AD4M33AU
Automatické uvažování

Limity formálních metod

Petr Pudlák

Sebe-referenční programy a tvrzení

- Společný princip dokazování mnoha těchto tvrzení, že něco nejde, je to, že exaktně zformalizujeme nějaký paradox.
- Použitelné paradoxy:
  - Paradox lháře: *„Toto tvrzení je nepravdivé.“*
  - Berryho paradox: *„Číslo* n *je nejmenší číslo, které nelze definovat 11 slovy.“*

Halting problém

- Nelze sestrojit algoritmus *H*, který vždy dovede rozhodnout, jestli nějaký program skončí nebo ne.
- Jiná formulace: Mějme program *H* takový, který správně rozhoduje, jestli nějaký program skončí nebo ne. Pak lze sestrojit program *P*, na kterém se *H* zacyklí.
- Princip důkazu je ten, že k programu *H* sestrojíme program, který *„skončí právě když neskončí“* (paradox lháře). *H* pak nemůže správně odpovědět, tedy nikdy neskončí.

Sebe-referenční program

- Sestrojme program, který vypisuje svůj vlastní zdrojový kód:
  - Značme *[P]* zdrojový kód programu *P*.
  - Sestrojíme pro *Q*, který dostane na vstupu zdrojový kód *[P]* programu *P* a zkonstruuje zdrojový kód nového programu, který spustí *P* na zdrojový kód *P*:
    Q([P]) := [P([P])]
  - Pak *Q([Q]) = [Q([Q])]*, tedy *Q* vypisuje svůj zdrojový kód.
- (Příklad.)

Sebe-referenční program

- Cvičení: Napište ve zvoleném jazyce program, který vypíše svůj vlastní zdrojový kód.

Halting problém

- Předpokládejme, že existuje algoritmus *H* takový, že:
  - *H([P]) := True* pokud *P* skončí
  - *H([P]) := False* pokud *P* neskončí
- Sestrojme program *R*:
  - *R([P]) := [* **if** *H([P([P])])* **then** *{* **while** *True* **do** *{ } }]*
- Zkoumejme, zda *R([R])* skončí nebo ne:
  - Pokud skončí, tak to znamená, že *H([R([R])] = False*, tedy že *R([R])* neskončí.
  - Pokud neskončí, tak to znamená, že *H([R([R])] = True*, tedy že *R([R])* skončí.
- Tedy nemůže existovat takový algoritmus *H*.

Odbočka: Funkc. programování a lambda systém F

- Turingův stroj umožňuje sestrojovat jak končící i nekončící programy, a nelze je rozlišit.
- Existují formalismy, které umožňují popisovat právě všechny „rozumné“ končící programy.
- Lambda systém F (také [Girard–Reynoldsův] polymorfní lambda kalkulus nebo lambda kalkulus 2. řádu) umožňuje zapsat právě ty funkce, o kterých lze v Peanově aritmetice 2. řádu dokázat, že jsou totální.
- Tedy v podstatě všechny v praxi myslitelné končící programy.

Teorie

- **Teorie** znamená množinu formulí.
- Obvykle máme nějakou množinu **A** axiomů a uvažujeme teorie typu:
  „**T** je nejmenší množina taková, že **A**⊆**T** a která je uzavřená na aplikaci daných odvozovacích pravidel.“
- Tedy T je množina všech tvrzení, které lze odvodit z **A** pomocí odvozovacích pravidel.

Gödelova první věta o neúplnosti logiky 1. řádu

- **„Každá efektivně generovaná teorie, ve které lze popsat elementární aritmetiku, nemůže být zároveň konzistentní a úplná.“**
- Konzistentní = bezesporná (sporná = všechna tvrzení jsou dokazatelná.)
- Efektivně generovaná = lze napsat program, který postupně vypisuje všechny formule teorie.
- Úplná = lze v ní dokázat všechna pravdivá tvrzení aritmetiky (o přirozených číslech).

Myšlenka důkazu – aritmetizace

- Jazyk teorie **T** zakódujeme čísly (tzv. Gödelovo číslování):
  - Každý symbol v jazyce reprezentujeme nějakým číslem.
  - Konečné posloupnosti symbolů zakódujeme pomocí jednoho čísla. Jsou-li $x_1,\ldots,x_n$ kódy pro symboly $S_1,\ldots,S_n$, zakódujeme sekvenci těchto symbolů číslem

    $$p_1^{x1} \cdot p_2^{x2} \ldots p_n^{xn}$$

    kde $p_1,\ldots,p_n$ je prvních n prvočísel.

Myšlenka důkazu – aritmetizace

- Pomocí vhodného aritmetického predikátu vyjádříme skutečnost, že daná posloupnost symbolů je formulí v jazyce teorie **T**.
- Pro danou formuli φ pak označme $[\varphi]$ číslo, které je kódem φ.

Myšlenka důkazu – aritmetizace

- Jelikož všechna dokazatelná tvrzení teorie **T** lze efektivně generovat algoritmem, můžeme tento algoritmus popsat aritmetickou formulí.
- Sestrojíme tedy formuli *bew(X)*, která říká, že „*X* je kód formule dokazatelné v teorii **T**“.
- Pokud je *φ* dokazatelná v **T** tak v **T** je dokazatelné také *bew([φ])*.

Myšlenka důkazu – diagonalizace

- Podobně jako u halting problému sestrojíme sebe-referenční formuli formuli ψ. Ta bude říkat „jsem nedokazatelná“, ψ bude tvrdit $\neg bew([\psi])$.
- Předpokládejme, že ψ je dokazatelná v **T**:
  - Jinými slovy, $\neg bew([\psi])$ je dokazatelné v **T**.
  - Zároveň víme: pokud je nějaká φ dokazatelná v **T** tak v **T** je dokazatelné také $bew([\varphi])$. V **T** je tedy dokazatelné $bew([\psi])$, tedy **T** by byla sporná teorie.
- Tedy ψ je nedokazatelná v **T**.

Diagonalizace

- Uvažujme nyní formule, které mají právě jednu volnou proměnnou *x*. Značme *φ(m)* formuli vzniklou z *φ* substitucí čísla *m* za proměnnou *x.*
- Podobně jako u halting problému lze z čísla *[φ]* sestrojit číslo *[φ([φ])].*
- Sestrojme formuli *χ(m)* říkající „¬*bew([φ([φ])])*“, je-li *m* kódem formule *φ.*
- Pak *χ([χ])* říká „¬*bew([χ([χ])])*“, neboli *χ([χ])* říká „*χ([χ])* je nedokazatelná.“

Důsledky:

- Pomocí formálních metod nelze nikdy dokázat všechna fakta o přirozených číslech. Pro jakýkoliv systém (program), který sestrojíme, bude existovat nedokazatelné tvrzení.

Důsledky:

- Gödelova věta o úplnosti logiky 1. řádu:
  - Pro libovolnou teorii **T** a domněnku *C* platí:
    T $\vdash$ *C* právě když neexistuje žádný model *M* takový, že *M* $\models$ **T** a *M* $\not\models$ *C*
- Dohromady vět o úplnosti a neúplnosti plyne:
  - Existuje nějaké pravdivé tvrzení *D* o přirozených číslech, které není dokazatelné v dané teorii **T**.
  - Existuje tedy model *M* t.ž. *M* $\models$ **T** a *M* $\not\models$ *D.*
  - Model *M* tedy nemůže být $\mathbb{N}$. Takový model se nazývá **nestandardní** a obsahuje kromě všech přirozených čísel ještě další prvky, tzv. **nestandardní čísla**.

Důsledky:

- Víme, že máme algoritmus, který vypisuje všechna tvrzení dané teorie **T**.
- **Neexistuje algoritmus, který vypíše právě ta tvrzení, která nejsou v teorii T!**
- Kdyby existoval, dostali bychom kombinací těchto 2 algoritmů rozhodovací proceduru pro teorii **T**, tedy konzistentní a úplný systém.
- Gödelova věta říká, že takový nemůže existovat.

Důsledky:

- **Neexistuje algoritmus, který by pro každou splnitelnou množinu axiomů A našel model.**
- Nemůžeme tedy sestrojit úplný hledač modelů.
- Kdyby existoval, mohli vyvrátit každé nepravdivé *C* tvrzení v **T** nalezením modelu
  $M \models \mathbf{T} \cup \{\neg C\}$
- (To ovšem neznamená, že nelze sestrojit program, který by nacházel modely v některých případech.)

Praktické limity automatického dokazování

- I v případě úplných metod (dokazování) jsme omezeni:
  - časem – výsledek má pro nás smysl jen pokud ho dostaneme v rozumném čase;
  - pamětí – fyzické omezení techniky, kterou používáme.
- Velmi závisí na použitém algoritmu, jak efektivně dovede řešit úlohu v daném čase a s daným množstvím paměti.

Praktické limity automatického dokazování

- Často bychom potřebovali používat teorii s nekonečným (rekurzivně spočetným) množstvím axiomů, například aritmetiku se schématem axiomů indukce:
- Pro každou formuli $\varphi(X,\overline{Y})$ přidáme axiom:
  $(\forall \overline{Y})\ (\ \varphi(0,\overline{Y})$
  $\quad\quad \& (\forall X)\ (\ \varphi(X,\overline{Y}) \rightarrow \varphi(s(X),\overline{Y})\ )$
  $\quad \rightarrow (\forall X)\ \varphi(X,\overline{Y})\ )$
  kde $\overline{Y}$ zkracuje $Y_1,\ldots,Y_k$.
- Současné dokazovače ale pracují pouze s konečným počtem axiomů.