Matematická logika

přednáška desátá

Miroslav Kolařík

Zpracováno dle textu R. Bělohlávka:
Matematická logika – poznámky k přednáškám, 2004.

Obsah

Motivace

Klasická logika nemá prostředky k formalizaci tvrzení obsahujících modality, například „je možné, že . . . “, „je nutné, že . . . “. Rozšíření klasické logiky, kde toto je možné se nazývá **modální logika**. ML našla uplatnění například ve formalizaci znalostních systémů a systémů, které pracují s časem.

Základy syntaxe a sémantiky ML

Zaměříme se na výrokovou ML. Oproti jazyku klasické VL obsahuje jazyk ML navíc unární spojky $\Box$ a $\Diamond$ ($\Box\varphi$ se čte „je nutné, že φ“ a $\Diamond\varphi$ se čte „je možné, že φ“). Definice formule se příslušným způsobem rozšíří, tj. přidáme pravidlo „je-li φ formule, jsou i $\Box\varphi$ a $\Diamond\varphi$ formule“. Poznamenejme, že konkrétní význam $\Box\varphi$ může být „je známo, že φ“, „věří se, že φ“, „vždy v budoucnosti bude platit φ“ apod.

Sémantika ML je založena na pojmu **možný svět**. Možný svět je obecná kategorie (v jednom možném světě může v daný okamžik pršet, ve druhém ne, apod.), která má řadu interpretací. Možné světy mohou být časové okamžiky; mohou reprezentovat názory jednotlivých expertů (co možný svět, to expert) apod. Speciální interpretací světů získáme **logiku času** (**temporální logika**), což je logika zabývající se tvrzeními, jejichž pravdivostní hodnota závisí na čase. **Logika znalostí** (**epistemická logika**) se zabývá spojkami „ví se, že . . . “, „věří se, že . . . “ apod.

Definice

Kripkeho struktura pro výrokovou ML je trojice $\mathbf{K} = \langle W, e, r \rangle$, kde $W \neq \emptyset$ je množina možných světů, e je zobrazení přiřazující každému $w \in W$ a každému výrokovému symbolu p pravdivostní hodnotu $e(w, p)$ (p je/není pravdivé ve w), $r \subseteq W \times W$ je relace dosažitelnosti ($\langle w, w' \rangle \in r$ znamená, že z w je možné se dostat do w').

Pro $r = W \times W$ (každý svět je dosažitelný z každého) se příslušná logika nazývá **logika znalostí**. Platí-li pro nějaké $W' \subseteq W$, že $r = W \times W'$, nazývá se příslušná logika **logika domění**.

Definujme nyní pravdivostní hodnotu $\| \varphi \|_{\mathbf{K},w}$ formule φ v $\mathbf{K}$ v možném světě w takto: $\| p \|_{\mathbf{K},w} = e(w,p)$ (pro výrokový symbol p); $\| \varphi \wedge \psi \|_{\mathbf{K},w} = 1$, právě když $\| \varphi \|_{\mathbf{K},w} = 1$ a $\| \psi \|_{\mathbf{K},w} = 1$ (tedy jako v klasické logice) a podobně pro ostatní výrokové spojky; $\| \Box\varphi \|_{\mathbf{K},w} = 1$, právě když $\| \varphi \|_{\mathbf{K},v} = 1$ pro každý $v \in W$, pro který $\langle w,v \rangle \in r$ (tj. „je nutné, že φ“ znamená, že φ je pravdivá v každém možném světě dosažitelném z w); $\| \Diamond\varphi \|_{\mathbf{K},w} = 1$, právě když $\| \varphi \|_{\mathbf{K},v} = 1$ pro nějaký $v \in W$, pro který $\langle w,v \rangle \in r$ (tj. „je možné, že φ“ znamená, že φ je pravdivá v nějakém možném světě dosažitelném z w). Poznamenejme ještě, že $\Box\varphi$ a $\neg\Diamond\neg\varphi$ (a $\Diamond\varphi$ a $\neg\Box\neg\varphi$) mají stejné pravdivostní hodnoty. Tedy stačí zavést jen jednu spojku a druhou brát jako odvozenou.

Poznámka: Je-li r reflexivní, tranzitivní a platí-li, že pro každé $v, w \in W$ je $\langle v, w \rangle \in r$ nebo $\langle w, v \rangle \in r$, pak se odpovídající logika nazývá logika času (temporální logika) a $w \in W$ se chápou jako časové okamžiky. (Existují i jiné systémy logiky času.)
Pak tedy $\| \Box\varphi \|_{\mathbf{K},w} = 1$ znamená, že φ je pravdivá ve všech časových okamžicích počínaje w. Pro r^{-1} pak $\| \Box\varphi \|_{\mathbf{K},w} = 1$ znamená, že φ je pravdivá ve všech časových okamžicích až po w. Podobně pro $\| \Diamond\varphi \|_{\mathbf{K},w} = 1$.

Obsah

Logické programování je jedním z paradigmat programování. Uveďme nyní základní rysy, kterými se logické programování zásadně odlišuje od ostatních programovacích paradigmat:

(1) programátor specifikuje, **co** se má vypočítat, a ne **jak** se to má vypočítat a **kam** uložit mezivýsledky

(2) Prolog nemá příkazy pro řízení běhu výpočtu ani pro řízení toku dat, nemá příkazy cyklů, větvení, přiřazovací příkaz

(3) neexistuje rozdělení proměnných na vstupní a výstupní

(4) nerozlišuje se mezi daty a programem.

Prolog je logický programovací jazyk. Vznikl ve Francii (Marseille) začátkem 70. let. Byl vytvořen Alainem Colmeranerem ve spolupráci s Philippem Rousselem, na základě procedurálního výkladu Hornovy klauzule od Roberta Kowalskiho. Jméno „Prolog“ vzniklo jako zkratka z „PROgrammation à LOGic“ (francouzsky „programování v logice“).

Prolog je interpretační (neprocedurální) jazyk. Patří mezi deklarativní programovací jazyky – potlačuje imperativní složku. (Připomeňme, že imperativní paradigma popisuje **jak** vyřešit problém, deklarativní paradigma popisuje **co** je problém.)

Prolog je využíván především v oboru umělé inteligence a v počítačové lingvistice (obzvláště zpracování přirozeného jazyka, pro nějž byl původně navržen).

Prolog je založen na PL (prvního řádu); konkrétně se omezuje na Hornovy klauzule. Základními využívanými přístupy jsou unifikace, rekurze a backtracking.

Program a výpočet v logickém programování

Základní koncepci logického programování vyjadřuje následující dvojice „rovností“:

$$\begin{aligned} \text{program} &= \text{množina axiomů} \\ \text{výpočet} &= \text{konstruktivní důkaz uživatelem zadaného cíle,} \end{aligned}$$

nebo volněji: program je souborem tvrzení, kterými programátor (uživatel, expert) popisuje určitou část okolního světa; výpočet nad daným programem, který je iniciován zadáním dotazu, je hledání důkazu dotazu z daného souboru tvrzení.

Shrňme a doplňme základní rysy logického programování:

- logický program je konečná množina formulí speciálního tvaru
- výpočet je zahájen zadáním formule – dotazu (tu zadá uživatel)
- cílem výpočtu je najít důkaz potvrzující, že dotaz logicky vyplývá (je dokazatelný) z logického programu (konstruktivnost)
- pokud je takto zjištěno, že dotaz z programu vyplývá, výpočet končí a uživateli je oznámeno „Yes“ s hodnotami případných proměnných, které se v dotazu vyskytují
- pokud není zjištěno, že dotaz z programu vyplývá, výpočet končí a uživateli je oznámeno „No“
- může se stát, že výpočet neskončí.

Poznámka: Programátor není zbaven zodpovědnosti za to, jak bude výpočet probíhat. Výpočet (tj. logické dokazování) je řízen prologovským překladačem a programátor musí znát pravidla (alespoň chce-li psát efektivní programy), kterými se výpočet řídí a v souladu s nimi program v Prologu vytvářet.

Poznámka: Základem Prologu je databáze klauzulí, které lze dále rozdělit na fakta a pravidla, nad kterými je možno klást dotazy formou tvrzení, u kterých Prolog zhodnocuje jejich pravdivost (dokazatelnost z údajů obsažených v databázi). Nejjednoduššími klauzulemi jsou fakta, která pouze vypovídají o vlastnostech objektu nebo vztazích mezi objekty. Složitějšími klauzulemi jsou pravidla, která umožňují (pomocí implikace) odvozovat nová fakta. Zapisují se ve tvaru hlavička :- tělo, kde hlavička definuje odvozovaný fakt, tělo podmínky, za nichž je pravdivý, obsahuje jeden či více cílů. Pokud se interpretu podaří odvodit, že tělo je pravdivé, ověřil tím pravdivost hlavičky.

Ukázka prologovského programu: viz přednáška.

Obsah

Definice

Literál je libovolná atomická formule (tzv. pozitivní literál) nebo negace atomické formule (tzv. negativní literál). **Klauzule** je libovolná disjunkce literálů. **Hornovská klauzule** je klauzule, ve které se vyskytuje maximálně jeden pozitivní literál. Symbolem $\square$ se označuje prázdná klauzule (tj. klauzule obsahující 0 literálů).

Poznámka: $\square$ je v logickém programování symbolem sporu.

Poznámka: Klauzule $L_1 \vee \cdots \vee L_n$ (L_i jsou literály) se v logickém programování často zapisují jako $\{L_1, \ldots, L_n\}$; zde čárky znamenají disjunkci. Pak $\square$ se značí $\{\}$.

Jak brzy uvidíme, pracuje Prolog následovně: Je-li dán logický program P a zadá-li uživatel dotaz G (popř. obecněji $G_1, \ldots, G_n$), překladač Prologu přidá k P negaci dotazu, tj. přidá $\neg G$ a snaží se z $P, \neg G$ (to je vlastně množina formulí) odvodit (tzv. rezoluční metodou) spor, tj. odvodit $\square$. Dá se dokázat, že G sémanticky vyplývá z P ($P \models G$), právě když je z $P, \neg G$ odvoditelná $\square$. Oznámí-li prologovský překladač po zadání dotazu G na program P odpověď „Yes“, znamená to právě, že překladač odvodil z $P, \neg G$ klauzuli $\square$.

Hornovská klauzule má jeden z následujících tvarů:

(a) (nenulový počet negativních literálů, právě jeden pozitivní literál) $A \vee \neg B_1 \vee \cdots \vee \neg B_n$ (A, B_i jsou atomické formule); tato klauzule je ekvivalentní formuli $B_1 \wedge \cdots \wedge B_n \Rightarrow A$ (Proč? Uvědomme si, že $B \Rightarrow A$ je ekvivalentní $A \vee \neg B$ a že $\neg(B_1 \wedge \cdots \wedge B_n)$ je ekvivalentní $\neg B_1 \vee \cdots \vee \neg B_n$.) Tyto klauzule odpovídají prologovským pravidlům.

(b) (nulový počet negativních literálů, právě jeden pozitivní literál) A. Tyto klauzule odpovídají prologovským faktům.

(c) (nenulový počet negativních literálů, žádný pozitivní literál) $\neg C_1 \vee \cdots \vee \neg C_n$ (C_i jsou atomické formule); tato klauzule, respektive její obecný uzávěr $\forall(\neg C_1 \vee \cdots \vee \neg C_n)$ je ekvivalentní formuli $\neg\exists(C_1 \wedge \cdots \wedge C_n)$ (Proč? Uvědomme si, že $\neg(\exists x)\varphi$ je ekvivalentní $(\forall x)\neg\varphi$.) Tyto klauzule odpovídají prologovským dotazům.

Definitní logický program; substituce

Definice

Logický program (někdy **definitní logický program**) je konečná množina hornovských klauzulí s jedním pozitivním literálem (tj. klauzulí odpovídajících pravidlům a faktům).

Pro naše potřeby nyní rozšíříme výklad o substitucích. Obecně budeme uvažovat substituce tvaru $(x_1/s_1, \ldots, x_n/s_n)$ (x_i jsou proměnné, s_i termy), kde proměnné x_i jsou navzájem různé (tj. pro $i \neq j$ je $x_i \neq x_j$) a dále $x_i \neq s_i$. Speciálně, () označuje tzv. prázdnou substituci; pro ni platí $\varphi() = \varphi$ a $t() = t$. Pro substituci $\theta = (x_1/s_1, \ldots, x_n/s_n)$ bude $\varphi\theta$ (popřípadě $t\theta$) označovat výsledek substituce použité na formuli φ (popřípadě term t); tento výsledek vznikne současným nahrazením proměnných $x_1, \ldots, x_n$ termy $s_1, \ldots, s_n$.

$\varphi(\sigma\theta)$ bude formule, která vznikne aplikací substituce θ na formuli vzniklou aplikací substituce σ na φ. Pro substituce $\sigma = (x_1/s_1, \ldots, x_n/s_n)$ a $\theta = (y_1/t_1, \ldots, y_m/t_m)$ definujeme substituci $\sigma\theta$ jako substituci, která vznikne z $(x_1/s_1\theta, \ldots, x_n/s_n\theta, y_1/t_1, \ldots, y_m/t_m)$ vymazáním

(1) $x_i/s_i\theta$, pro která $x_i = s_i\theta$

(2) y_i/t_i, pro která $y_i \in \{x_1, \ldots, x_n\}$.

Lehce se ověří následující tvrzení.

Lemma

Pro substituce σ, θ, τ platí:

(1) $\sigma() = ()\sigma = \sigma$

(2) $E(\sigma\theta) = (E\sigma)\theta$ pro libovolnou formuli (popř. term) E

(3) $(\sigma\theta)\tau = \sigma(\theta\tau)$.

Definice

Substituce θ se nazývá **obecnější** než substituce σ, jestliže existuje substituce τ, pro kterou $\sigma = \theta\tau$.

Tedy: θ je obecnější, protože z ní můžeme dostat σ dodatečným „upřesněním“ τ.

Definice

Substituce θ se nazývá **unifikační substituce** (také **unifikace**) množiny $\{\varphi_1, \ldots, \varphi_n\}$ formulí (popř. termů), pokud $\varphi_1\theta = \varphi_2\theta = \cdots = \varphi_n\theta$.

Tedy: unifikace je substituce, po jejíž aplikaci přejdou všechny formule φ_i ve stejnou formuli.

Definice

Unifikace θ množiny $\{\varphi_1, \ldots, \varphi_n\}$ se nazývá **nejobecnější unifikací** (zkratka **mgu** z anglického „most general unifier“) této množiny, jestliže je obecnější než každá jiná unifikace množiny $\{\varphi_1, \ldots, \varphi_n\}$.

Poznamenejme, že nejobecnější unifikace (mgu) se dá najít algoritmicky, přičemž není určena jednoznačně.

Obecná rezoluční metoda byla navržena v 60. letech 20. století Robinsonem. Základem je tzv. rezoluční odvozovací pravidlo, které ze dvou klauzulí odvodí jinou klauzuli, jejich tzv. **rezolventu**. Obecný tvar rezolučního pravidla je:

z klauzule $\{L_1, \ldots, L_{i-1}, A, L_{i+1}, \ldots, L_n\}$
a klauzule $\{L'_1, \ldots, L'_{i-1}, \neg A', L'_{i+1}, \ldots, L'_m\}$
odvoď klauzuli (rezolventu výše uvedených klauzulí)
$\{L_1\theta, \ldots, L_{i-1}\theta, L_{i+1}\theta, \ldots, L_n\theta, L'_1\theta, \ldots, L'_{i-1}\theta, L'_{i+1}\theta, \ldots, L'_m\theta\}$,
je-li θ mgu množiny $\{A, A'\}$.

Přitom L_i, L'_j jsou literály a A, A' jsou atomické formule.

Rezoluční pravidlo je svou povahou odvozovací pravidlo – z nějakých formulí odvodí jinou formuli. Pravidlo MP lze chápat jako speciální případ rezolučního pravidla: MP říká „z A a $A \Rightarrow B$ odvoď B“. Formule $A \Rightarrow B$ je ekvivalentní klauzuli $\neg A \vee B$. Tedy odvození pomocí MP odpovídá odvození $\{B\}$ z $\{\neg A, B\}$ a $\{A\}$. To lze pomocí rezolučního pravidla s unifikací, za kterou vezmeme identickou (prázdnou) substituci.

Pro danou množinu F klauzulí označme
$R(F) = \{C | C$ je rezolventou nějakých klauzulí z $F\}$.
Dále definujme $R^0(F) = F$ a $R^{n+1}(F) = R^n(F) \cup R(R^n(F))$.
Tedy $R^n(F)$ je množina klauzulí odvoditelných z F nejvýše n-násobným použitím kroku „utvoř všechny možné rezolventy“.

Nyní (bez důkazu) uvedeme základní výsledek popisující rezoluční metodu:

Věta

Množina F klauzulí je sporná, právě když existuje n tak, že $\square \in R^n(F)$.