Nerozhodnuteľnosť, neúplnosť a logika 2. rádu

12. prednáška
Logika pre informatikov a Úvod do matematickej logiky

Ján Kľuka, Ján Mazák, Jozef Šiška

Letný semester 2024/2025

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Obsah 12. prednášky

Logika 1. a 2. rádu

Nerozhodnuteľnosť platnosti formuly

Deskriptívna zložitosť

Neúplnosť

Dôkazy ako programy

Nadväzujúce predmety

Cieľom tejto prednášky je jemne doplniť filozofický, matematický, algoritmický a historický kontext týkajúci sa logiky.

Nie je nutné, aby ste uvedené veci podrobne ovládali — ide o veľké idey, nezaujímame sa o technické detaily.

Presná formulácia a dôkazy tvrdení z tejto prednášky:

- predmet Matematická logika (2-INF-114)

Logika 1. a 2. rádu

Zhrnutie: logika 1. rádu

Uvažujme jazyk logiky 1. rádu a v ňom teóriu T a formulu F.

Veta 17.1 (Korektnosť logiky 1. rádu)

Ak $T \vdash F$, tak $T \vDash F$.

Veta 17.2 (Úplnosť logiky 1. rádu)

Ak $T \vDash F$, tak $T \vdash F$.

Veta 17.3 (Kompaktnosť logiky 1. rádu)

Každá nesplniteľná teória má konečnú podmnožinu, ktorá je nesplniteľná.

Ako vyjadriť tranzitívny uzáver

Majme binárnu reláciu reprezentovanú predikátom R. Pre každé $n \geq 1$ označme $R_n(a, b)$ formulu

$$\exists x_1 \, \exists x_2 \cdots \exists x_n \big(R(a, x_1) \wedge R(x_1, x_2) \wedge \cdots \wedge R(x_n, b) \big)$$

(jednotlivé formuly R_n sú konečné, ale je ich nekonečne veľa — pre každé n by sme mohli náš jazyk rozšíriť o predikát, ktorý R_n definuje).

Ďalej nech $R_0 = R$ a nech

$$R^*(a, b) \quad \text{vtt} \quad \text{existuje } n \text{ také, že platí } R_n(a, b).$$

Naše R^* vyjadruje tranzitívny uzáver relácie R.

Dá sa však R^* vyjadriť prvorádovou formulou?

Ako vyjadriť tranzitívny uzáver

Predpokladajme, že R^* sa dá vyjadriť prvorádovou formulou.

Uvažujme nekonečnú prvorádovú teóriu

$$T = \{\neg R_0(a,b), \neg R_1(a,b), \ldots, \neg R_n(a,b), \ldots, \quad R^*(a,b)\}.$$

1. T nemá model: ak $R^*(a,b)$, tak pre nejaké n platí $R_n(a,b)$.
2. Každá konečná podmnožina T má model: ak n je najväčší z indexov R_n obsiahnutých v podmnožine, vyhovujúci model je napr. $D = \mathbb{N}$, $i(R) = \{(0,1),(1,2),\ldots,(n+1,n+2)\}$, $i(a) = 0$, $i(b) = n+2$.

Tieto dve pozorovania sú však v spore s kompaktnosťou logiky 1. rádu.

Formulou logiky 1. rádu tranzitívny uzáver nevieme vyjadriť.

Ako vyjadriť tranzitívny uzáver

V logike 2. rádu by to už šlo:

$$R^*(a,b) \leftrightarrow$$
$$\forall P \left(\Big(\forall x \left(R(a,x) \rightarrow P(x) \right) \wedge \forall x \, \forall y \left(P(x) \wedge R(x,y) \rightarrow P(y) \right) \Big) \rightarrow P(b) \right)$$

Tento popis je založený na tom, že vlastnosť „b *je* R*-potomkom* a“ interpretujeme ako „b dedí od a každú vlastnosť, ktorú majú všetci priami R-potomkovia a a zároveň sa zachováva cez R“.

Syntax a sémantiku logiky 1. rádu možno rozšíriť tak, aby umožňovala kvantifikovanie cez predikáty; dostaneme tak logiku 2. rádu.

Použitie formúl 2. rádu zvyšuje vyjadrovaciu silu jazyka, ale zároveň stratíme kompaktnosť a úplnosť.

Ako vyjadriť indukciu

Predstavme si, že chceme pridať axiómu umožňujúcu využívať v dôkazoch matematickú indukciu. Nech P^1 je predikát a S^1 funkčný symbol pre nasledovníka. Chceli by sme

$$\forall P \quad \Big(\big(P(0) \wedge \forall x\big(P(x) \to P(S(x))\big)\big) \to \forall x\, P(x)\Big).$$

Toto je však formula logiky 2. rádu — v logike 1. rádu nič ako $\forall P$ pre predikát P nemáme. Jediné, čo nám ostáva, je pridať axiómu

$$\Big(P(0) \wedge \forall x\big(P(x) \to P(S(x))\big)\Big) \to \forall x\, P(x),$$

kde P je predikát **vyjadriteľný formulou** v našom konkrétnom prvorádovom jazyku $\mathcal{L}$. To má nevýhody:

- dôkaz využívajúci túto axiómu bude platiť len pre jeden konkrétny predikát;
- dôkazy sa budú týkať len predikátov vyjadriteľných v $\mathcal{L}$.

Peanova aritmetika

Jednou z možností, ako formalizovať aritmetiku, je **Peanova aritmetika (PA)**. Jazyk vychádza z prvorádovej logiky s rovnosťou, indivíduová konštanta je 0, funkčné symboly sú S^1, $+^2$, $\cdot^2$ (nasledovník, sčítanie, násobenie). Prirodzené čísla reprezentujeme ako *termy* bez premenných, napr. 3 je $S(S(S(0)))$. Axiómy:

$$\begin{aligned}
&\forall x \;\; 0 \neq S(x) \\
\forall x\, \forall y \;\; &(S(x) \doteq S(y) \rightarrow x \doteq y) \\
&\forall x \;\; x + 0 \doteq x \\
\forall x\, \forall y \;\; &x + S(y) \doteq S(x + y) \\
&\forall x \;\; x \cdot 0 \doteq 0 \\
\forall x\, \forall y \;\; &x \cdot S(y) \doteq (x \cdot y) + x
\end{aligned}$$

Táto množina axióm je nezávislá (žiadna nevyplýva z iných).

Peanova aritmetika

K uvedeným axiómam treba pridať axiómu pre indukciu. Máme dve možnosti:

- Formuly 1. rádu.
  Nevýhoda: neštandardné modely (aj s nespočítateľnou doménou); ich existencia vyplýva o.i. z vety o kompaktnosti.
- Formula 2. rádu.
  Výhoda: všetky modely sú izomorfné s prirodzenými číslami.

Peanova aritmetika

Prečo práve tri funkčné symboly?

Ak vypustíme násobenie (hoci definovateľné pomocou sčítania), nevieme vyjadrovať vlastnosti sčítania (napr. vzťah pre násobok by zahŕňal tranzitívny uzáver).

Naopak, pridať funkčný symbol pre umocňovanie (či inú funkciu na prirodzených číslach) netreba, lebo prvorádová PA (dokonca aj bez indukcie) zachytáva **každú *vypočítateľnú* funkciu**.

Peanova aritmetika

Presnejšie, jazyk PA umožňuje formulou vyjadriť každú **primitívne rekurzívnu funkciu** (t. j. takú, čo sa dá počítať programom využívajúcim len cykly s vopred daným počtom opakovaní), a následne vie PA dokazovaním rozhodnúť o platnosti takýchto formúl.

Označme $\overline{x}$ reprezentáciu prirodzeného čísla x v PA (ako termu vznikajúceho opakovanou aplikáciou symbolu S, napr. $\overline{2} = S(S(0))$). Funkciu f^1 možno reprezentovať akousi formulou F takou, že $F(\overline{x}, \overline{y})$ je pravda vtt $f(x) = y$.

Ak $f(x) = y$, tak $PA \vdash F(\overline{x}, \overline{y})$, a ak $f(x) \neq y$, tak $PA \vdash \neg F(\overline{x}, \overline{y})$.

Nerozhodnuteľnosť platnosti formuly

Nerozhodnuteľnosť platnosti formuly vo FOL

Veta 18.1

Platnosť formuly v prvorádovej logike je nerozhodnuteľná.

Dôkaz: redukciou na problém zastavenia.

Ukážeme, ako by sme vedeli rozhodnúť, či daný Turingov stroj M zastaví, ak by sme vedeli rozhodovať platnosť prvorádových formúl.

Nerozhodnuteľnosť platnosti formuly vo FOL

Jazyk našej logiky bude obsahovať

- indivíduová konštanta ε pre prázdny reťazec;
- unárny funkčný symbol a^1 pre každé písmeno a v abecede;
- binárny predikát f_q pre každý stav q TS M.

Naša interpretácia tohto jazyka:

- $a(w)$ označuje reťazec aw;
- $f_q(x, y)$ indikuje, že M dosiahne na danom vstupe stav q, pričom na páske je reťazec $\overline{x}y$ (x v opačnom poradí) a hlava M je na prvom znaku y.

Nerozhodnuteľnosť platnosti formuly vo FOL

Krok výpočtu zachytáva formula

$$\forall x\, \forall y \quad f_q(x, a(y)) \to f_{q'}(b(x), y)$$

M prečíta z pásky a, zapíše b, prejde zo stavu q do stavu q' a posunie hlavu doprava. (Takúto formulu pridáme do teórie popisujúcej činnosť M pre každú dvojicu a, b.)

Pre pohyb hlavy doľava máme formulu

$$\forall x\, \forall y \quad f_q(c(x), a(y)) \to f_{q'}(x, c(b(y))).$$

Nerozhodnuteľnosť platnosti formuly vo FOL

Pre polohu hlavy na ľavom okraji pásky pridáme

$$\forall y \quad f_q(\varepsilon, a(y)) \to f_{q'}(\varepsilon, b(y))$$

(hlava sa nehýbe, len prepisuje znak na páske).

Podobne doriešime aj polohu na pravom okraji, keď sa hlava posunie na časť pásky, kam sa ešte nezapisovalo.

Konkrétne detaily závisia od uvažovaného variantu Turingovho stroja (páska môže byť obojstranne nekonečná apod.).

Nerozhodnuteľnosť platnosti formuly vo FOL

Začiatok výpočtu z počiatočného stavu q_0 na slove w popisuje formula $f_{q_0}(\varepsilon, w)$ a zastavenie stroja v (jedinom) akceptačnom stave q_{acc} vyjadruje formula

$$F_M = f_{q_0}(\varepsilon, w) \wedge T \rightarrow \exists x\, \exists y\ f_{q_{\text{acc}}}(x, y),$$

kde T je konjunkcia implikácií popisujúcich povolené prechody TS M.

Každý akceptačný výpočet M vieme prerobiť na dôkaz F_M (konečné slovo na páske a poloha hlavy popisujú hodnoty premenných x a y v konzekvente implikácie F_M; jednotlivé kroky výpočtu naznačujú, ako inštancovať všeob. kvantifikátory formúl z T). Keďže prvorádová logika je korektná, tak ak M zastaví, F_M je platná formula.

Naopak, ak F_M je platná, tak je pravdivá v každej interpretácii (štruktúre), čiže aj v tej našej týkajúcej sa TS M. Keďže premisy F_M sú splnené, musí byť splnený aj záver $\exists x\, \exists y\, f_{q_{\text{acc}}}(x, y)$, takže M zastaví.

Čiastočná rozhodnuteľnosť platnosti formuly

Veta 18.2

Platnosť formuly v prvorádovej logike (so spočítateľným jazykom) je čiastočne rozhodnuteľná.

Dôkaz: stačí enumerovať všetky dôkazy v danom jazyku.

Formula je platná vtt jej negácia je nesplniteľná. Dôsledky:

Veta 18.3

(1) Nesplniteľnosť formuly v prvorádovej logike je nerozhodnuteľná a čiastočne rozhodnuteľná.
(2) Splniteľnosť v prvorádovej logike nie je ani čiastočne rozhodnuteľná.

Dôkaz (2): ak by sme splniteľnosť vedeli rozhodovať čiastočne, môžeme paralelne spustiť testovanie splniteľnosti aj nesplniteľnosti, a jeden z týchto výpočtov by musel skôr či neskôr skončiť, čím by sme vedeli rozhodovať nesplniteľnosť formuly, a to je spor.

Monadic FOL

Monadická prvorádová logika (MFOL) je prvorádová logika, v ktorej nemáme funkčné symboly a predikáty majú len jeden argument. (V takomto jazyku možno vyjadriť bežné sylogizmy a počas väčšiny 19. storočia sa verilo, že postačuje na formalizáciu uvažovania.)

Veta 18.4 (1915)

Pravdivosť formuly v MFOL je rozhodnuteľná.

Ak pridáme čo len jeden predikát arity 2, stratíme rozhodnuteľnosť.

Tento príklad ilustruje, že zvoliť jazyk s veľkou vyjadrovacou silou (napr. pre databázový systém) neraz prináša algoritmické problémy, preto to nie je samozrejmá voľba.

Deskriptívna zložitosť

Deskriptívna zložitosť

Pri klasickom pohľade na zložitosť algoritmov skúmame, koľko krokov spraví Turingov stroj v závislosti od veľkosti vstupu, resp. či Turingov stroj s dodatočnými obmedzeniami vôbec vie problém riešiť.

Alternatívny pohľad: akú zložitú logickú formulu potrebujeme na popis daného problému (jazyka akceptovaného Turingovým strojom)?

Príklady rôzne zložitých formúl:

- prvorádová logika (FOL)
- druhorádová logika (SOL) — kvantifikujeme aj predikáty/množiny objektov
- existential SOL: $\exists X_1\, \exists X_2 \cdots \exists X_n\ F$, F je formula FOL
- universal SOL: $\forall X_1\, \forall X_2 \cdots \forall X_n\ F$, F je formula FOL

Deskriptívna zložitosť

Existencia trojuholníka v grafe sa dá vyjadriť vo FOL:

$$\exists x\, \exists y\, \exists z\ \big(V(x) \wedge V(y) \wedge V(z) \wedge \\ \wedge\, x \neq y \wedge y \neq z \wedge z \neq x \wedge E(x,y) \wedge E(y,z) \wedge E(z,x)\big)$$

Pri testovaní, či konkrétny graf spĺňa túto formulu, nerozhodujeme o platnosti formuly vo všeobecnosti (to je nerozhodnuteľný problém), ale len vyhodnocujeme jej splnenie v konkrétnej interpretácii: predikát V popisuje vrcholy, E hrany.

Pri vyhodnotení kvantifikátora stačí preskúmať všetky vrcholy, čiže pre k kvantifikátorov vo formule preveríme $O(|V|^k)$ možností. Preto tento problém patrí do triedy P (zjemnenie úvah umožňuje dokázať príslušnosť do LOGSPACE).

Deskriptívna zložitosť

Existencia 3-farbenia grafu: NP-complete;
nevyjadriteľné vo FOL, vyjadriteľné v existential SOL:

$$\exists R\, \exists G\, \exists B \big(\forall x\, \forall y\, (E(x,y) \to \neg R(x) \vee \neg R(y)) \wedge \cdots \big)$$

(R, G, B sú predikáty vyjadrujúce jednotlivé farby).
Pri vyhodnocovaní formuly pre konkrétny graf za doménu zoberieme vrcholy grafu. Existencia predikátu zodpovedá existencii podmnožiny všetkých vrcholov; túto podmnožinu vieme nedeterministicky uhádnuť a vyhodnotenie prvorádovej časti formuly už pridá len polynomiálny faktor (exponent závisí od počtu vnorených kvantifikátorov), preto tento problém patrí do triedy NP.

Deskriptívna zložitosť

- FOL: trieda zložitosti AC^0 (vlastná podmnožina LOGSPACE)
- FOL + tranzitívny uzáver: non-deterministic LOGSPACE
- FOL + least fixed point operator: P
  (súvisí s databázami: dotazy sú podmnožinou prvorádových formúl a navyše je povolená rekurzia počítaná seminaivnou evaluáciou, čiže ako least fixed point)

- existential SOL: NP
- universal SOL: co-NP
- SOL: PH (obsahuje NP aj coNP, ale vlastná podmnožina PSPACE)
- SOL + least fixed point operator: EXPTIME

Neúplnosť

Efektívna axiomatizovateľnosť

Ako formalizovať, že niečo „možno vypočítať“?

Vieme vymýšľať rôzne konkrétne modely na počítanie a porovnávať, čo dokážu. Ukazuje sa, že všeobecné modely počítania s prirodzenými číslami (nie naschvál oslabené dodatočnými reštrikciami) vedú k tomu istému pojmu vypočítateľnosti.

- Turingove stroje
- frázové gramatiky
- Minského registrové automaty
- lambda kalkul
- čiastočné rekurzívne funkcie

Church-Turing Thesis

Ak sa funkcia z $\mathbb{N}$ do $\mathbb{N}$ dá vypočítať,
dá sa vypočítať na Turingovom stroji.

Efektívna axiomatizovateľnosť

Z hľadiska logiky nás zaujímajú formálne systémy, s ktorými možno algoritmicky pracovať:

- konečná množina symbolov
- algoritmy na prácu s termami
- algoritmy na prácu s formulami
- jednoznačné interpretovanie formuly v danej štruktúre
- rozhodnuteľnosť, či niečo je axióma
  (resp. enumerácia axióm/formúl v teórii)
- rozhodnuteľnosť, či niečo je dôkaz
  (resp. enumerácia dôkazov)

Teórii s týmito vlastnosťami (resp. formálnemu systému, ktorého je súčasťou) hovoríme **efektívne axiomatizovateľná**
(v angličtine *effectively*, nie efficiently).

Sémantická a negačná úplnosť

Doterajší pojem úplnosti, tzv. **sémantická úplnosť** formálneho systému, hovoril o tom, že pravdivé veci možno dokázať:
Pre každú teóriu T a uzavretú formulu F, ak $T \vDash F$, tak $T \vdash F$.

Zaujímavá je však aj **negačná úplnosť** teórie T: pre každú uzavretú formulu F (v jazyku tejto teórie) platí $T \vdash F$ alebo $T \vdash \neg F$.

Sémantická a negačná úplnosť

Uvažujme výrokovú logiku s atómami A, B, C a teóriu $T = \{A \wedge B\}$.
Sémantická úplnosť vyplýva z vlastností výrokovej logiky.

Avšak napr. formula C je nezávislá od teórie T, neplatí $T \vDash C$ ani $T \vDash \neg C$. Preto $T \nvdash C$ a $T \nvdash \neg C$, čiže T nie je negačne úplná.

Sémantická a negačná úplnosť

Je ľahké vytvárať teórie, ktoré sú negačne neúplné:
stačí vynechať zopár podstatných axióm/formúl.

Ak sa však snažíme nájsť teóriu T, ktorá čo najúplnejšie popisuje nejakú časť sveta vyjadriteľnú v jazyku $\mathcal{L}$, chceli by sme, aby pre každú uzavretú formulu jazyka $\mathcal{L}$ platilo buď $T \vDash F$ alebo $T \vDash \neg F$. Preto by sme chceli vedieť efektívne rozhodnúť, či $T \vdash F$ alebo $T \vdash \neg F$. Hľadáme teda negačne úplné teórie.

Prvá Gödelova veta o neúplnosti

Veta 20.1

Nech T je prvorádová teória v jazyku $\mathcal{L}$ s týmito vlastnosťami:

- *je konzistentná (nevyplýva z nej nepravda);*
- *je efektívne axiomatizovateľná;*
- *obsahuje aritmetiku.*

Potom T nie je negačne úplná
(t. j. existuje tvrdenie v $\mathcal{L}$, ktoré nemožno z T dokázať ani vyvrátiť).

Prvá Gödelova veta o neúplnosti

Idea dôkazu — Gödelova veta G: „toto je veta, ktorá sa nedá dokázať“.

Aritmetiku potrebujeme na to, aby sme vedeli jednoznačne očíslovať všetky možné formuly a tiež všetky možné dôkazy (efektívna axiomatizovateľnosť zaručuje, že formuly aj dôkazy sa dajú enumerovať, je ich teda spočítateľne veľa). Potom vieme kľúčovú vetu G zapísať ako prvorádovú formulu.

Ak by sa G dala dokázať, je nepravdivá, čo je spor s konzistentnosťou. Ak sa G dokázať nedá, je pravdivá. Ale potom $\neg G$ je nepravdivá, a teda sa nemôže dať v konzistentnom systéme dokázať. Teória T teda nie je negačne úplná.

Prvá Gödelova veta o neúplnosti

Skonštruovaná Gödelova veta G_T je síce dlhá, ale má jednoduchú štruktúru, nejde o nejakú absurdne zložitú formulu.

A nielen to. Ak do T doplníme ako axiómu jej G_T alebo $\neg G_T$, naďalej budú splnené predpoklady prvej Gödelovej vety, a výsledná teória bude stále negačne neúplná.

Toto prekvapivé tvrdenie z r. 1931 znamenalo krach Hilbertovho programu z r. 1900 (snahy o kompletnú axiomatizáciu matematiky v štýle Euklidových základov).

Tarského veta

Kľúčovým krokom v dôkaze prvej Gödelovej vety je konštrukcia formuly $F_1(k)$ vyjadrujúcej, že formula s kódom k je dokázateľná v rámci daného logického systému.

Tarského veta: v rámci daného logického systému nie je možné zostrojiť formulu $F_2(k)$, ktorá by vyjadrovala, že formula s kódom k je platná.

Pozorovanie:

Veľká časť logiky ako vedného odboru je zameraná na nahradenie práce s *pravdivosťou* prácou s *dokázateľnosťou*, ktorá je na rozdiel od pravdivosti algoritmicky uchopiteľná.

Pojem výpočtu je s logikou veľmi úzko spätý: kvôli aplikáciám v logike vlastne vznikli prvé formalizácie pojmu výpočet či algoritmus (Gödel cca 1930 — primitívne rekurzívne funkcie).

Druhá Gödelova veta o neúplnosti

Veta 20.2

Ak S je bezosporný efektívne axiomatizovateľný formálny systém obsahujúci aritmetiku (napr. PA), jeho bezospornosť nemožno dokázať v rámci S.

Čiže dôkaz treba spraviť neformálnou metaúvahou alebo v rámci iného „silnejšieho“ formálneho systému. Ako však dokážeme bezospornosť tohto druhého systému? …

V tomto zmysle nevieme dokázať ani len bezospornosť PA, nieto zložitejších formalizácií matematiky.

Dôkazy ako programy

Prirodzená dedukcia / natural deduction

Prirodzená dedukcia: gentzenovský systém, konštruktívne dôkazy (nie ako tablá, kde odvodzujeme spor).

Dôkaz pre $(A \wedge C) \to (B \to A)$:

1. Predpokladajme $A \wedge C$ (hypotéza H_1).
2. Predpokladajme B (hypotéza H_2).
3. Z $A \wedge C$ odvodíme A — t.j. $\{H_1, H_2\} \vdash A$
   (pravidlo $\wedge E_L$ použité na 1).
4. Z predpokladu B sme odvodili A, preto $\{H_1\} \vdash B \to A$
   (pravidlo $\to\ I$ použité na 3, discharging H_2).
5. Z predpokladu $A \wedge C$ sme odvodili $B \to A$, preto
   $\vdash (A \wedge C) \to (B \to A)$
   (pravidlo $\to I$ použité na 4, discharging H_1).

Lambda kalkul

Lambda kalkul:

- Formálny systém na zapisovanie výpočtov.
- **Výpočty sú *termy*** vznikajúce kombinovaním dvoch jednoduchých operácií:
  - function abstraction (viazanie voľných premenných),
  - function application (substitúcia / skladanie funkcií).
- Termy pozostávajú z funkcií
  (aj konštanty vnímame ako konštantné funkcie).

Lambda kalkul: viazanie premenných

Viazanie premennej:

- K danému termu M možno vytvoriť nový term $\lambda x.M$.
- Premenná x, ktorá reprezentuje vstup do funkcie, je tým viazaná v rámci termu M (všetky jej výskyty).
- Term M môže obsahovať voľné premenné.
- Príklad: v $\lambda x.(x + y)$ je x je viazaná, ale y voľná premenná (jej hodnota prichádza kdesi zvonku).

Lambda kalkul: skladanie funkcií

Aplikovanie funkcie / skladanie funkcií / substitúcia / β-redukcia:

- Z termov M, N môžeme vytvoriť term (MN).
- Príklad: $((\lambda x.M)N)$ — funkciu $\lambda x.M$ aplikujeme na argument N, čiže všetky viazané výskyty x v rámci M sú nahradené N.
- Príklad: $((\lambda x.(x + 2))\ 3)$.
  - Tu je M výraz $(x + 2)$, x je viazaná premenná a N je 3.
  - Substitúcia: nahraď x vo výraze $(x + 2)$ číslom 3.
  - Výsledok: $(3 + 2)$, čo sa vyhodnotí na 5.

Lambda kalkul: typy

Lambda kalkul je turingovsky úplný, čiže pripúšťa aj nekonečné výpočty (napr. je možné funkciu aplikovať na seba). Ak sa nám to nepáči, môžeme pridať mechanizmus na prácu s typmi, ktorý zabezpečí konečnosť (napr. Simply Typed Lambda Calculus). Ukážka:

- atomické typy: void (príp. bool, int…)
- $T_1 \to T_2$: funkcia s argumentom typu T_1 vráti hodnotu typu T_2, napr. $\lambda x : T_1.e$, kde e je telo funkcie vracajúce T_2
- product type $T_1 \times T_2$, usporiadaná dvojica objektov daných typov

Typy pridávajú obmedzenia pre skladanie funkcií, vďaka čomu máme kontrolu aj nad algoritmickými aspektmi (termy v lambda kalkule popisujú výpočty).

Lambda kalkul: typy a logické spojky

K danej výrokovologickej formule vieme zostaviť term v lambda kalkule.

- Formule $A \wedge C$ zodpovedá typ $A \times C$. Na prácu s objektmi tohto typu sa hodí funkcia proj_i, ktorá vráti i-tú zložku danej dvojice.
- Formule

  $$(A \wedge C) \to (B \to A)$$

  zodpovedá term

  $$\lambda p : (A \times C).(\lambda q : B.\text{proj}_1(p)).$$

  Táto funkcia prijíma dôkaz p výroku $A \wedge C$ a vracia „vnútornú“ funkciu. Táto vnútorná funkcia prijíma dôkaz q výroku B a vracia dôkaz A (získaný z p).

Curryho-Howardova korešpondencia

Medzi dôkazmi a výpočtami existuje „izomorfizmus“.

Logika (Prirodzená dedukcia)	**Výpočty (Lambda kalkul)**
Výrok (A, B, C) $P \wedge Q$ $P \to Q$	Typ (A, B, C) product type $P \times Q$ functional type $P \to Q$
Dôkaz / pravidlá $\wedge E_L$ (napr. z $p : A \wedge C$ dostaň A) $\to I$ (predpokladaj X, odvoď Y)	Term (program) / operácie Projekcia ($\text{proj}_1(p)$) Lambda abstrakcia ($\lambda x : X.y$)
Formula: $(A \wedge C) \to (B \to A)$	Typ: $(A \times C) \to (B \to A)$
Náš dôkaz (kroky dedukcie)	Náš program (lambda term): $\lambda p.(\lambda q.\text{proj}_1(p))$

Formula je dokázateľná vtt existuje term zodpovedajúceho typu.

Lean: jazyk a interaktívny dokazovací systém

Lean je dôkazový asistent založený na teórii typov (Calculus of Constructions).

- Implementuje Curryho-Howardovu korešpondenciu: výroky sú typy, dôkazy sú termy.
- Na dokázanie formuly treba skonštruovať term zodpovedajúceho typu.
- Lean vie verifikovať existujúce dôkazy (termy).
- Lean tiež ponúka tzv. „taktiky“, ktoré pomáhajú automatizovať konštrukciu dôkazu (napr. mu možno dať pokyn „sprav rozbor prípadov“, „zovšeobecni“, „použi lemu“).
- `mathlib` je rastúca knižnica formálne verifikovaných matematických tvrdení
- ľudia s Fieldsovou medailou v tom robia formalizované dôkazy

Lean: dôkaz pomocou taktík

```
-- Declare A, B, C as propositional atoms
variable (A B C : Prop)

theorem my_tactic : (A ∧ C) → (B → A) := by
  -- Assume the hypothesis (A ∧ C) and call it h_ac
  -- The goal becomes: B → A
  intro h_ac
  -- Deconstruct h_ac : A ∧ C into its components.
  -- This introduces h_a : A and h_c : C into the context.
  -- The goal remains: B → A
  obtain ⟨h_a, h_c⟩ := h_ac
  -- Assume the hypothesis B (the antecedent of the implication)
     and call it h_b
  -- The goal becomes: A
  intro h_b
  -- The goal is now A, and we have h_a : A in our context.
  -- Use h_a to exactly prove the goal.
  exact h_a
```

Lean: dôkaz priamou konštrukciou termu

```
-- Declare A, B, C as propositions
variable (A B C : Prop)

-- Direct proof term
theorem my_term : (A ∧ C) → (B → A) :=
  fun h_ac : A ∧ C =>   -- Assume h_ac is a proof of A ∧ C
    fun h_b : B =>      -- Assume h_b is a proof of B
      -- Return the left component of h_ac, which is a proof of A
      (like proj_1(p))
      h_ac.left
```

Lean: kvantifikátory a použitie vety

```
-- Import necessary definitions (Nat, <, succ) and theorems
import Mathlib.Data.Nat.Order.Basic

-- Theorem from Mathlib: 0 is less than the successor of any n.
theorem zero_lt_succ (n : Nat) : 0 < Nat.succ n

-- Prove that there exists a natural number greater than 0.
example : ∃ x : Nat, x > 0 :=
  -- The type corresponding to an existential quantifier
  -- is a pair (witness, proof for the witness).
  -- We provide the witness `47` (which is `Nat.succ 46`)
  -- and use `zero_lt_succ` to prove `47 > 0`.
  ⟨47, zero_lt_succ 46⟩
```

Lean: AlphaProof

AlphaProof (od Google DeepMind) má za cieľ objavovať a overovať matematické dôkazy pomocou umelej inteligencie. Za základ si vybrali Lean:

- Lean ponúka spoľahlivú formálnu verifikáciu.
- Táto verifikácia je veľmi rýchla, používa ju napr. Amazon ako základnú metódu na verifikáciu softvéru.
- Hoci problém splniteľnosti formuly je nerozhodnuteľný, Lean má dobre navrhnutý systém obmedzení, ktorý garantuje konečnosť verifikácie (napr. umožňuje použiť len rekurziu, ktorá dokázateľne skončí; ak dôkaz nevie nájsť Lean, žiada ho od užívateľa).

Lean: AlphaProof

- Jazyk, ktorý používa Lean, zahŕňa nielen FOL, ale aj logiku vyšších rádov (napr. je možné kvantifikovať cez funkcie, typy a predikáty). Vyjadrovacia sila formalizmu, na ktorom je Lean postavený (Calculus of Inductive Constructions), stačí viac-menej na všetko (okrem „naschvál skonštruovaných extrémov“).
- Systém taktík umožňuje automatizáciu dôkazov. AI môže generovať stratégie dôkazu na vysokej úrovni alebo konkrétne volania taktík, ktoré Lean následne vykoná a overí.
- `mathlib` — obrovská aktívne vyvíjaná knižnica formalizovanej matematiky; možno na nej trénovať AI.

Nadväzujúce predmety

Nadväzujúce predmety

Presná formulácia a dôkazy tvrdení z tejto prednášky:

- Matematická logika (2-INF-114)

Viac o vypočítateľnosti:

- Teória vypočítateľnosti (2-INF-121)

Aplikácie logiky:

- Špecifikácia a verifikácia programov (1-AIN-470)
- Výpočtová logika (2-AIN-108)
- Reprezentácia znalostí a inferencia (2-AIN-144)
- Výpočtová fuzzy logika, modelovanie a systémy (2-AIN-113)
- Ontológie a znalostné inžinierstvo (2-AIN-286)