podobě SLD-stromu.

```
Program 1:                          Program 2:
 1. p :- q,r.       4. r :- q.       1. p :- r.       4. r.
 2. p :- r.         5. r.            2. p :- q,r.     5. r :- q.
 3. q :- p.                          3. q :- p.

 ?- p,q.
```

Řešení 4.2: SLD-strom konstruujeme tak, že do kořene dosadíme dotaz Q a pro každou možnou klauzuli C ze vstupní množiny, kterou lze použít k rezoluci, přidáme podstrom, který má v kořeni rezolventu Q a C (hranu vedoucí z Q k tomuto podstromu označíme číslem programové klauzule C).

Protože pracujeme s množinou Hornových klauzulí, obsahuje každý uzel stromu cílovou klauzuli (tj. klauzuli obsahující pouze negativní literály).



Příklad 4.3: Sestrojte SLD-stromy pro následující programy a dotazy v jazyce Prolog:

```
Program 1:
 1. p :- a,r.  5. r :- t,a.
 2. a :- b.    6. r :- s.
 3. a.         7. s.
 4. b :- a.

 ?- p.
```

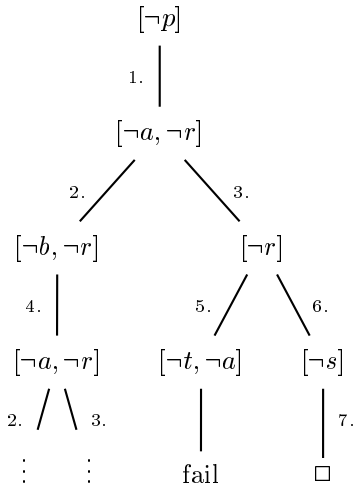
```
Program 2:
 1. p :- s,t.  5.  r :- w.
 2. p :- q.    6.  r.
 3. q.         7.  s.
 4. q :- r.    8.  t :- w.

 ?- p.
```

Řešení 4.3: Můžeme postupovat stejně jako při řešení předchozího příkladu. Programy převedeme na uspořádané klauzule, ze kterých poté konstruujeme samotný strom. Tento postup jsme použili pro `Program 1`.

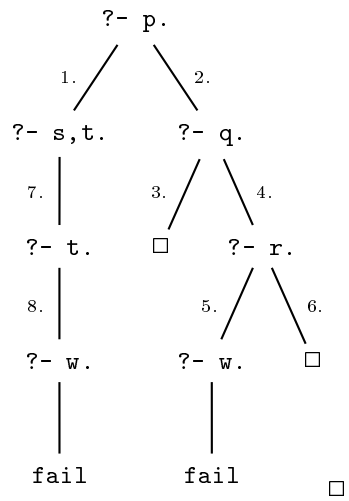
Druhou možností je použít při konstrukci SLD-stromu notaci jazyka Prolog. Místo uspořádaných klauzulí tak budeme do jednotlivých uzlů zapisovat

prologovské dotazy vzniklé aplikací rezolučního pravidla. Ušetříme si tak práci s přepisem pravidel na formule v CNF. Strom pak můžeme konstruovat velmi jednoduše tak, že první literál dotazu nahradíme pravou stranu programového pravidla, které používáme k rezoluci. Tuto metodu jsme použili pro konstrukci SLD-stromu pro `Program 2`.

`Program 1:`



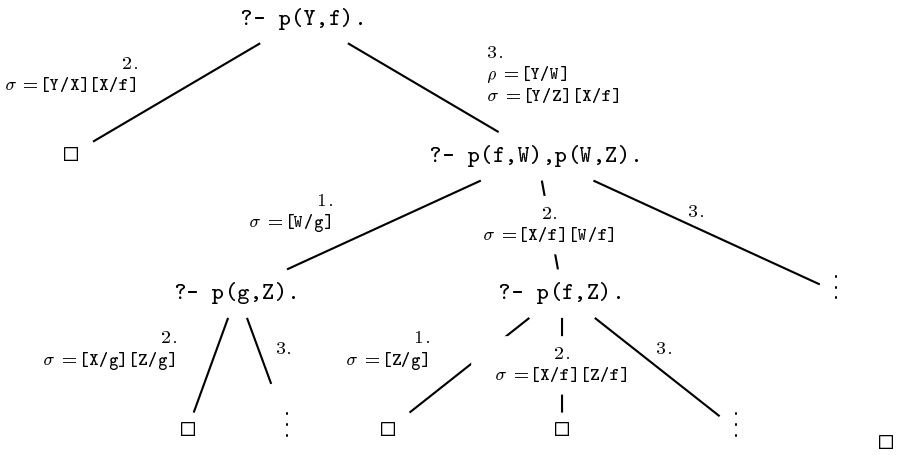
`Program 2:`



Příklad 4.4: Napište SLD-strom pro následující program a dotaz.

```
1. p(f,g).      3. p(Z,X) :- p(X,Y), p(Y,Z).
2. p(X,X).

?- p(Y,f).
```

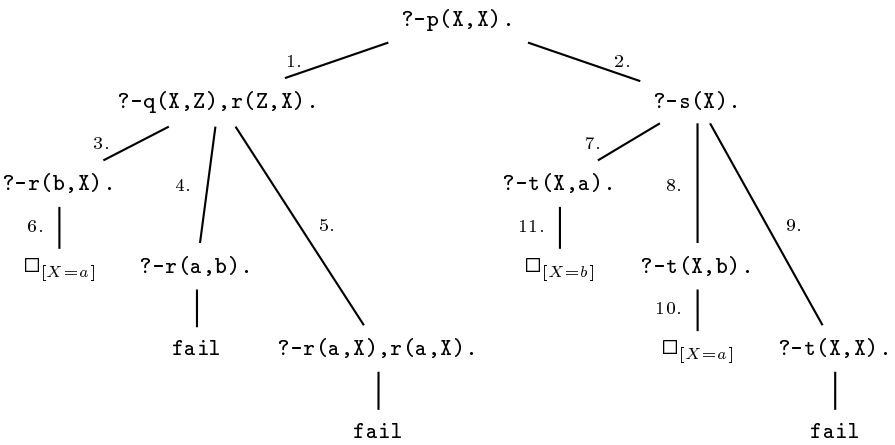
Řešení 4.4: Pro řešení využijeme druhý způsob zápisu. Program využívá predikátovou logiku, musíme proto použít rezoluční pravidlo pro tuto logiku. Budeme tedy opět přejmenovávat proměnné (substituce ρ - aplikuje se na programovou klauzuli, ale stejně tak bychom mohli přejmenovávat i proměnné v cílové klauzuli) v případě, že se vyskytnou v obou klauzulích, a hledat nejobecnější unifikátor literálů (substituce σ), na kterých budeme rezoluci provádět.



Příklad 4.5: Vytvořte SLD-strom všech možných řešení k následujícímu programu a dotazu:

```
1. p(X,Y) :- q(X,Z), r(Z,Y).      7. s(X) :- t(X,a).
2. p(X,X) :- s(X).                8. s(X) :- t(X,b).
3. q(X,b).                        9. s(X) :- t(X,X).
4. q(b,a).                       10. t(a,b).
5. q(X,a) :- r(a,X).             11. t(b,a).
6. r(b,a).

?- p(X,X).
```

Řešení 4.5: Sestrojíme následující SLD-strom v notaci jazyka Prolog.



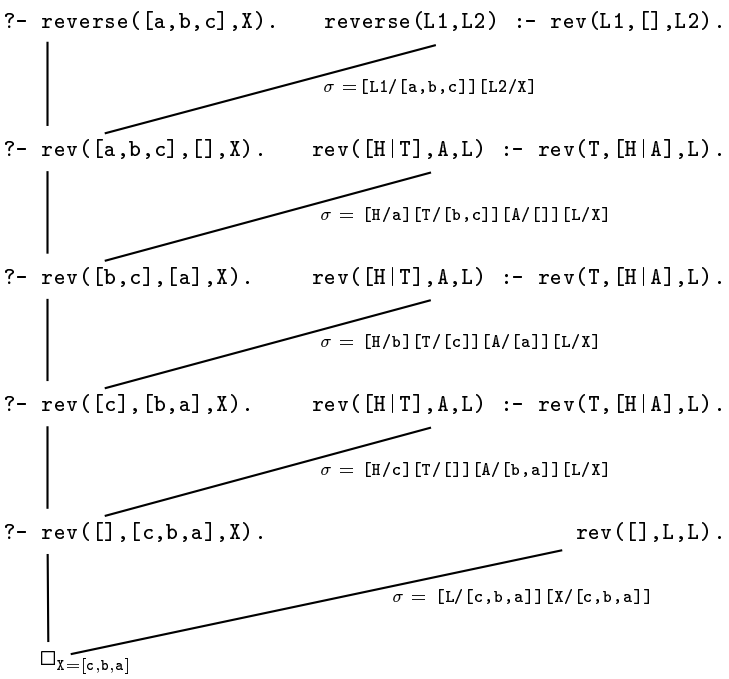
Ve stromu jsme pro přehlednost uvedli pouze čísla klauzulí z programu, které jsme použily při SLD-rezolučním vyvracení. Vynechali jsme tedy jak substituce použité pro přejmenování proměnných, tak nejobecnější unifikátory.

□

Příklad 4.6: Zkonstruujte SLD vyvrácení cíle `?- reverse([a,b,c],X).` za předpokladu, že máme nadefinován predikát `reverse/2` následovně:

```
reverse(L1,L2) :- rev(L1,[],L2).
rev([H|T],A,L) :- rev(T,[H|A],L).
rev([],L,L).
```

Řešení 4.6: Strom rezolučního vyvrácení dotazu `?- reverse([a,b,c],X).` vypadá následovně.



Zajímavé na tomto příkladu je zejména to, že jsme definovali program, který neřeší pouze rozhodovací problém (zda dotaz z programu vyplývá či nikoli), ale počítá novou hodnotu ze zadaných argumentů. Jedná se o program pro převrácení seznamu zadaného jako první argument.

Pokud na proměnnou `X` postupně aplikujeme všechny substituce v pořadí shora dolů (v tomto případě tedy pouze poslední substituci σ), získáváme substituci `X = [c,b,a]` uvedenou u prázdné klauzule. Tuto substituci nám Prolog také předloží jako odpověď na dotaz `?- reverse([a,b,c],X).`, tedy na dotaz, zda z uvedeného programu vyplývá formule `∃X reverse([a,b,c],X)`. □

5 Tabla ve výrokové logice

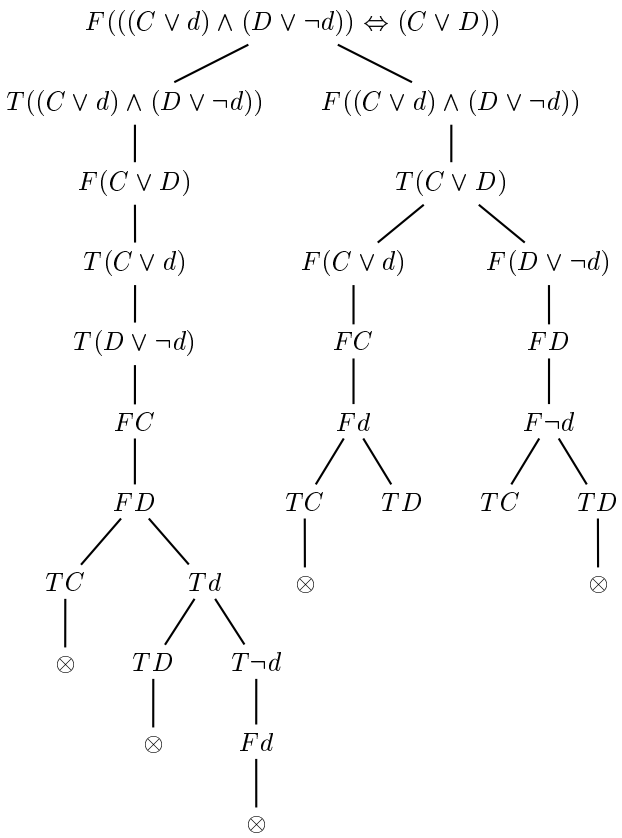
Příklad 5.1: Sestrojte *ukončené tablo* s kořenem

$$F(((C \vee d) \wedge (D \vee \neg d)) \Leftrightarrow (C \vee D)),$$

kde C, D, d jsou výrokové symboly.

Řešení 5.1: Ukončené tablo je strom, ve kterém je každá cesta *ukončená*. Cesta je ukončená, pokud je *sporná (kontradiktorická)* nebo je každý uzel ležící na této cestě *redukovaný*. Přitom cesta je *sporná* (značíme $\otimes$) právě tehdy, když se na ní vyskytují uzly $T\varphi$ a $F\varphi$ pro nějakou formuli φ. Uzel E na cestě P je *redukovaný*, pokud v atomickém tablu s kořenem E existuje cesta, jejíž všechny uzly se vyskytují i na cestě P. Obsahuje-li tablo neredukovaný uzel E ležící na cestě P, která není sporná, přidáme atomické tablo s kořenem E na konec cesty P.[8] Tímto postupem se časem dopracujeme k ukončenému tablu.

Kdybychom měli vybudovat *úplné systematické tablo*, tak je třeba vždy najít uzel E, který není na nějaké cestě redukovaný a je nejblíž kořenu (pokud je takových více, vybereme ten nejlevější). Atomické tablo s kořenem E pak přidáme na konec každé nesporné cesty procházející přes E, na které není E redukovaný.



Uvedené tablo není systematické, protože v jeho pravé části jsme redukovali uzly $F(C \vee d)$ a $F(D \vee \neg d)$ dříve než výše uvedený uzel $T(C \vee D)$.

Tablo obsahuje i cesty, které nejsou sporné. Lze z nich proto tedy vyčíst interpretaci splňující kořen tabla. Například nejlevější nesporná cesta v tablu obsahuje uzly TD, Fd, FC. Přiřadíme-li tedy symbolu D hodnotu 1 a symbolům

[8] Konvence: kořen E přidávaného atomického tabla se vypouští - v budovaném tablu se na cestě P již vyskytuje.

d a C hodnotu 0, získáváme valuaci, které nesplňuje formuli $((C \vee d) \wedge (D \vee \neg d)) \Leftrightarrow (C \vee D)$, takže vlastně splňuje předpoklad v kořeni tabla, že tato formule neplatí.

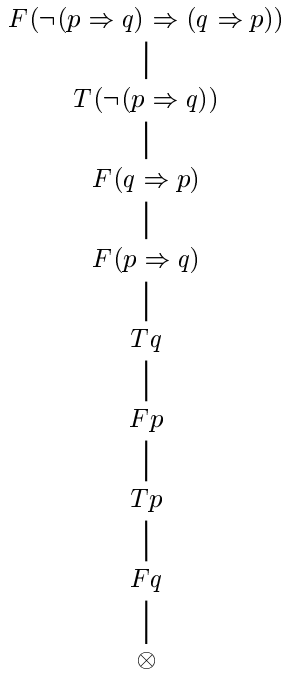
Všimněte si, že v levé části tabla jsou všechny cesty sporné. Tablo by obsahovalo pouze tuto levou část, pokud bychom v kořenu použili místo ekvivalence implikaci $\Rightarrow$. Tablo by pak bylo důkazem zmíněné implikace, která popisuje rezoluční pravidlo. □

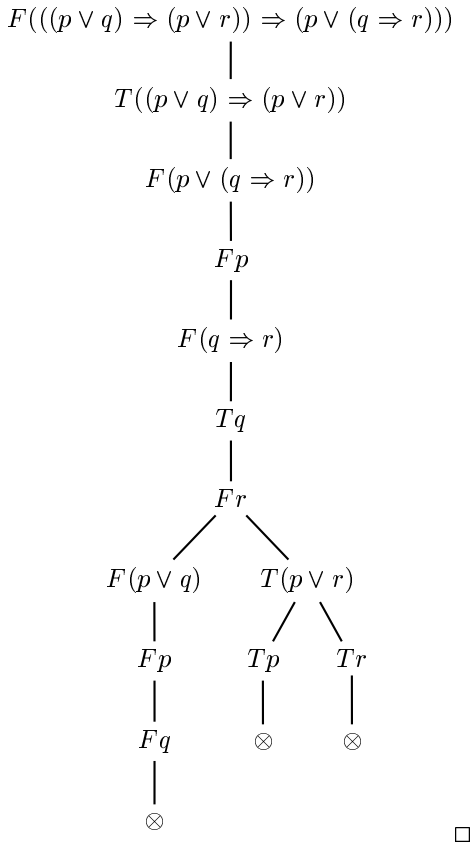
Příklad 5.2: Pomocí tabel dokažte, že následující formule jsou tautologické:

a) $\neg(p \Rightarrow q) \Rightarrow (q \Rightarrow p)$

b) $((p \vee q) \Rightarrow (p \vee r)) \Rightarrow (p \vee (q \Rightarrow r))$

Řešení 5.2: Pro formule vytvoříme ukončená kontradiktorická tabla následujícím způsobem:



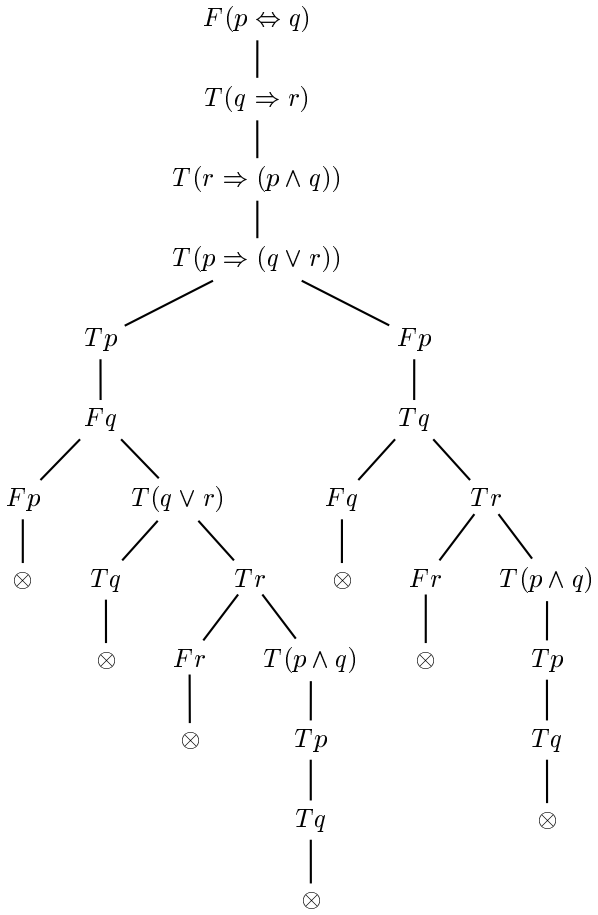


□

Příklad 5.3: Dokažte, že platí následující logické vyplývání:

$$\{q \Rightarrow r, r \Rightarrow (p \wedge q), p \Rightarrow (q \vee r)\} \models (p \Leftrightarrow q).$$

Řešení 5.3: Postupujeme stejně jako při řešení předchozího příkladu. Jediná změna je, že kdykoliv můžeme zvolit některý z předpokladů α a přidat $T\alpha$ na konec každé nesporné cesty, která $T\alpha$ zatím neobsahuje. Opět obdržíme uzavřené kontradiktorické tablo, čímž jsme dokázali, že formule $p \Leftrightarrow q$ logicky vyplývá z množiny předpokladů.



□

6 Tabla v predikátové logice

Příklad 6.1: Pomocí tabel dokažte, že následující formule jsou tautologie.

a) $\Phi_1 \equiv \forall x \varphi(x) \Rightarrow \exists x \varphi(x)$

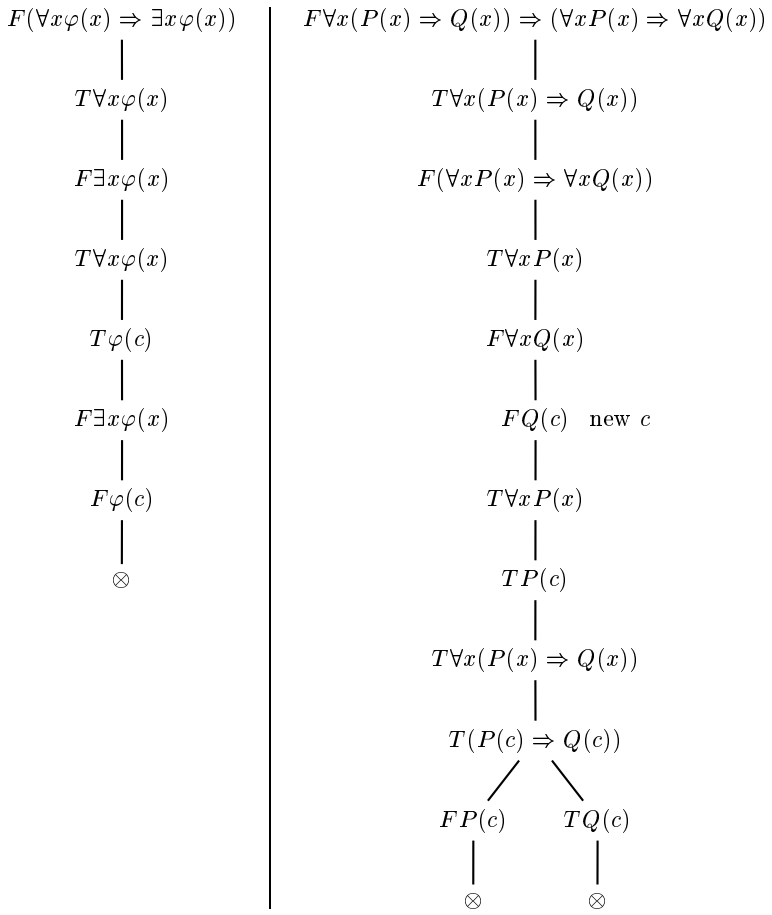
b) $\Phi_2 \equiv \forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall x P(x) \Rightarrow \forall x Q(x))$

c) $\Phi_3 \equiv \forall x(\varphi(x) \wedge \psi(x)) \Leftrightarrow (\forall x \varphi(x) \wedge \forall x \psi(x))$

d) $\Phi_4 \equiv \exists y \forall x(P(x,y) \Leftrightarrow P(x,x)) \Rightarrow \neg \forall x \exists y \forall z(P(z,y) \Leftrightarrow \neg P(z,x))$

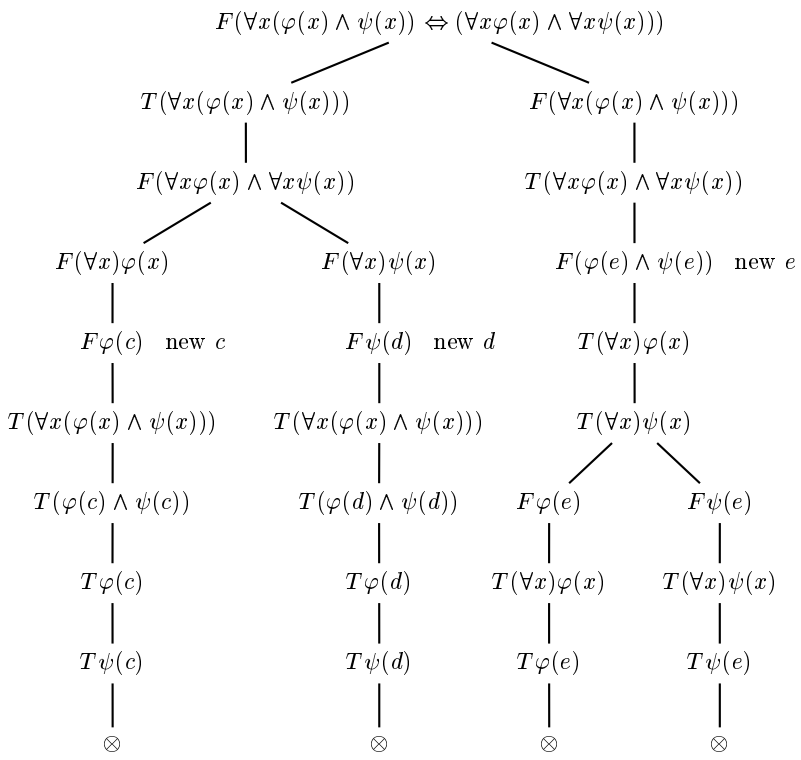
Řešení 6.1: Při konstrukci ukončeného tabla v predikátové logice postupujeme podobně jako v případě logiky výrokové. Redukci položky $T(\exists x)\varphi(x)$ (resp. $F(\forall x)\varphi(x)$) provedeme tak, že na konec cesty přidáme položku $T\varphi(c)$ (resp. $F\varphi(c)$), kde c je nová konstanta, která se nevyskytuje v žádném uzlu expandované cesty. Položku $T(\forall x)\varphi(x)$ (resp. $F(\exists x)\varphi(x)$) redukujeme na $T\varphi(t)$ (resp. $F\varphi(t)$), kde t je libovolný ground term (tj. term bez proměnných). Pozor, při redukci položek $T(\forall x)\varphi(x)$ a $F(\exists x)\varphi(x)$ nelze z přidávaného atomického tabla vynechat jeho kořen - je tedy nutné vždy zopakovat redukovanou položku.

Ukončená tabla pro formule Φ_1 (vlevo) a Φ_2 (vpravo):



Tablo je *ukončené*, pokud je každá cesta v tablu ukončená. Cesta je *ukončená*, pokud je sporná nebo je každý uzel ležící na této cestě redukovaný. Cesta je sporná, pokud se na ní vyskutuje $T\alpha$ a $F\alpha$ pro nějakou uzavřenou formuli α. Uzel odpovídající i-tému výskytu položky $T(\forall x)\varphi(x)$ na cestě P je redukovaný, pokud se na cestě vyskytuje položka $T\varphi(t_i)$ a $(i+1)$-ní výskyt položky $T(\forall x)\varphi(x)$, kde $t_1, t_2, \ldots, t_n, \ldots$ jsou všechny ground termy. Analogicky je definovaná redukovanost položek $F(\exists x)\varphi(x)$. Položka jiného typu je na cestě P redukovaná, pokud bylo při tvorbě tabla přidáno atomické tablo pro tuto položku k prefixu cesty P.

Ukončené tablo pro formuli Φ_3:

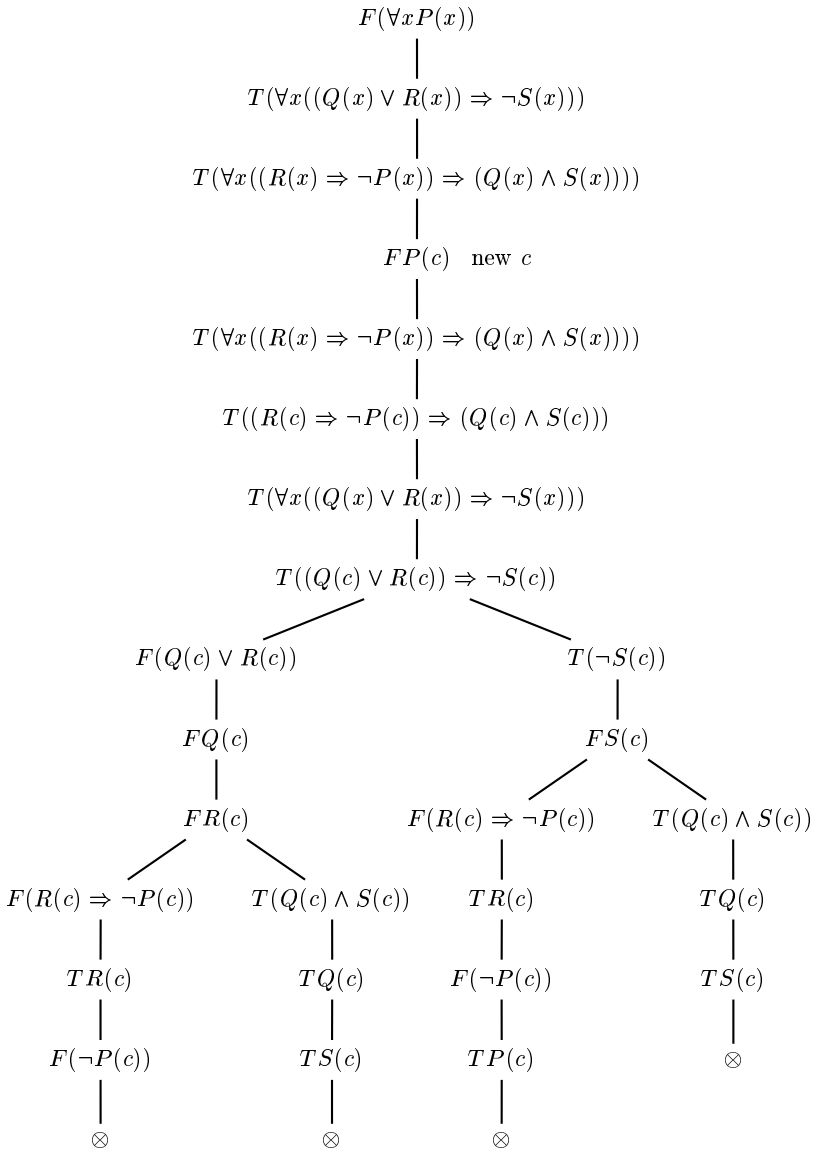


Řešení posledního příkladu ponecháváme na čtenáři. □

Příklad 6.2: Dokažte, že formule $\forall x P(x)$ logicky vyplývá z předpokladů:

$$\forall x((Q(x) \vee R(x)) \Rightarrow \neg S(x))$$
$$\forall x((R(x) \Rightarrow \neg P(x)) \Rightarrow (Q(x) \wedge S(x)))$$

Řešení 6.2: Dokazování platnosti logických úsudků (formule závěru ψ logicky vyplývá z předpokladů Σ) provádíme stejně jako v případě logiky výrokové.



□

Příklad 6.3: Pomocí tabel dokažte platnost následujících tvrzení.

a) Předpokládejte, že platí následující tři tvrzení:

- Existuje drak (označme $D/1$).
- Draci spí ($S/1$) nebo loví ($L/1$).
- Když jsou draci hladoví ($H/1$), tak nespí.

Důsledek: *Když jsou draci hladoví, tak loví.*

b) Předpokládejte, že platí následující dvě tvrzení:

- Všichni holiči ($B/1$) holí ($S/2$) každého, kdo se neholí sám.
- Žádný holič neholí někoho, kdo se holí sám.

Důsledek: *Holiči neexistují.*

Řešení 6.3: Úsudky nejprve převedeme z vět přirozeného jazyka na formule predikátové logiky prvního řádu. Tablový důkaz poté konstruujeme nad získanými formulemi stejně jako v předchozím příkladě (v tablech již neznačíme nové konstanty).

a) převodem úsudku o spících dracích obdržíme formule:

$$\begin{aligned} \text{Předpoklady:} \quad & \exists x D(x), \\ & \forall x(D(x) \Rightarrow (S(x) \vee L(x))), \\ & \forall x((D(x) \wedge H(x)) \Rightarrow \neg S(x)), \\ \text{Závěr:} \quad & \forall x((D(x) \wedge H(x)) \Rightarrow L(x)). \end{aligned}$$

Tablový důkaz úsudku je uveden na obrázku 1.

b) Formule získané převodem na formule predikátové logiky prvního řádu jsou:

$$\begin{aligned} \text{Předpoklady:} \quad & \forall x \forall y((B(x) \wedge \neg S(y,y)) \Rightarrow S(x,y)), \\ & \forall y(S(y,y) \Rightarrow \neg \exists x(B(x) \wedge S\ (x,y))), \\ \text{Závěr:} \quad & \neg \exists x B(x). \end{aligned}$$

Tablo dokazující platnost úsudku je uvedena na obrázku 2. Přitom ukončené kontradiktorické tablo získáme nahrazením uzlu * podstromem uzlu $TS(c,c)^*$.

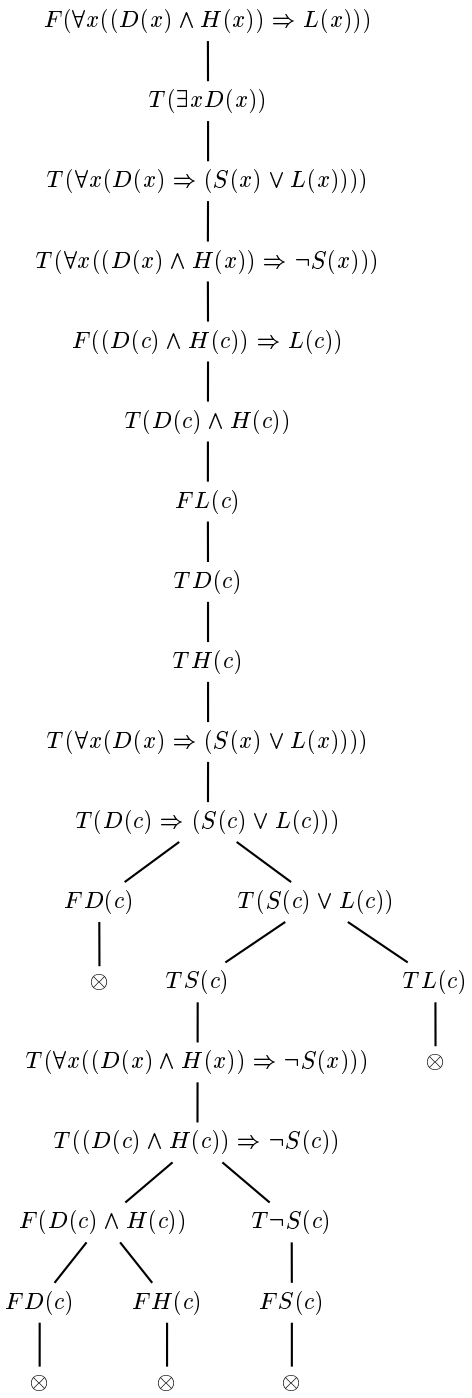
□

7 Tabla v modální logice

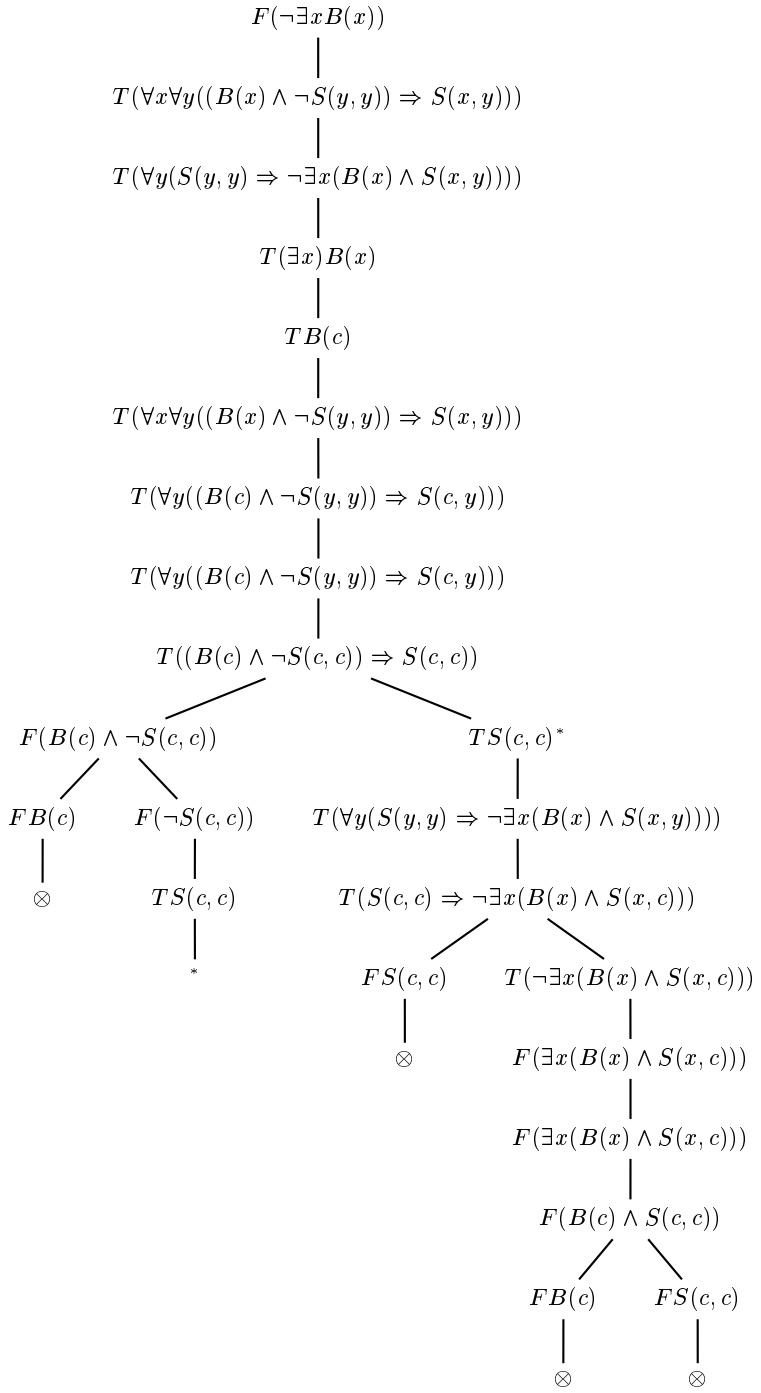
Při konstrukci ukončeného kontradiktorického tabla v modální logice postupujeme podobně jako v případě logiky prvního řádu. Máme-li dokázat, že formule φ je tautologií (tj. platí ve všech světech všech Kripkeho rámců nad použitým jazykem modální logiky), stačí zkonstruovat kontradiktorické tablo s kořenem $Fw \Vdash \varphi$. Každý uzel vytvářeného binárního stromu pak obsahuje výraz $Fv \Vdash \varphi$ nebo $Tv \Vdash \varphi$. Narozdíl od logiky prvního řádu je tedy nutné brát v úvahu také svět, ve kterém danou položku redukujeme.

Cesta v tablu je *sporná (kontradiktorická)*, pokud se na ní vyskytuje položka $Tv \Vdash \varphi$ a zároveň $Fv \Vdash \varphi$ pro nějaký svět v a formuli φ.

Kvůli technickému zjednodušení přijímáme konvenci, že jazyk modální logiky neobsahuje funkční symboly. Při expanzi položek $Tv \Vdash (\forall x)\varphi(x)$ a $Fv \Vdash (\exists x)\varphi(x)$ proto místo ground termů používáme pouze konstanty. Musíme však volit takové konstanty, o kterých víme, že se vyskytují ve světě v. Taková konstanta je buď součástí jazyka modální logiky, nebo se na expandované cestě již vyskytuje v nějaké položce týkající se světa v nebo nějakého jeho předchůdce (tj. světa w, pro který se na cestě vyskytuje wSv). Další možností je naopak volit konstantu, která se nevyskytuje nikde v celém tablu. Při expanzi položek $Tv \Vdash (\exists x)\varphi(x)$



Obrázek 1: Tablový důkaz úsudku o spících dracích.



Obrázek 2: Tablový důkaz úsudku o holičích.

a $Fv \Vdash (\forall x)\varphi(x)$ dosadíme namísto x libovolnou konstantu, která se dosud nevyskytuje v žádné položce tabla.

Položky s operátory $\Box$ a $\Diamond$ redukujeme následovně. Při redukci položky $Tv \Vdash \Diamond\varphi$ nebo $Fv \Vdash \Box\varphi$ na cestu nejdříve přidáme výraz vSw, kde w je nějaký nový svět, který ještě v tablu nebyl použit. Poté přidáme nový uzel $Tw \Vdash \varphi$ resp. $Fw \Vdash \varphi$. Položky typu $Tv \Vdash \Box\varphi$ a $Fv \Vdash \Diamond\varphi$ expandujeme na $Tw \Vdash \varphi$ a $Fw \Vdash \varphi$, kde w je libovolný svět, pro který je na expandované cestě položka vSw. Pokud žádný takový svět neexistuje, můžeme položky považovat za redukované.

Kořeny atomických tabel pro položky $Tv \Vdash (\forall x)\varphi(x)$, $Fv \Vdash (\exists x)\varphi(x)$, $Tv \Vdash \Box\varphi$ a $Fv \Vdash \Diamond\varphi$ bychom neměli při redukci vynechávat.

Příklad 7.1: Pomocí tabel dokažte, že následující formule jsou tautologie.

a) $\Phi_1 \equiv (\Box\forall x\varphi(x)) \Rightarrow (\forall x\Box\varphi(x))$

b) $\Phi_2 \equiv (\Box(\varphi \Rightarrow \psi)) \Rightarrow ((\Box\varphi \Rightarrow \Box\psi))$

c) $\Phi_3 \equiv \neg\Diamond(\neg(\varphi \wedge \exists x\psi(x)) \wedge \exists x(\varphi \wedge \psi(x)))$, x není volné ve formuli φ

d) $\Phi_4 \equiv \Diamond\exists x(\varphi(x) \Rightarrow \Box\psi) \Rightarrow \Diamond(\forall x\varphi(x) \Rightarrow \Box\psi)$, x není volné ve formuli ψ

Řešení 7.1: Viz obrázky 3 a 4.

□

Příklad 7.2: Mějme modální tablo s kořenem $Fw \Vdash \forall x\Box\varphi(x) \Rightarrow \Box\forall x\varphi(x)$ definované v obrázku 5. Rozhodněte o korektnosti uvedeného důkazu a své tvrzení zdůvodněte.

Řešení 7.2: Zadané tablo není (v naší sémantice modální logiky) korektně utvořené a nedokazuje tak pravdivost formule.

Chyba spočívá ve špatném použití atomického tabla pro operátor $\forall$, ve kterém jsme substituovali proměnnou x nevhodnou konstantou c. Při expanzi položky označené symbolem "*" nemůžeme nahradit proměnnou x konstantou c. Konstantu c jsme do tabla zavedli v kontextu možného světa v, zatímco při expanzi položky * se pohybujeme ve světě w. Expanze by byla správná, pokud by byl svět w následníkem světa v a nikoliv jeho předchůdcem.

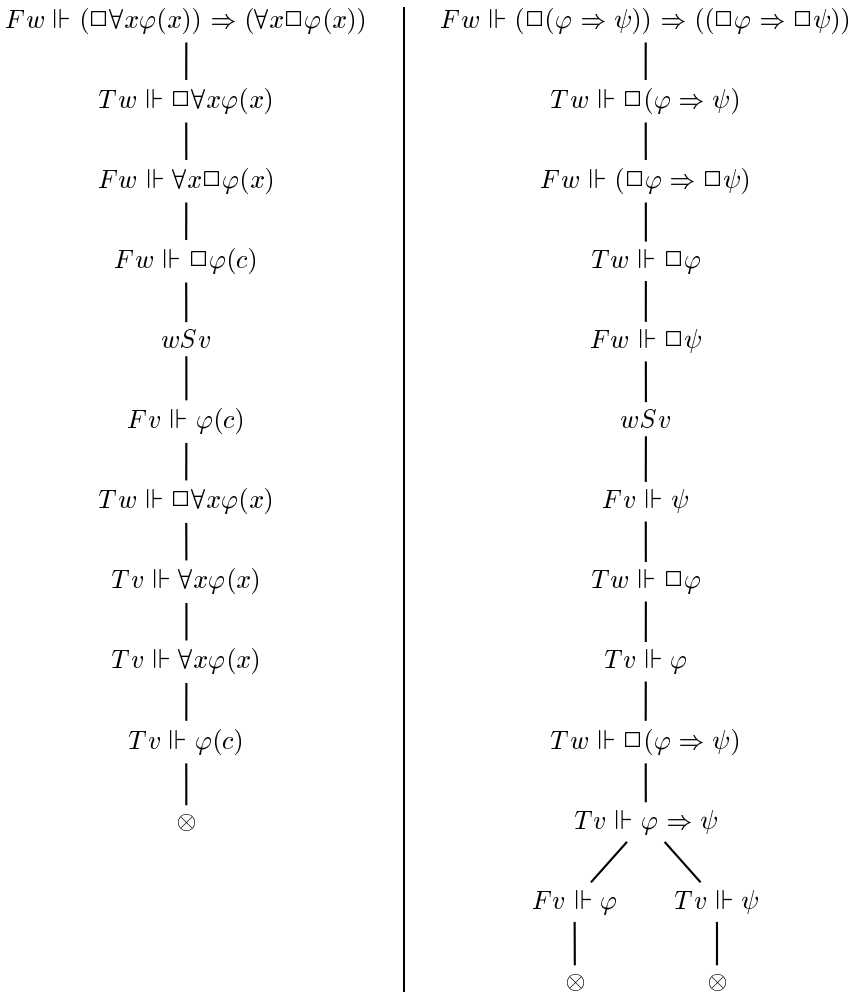
Příklad korektního ukončeného tabla s daným kořenem je na obrázku 6. V modální logice vždy předpokládáme, že jazyk logiky obsahuje alespoň jednu konstantu (nechť se jmenuje c_1). Tato konstanta pak musí být ve všech světech a proto ji můžeme použít v atomickém tablu pro $Tw \Vdash \forall x\Box\varphi(x)$. Tablo na obrázku 6 je nekonečné (aby byla položka $Tw \Vdash \forall x\Box\varphi(x)$ redukovaná, musí se nekonečně-krát opakovat). Protože tablo obsahuje cestu, která není sporná, lze z něj vyčíst Kripkeho strukturu se světem w, v němž formule v kořeni neplatí. Tato struktura obsahuje svět w s jedním následníkem v. Ve světě w platí $\forall x\Box\varphi(x)$, ale ve světě v je konstanta c taková, že $\varphi(c)$ ve v neplatí. Tudíž ve světe w neplatí formule v kořeni tabla. □

Příklad 7.3: Dokažte platnost úsudků:

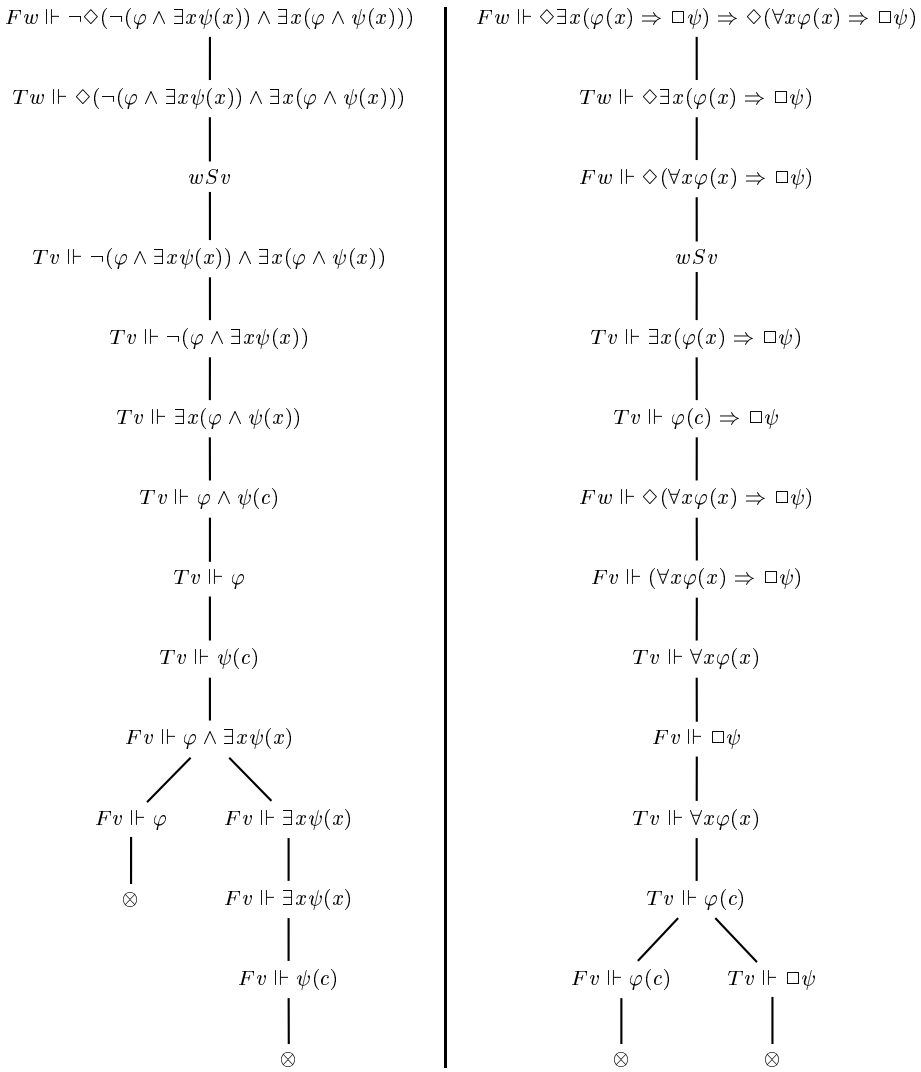
a) $\{\varphi\} \models \Box\varphi$

b) $\{\forall x\varphi(x)\} \models \Box\forall x\varphi(x)$

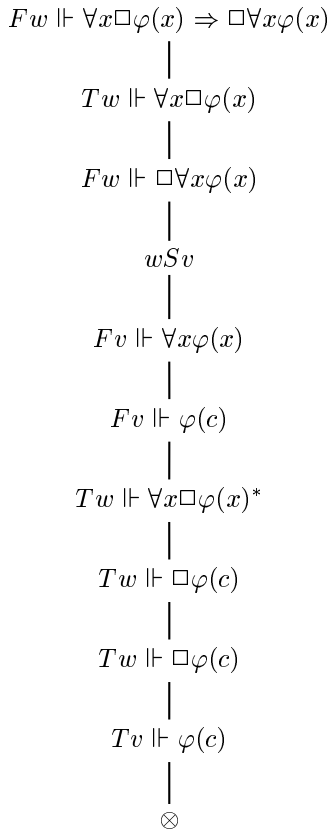
c) $\{\forall x\varphi(x)\} \models \forall x\Box\varphi(x)$



Obrázek 3: Ukončená kontradiktorická tabla pro Φ_1 (vlevo) a Φ_2 (vpravo).



Obrázek 4: Ukončená kontradiktorická tabla pro Φ_3 (vlevo) a Φ_4 (vpravo).



Obrázek 5:

d) $\{\varphi \Rightarrow \Box\varphi\} \models \Box\varphi \Rightarrow \Box\Box\varphi$

Řešení 7.3: Tablo pro logický úsudek $S \models \varphi$ konstruujeme stejně jako tablo pro φ, ale navíc můžeme kdykoliv na konec nějaké cesty P přidat položku tvaru $Tv \Vdash \alpha$, kde v je nějaký svět vyskytující se na cestě P a $\alpha \in S$.

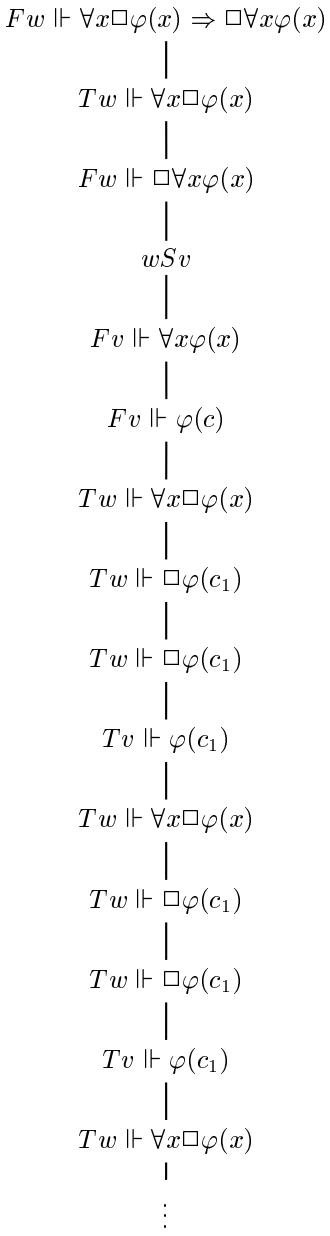
Ukončená kontradiktorická tabla pro úsudky (a), (b) a (c) jsou uvedena na obrázku 7. Tablo pro úsudek (d) ponecháváme jako cvičení. □

8 Induktivní logické programování

Příklad 8.1: Jsou dány výrokové symboly a, b, c, d, e a tabulka

a	b	c	d	e	*třída*
1	1	1	0	1	+
1	1	0	1	1	+
1	1	0	1	0	−

+ = pozitivní příklady
− = negativní příklad

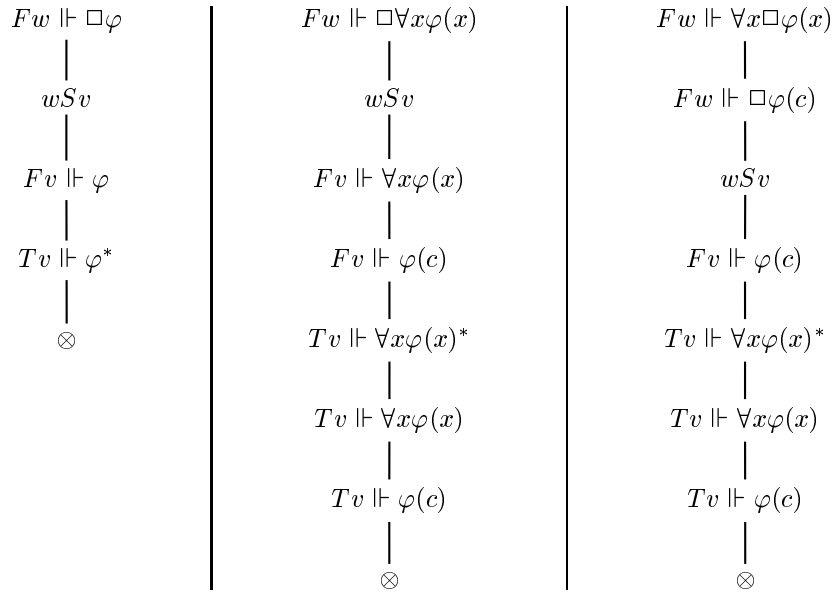


Obrázek 6:

Najděte všechny specializace výrokové formule b. Vyznačte, které z nich pokrývají negativní příklad.

Řešení 8.1: Ve výrokovém počtu používáme pouze jeden specializační operátor a tím je přidání pozitivního literálu pomocí konjunkce.[9] Můžeme tak nyní defi-

[9] Lze zavést i další specializační operátory, např. přidání negativního literálu pomocí konjunkce. Naše omezení na jeden zmíněný specializační operátor má za následek, že není obecně



Obrázek 7: Ukončená kontradiktorická tabla pro úsudky (a),(b) a (c). Položky odpovídající aplikaci předpokladů jsou označeny hvězdičkou.

novat specializační operátor ρ pro jazyk $\mathcal{L}$ tak, že

$$\rho(\varphi) = \{(\varphi \wedge l) \mid l \in \mathcal{L} \text{ a } l \text{ se nevyskytuje ve } \varphi\}$$

Jinými slovy, specializace formule φ (v našem případě $\varphi = b$) jsou všechny formule, které vzniknou přidáním některých prvků jazyka $\mathcal{L}$ k formuli φ (ze zadání je $\mathcal{L} = \{a, b, c, d, e\}$). Dále budeme výrazem $\rho^i(\varphi)$ označovat množinu specializací získaných z φ po i aplikacích operátoru ρ.

Aplikací operátoru ρ na formuli b tak obdržíme naslédující specializace (při zápisu vynecháváme spojky $\wedge$ a abstrahujeme od pořadí literálů v konjunkci):

$$\begin{aligned}
\rho^0(b) &= \{b\} \\
\rho^1(b) &= \{ab, bc, bd, be\} \\
\rho^2(b) &= \{abc, abd, abe, bcd, bce, bde\} \\
\rho^3(b) &= \{abcd, abce, abde, bcde\} \\
\rho^4(b) &= \{abcde\}
\end{aligned}$$

Všechny specializace formule b (značíme $\rho^*(b)$) pak obdržíme jako sjednocení $\rho^1(b) \cup \rho^2(b) \cup \rho^3(b) \cup \rho^4(b)$.

Nyní můžeme určit, které specializace pokrývají negativní příklad následovně. Definici každého příkladu z tabulky ze zadání můžeme chápat jako definici interpretace I. Ověření, zda daná formule φ pokrývá příklad e pak spočívá v ověření, zda interpretace daná e splňuje φ (můžeme zapsat jako $I_e(\varphi) = 1$). Je zřejmé, že pro interpretaci danou definicí negativního příkladu mohou být pravdivé pouze podformule formule $a \wedge b \wedge d$. Zajímají nás pak jen ty z $\rho^*(b)$, tedy ab, bd, abd. □

zaručeno nalezení formule pokrývající všechny pozitivní a žádný negativní příklad.

Příklad 8.2: Napište část specializačního grafu s kořenem `neteř(X,Y)`, která bude obsahovat klauzuli:

```
neteř(X,Y) :- žena(X), sourozenec(Y,Z), rodič(Z,X).
```

Řešení 8.2: Specializační graf vzniká z daného kořene tak, že kořen propojíme hranou s každou klauzulí, která vznikla z kořene aplikací specializačního operátoru. Postup pak opakujeme pro takto vzniklé klauzule.

Celkem používáme čtyři základní specializační operátory:

ztotožnění proměnných - např. specializací `p(X,Y).` dostáváme `p(X,X).`

přidání literálu do těla klauzule - přidávaným literálem je predikát z doménové znalosti. Přidávaný literál obsahuje čerstvé proměnné. Např. specializací `p(X,Y).` získáváme `p(X,Y):-q(U,V,W).`

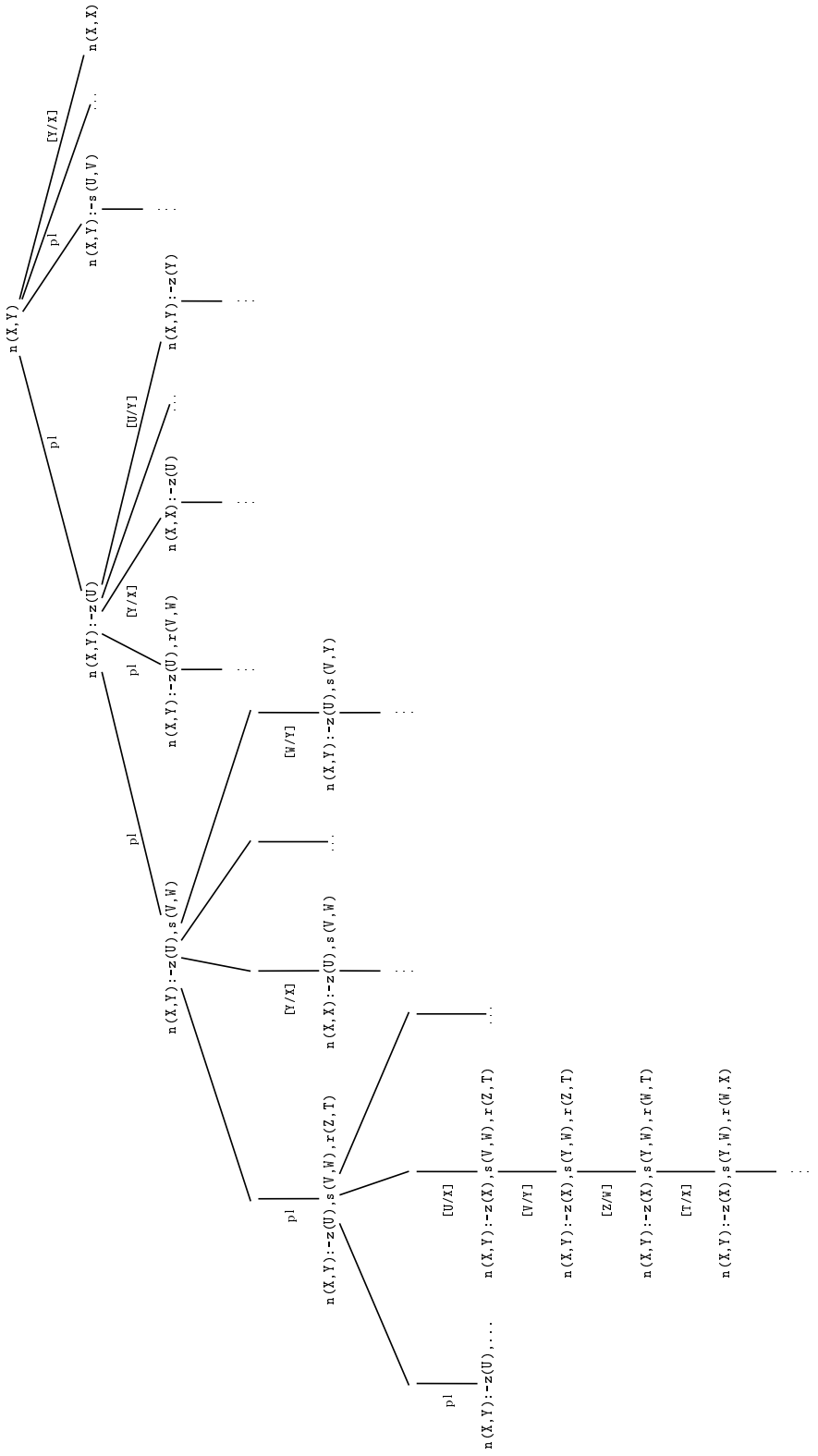
dosazení konstanty za proměnnou - dozasujeme konstanty (nulární funkční symboly) z doménové znalosti. Např. specializací `p(X,Y).` dostáváme `p([],X).`

dosazením nejobecnějšího termu za proměnnou - nejobecnějším termem se rozumí funkční symbol (z doménové znalosti) aplikovaný na čerstvé proměnné. Např. specializací `p(X,Y).` získáváme `p([U|V],Y).`

Předpokládáme, že doménová znalost v našem příkladu obsahuje predikáty `neteř(X,Y)`, `rodič(X,Y)`, `sourozenec(X,Y)` a `žena(X)`. Pro jejich zápis budeme dále používat pouze první písmena.

Řešením našeho příkladu je graf uvedený na obrázku 8 (hrany odpovídající přidání literálu jsou označené písmeny "pl"). Jelikož naše doménová znalost neobsahuje žádný funkční symbol, uvedený graf obsahuje specializace vzniklé buď přidáním literálu nebo ztotožněním proměnných. Do řešení jsme navíc nezahrnuli specializace vytvořené přidáním literálu `neteř(X,Y)`, protože tyto specializace nevedly ke klauzuli ze zadání. Pokud bychom však chtěli vytvořit úplný specializační graf, museli bychom tyto specializace také uvádět.

Všimněte si, že klauzule ze zadání se v řešení vyskytuje v mírně pozměněné podobě: místo proměnné `Z` je použita proměnná `W`. Tento rozdíl není na závadu, neboť pojmenování proměnných nemá vliv na význam uvedené programové klauzule. □



Obrázek 8: Specializační graf pro predikát `neteř(X,Y)`.