Formální Metody a Specifikace (LS 2011)
Přednáška 2:
Specifikace programů a základy praktické logiky 1

Stefan Ratschan

Katedra číslicového návrhu
Fakulta informačních technologií
Ceské vysoké učení technické v Praze

25. únor 2011

Příklad

```
result:= "sorted"
for i:= 1 to 10 do
  if a[i]>a[i+1] then
    result:= "unsorted"
  endif
endfor
return result
```

Je tento algoritmus správný?

- Pokud chceme vědět jestli dnes máme vařit guláš nebo řízek?
- Pokud chceme spojit iPod se sítí?
- Pokud chceme vědět jestli je pole a seřazené?

Správnost algoritmů má smysl jen ve spojení s určitou specifikací!

Specifikace

```
result:= "sorted"
for i:= 1 to 10 do
  if a[i]>a[i+1] then
    result:= "unsorted"
  endif
endfor
return result
```

Zkusíme napsat specifikaci.

Pozor: Vedlejší účinky!

Zatím: Algoritmy bez vedlejších účinek,
specifikace vstupů/výstupů (I/O specification)

Specifikace

- může dovolovat víc správných výstupů pro jeden vstup, ale
- chceme aby pro každý vstup existoval aspoň jeden výstup.

Specifikační jazyky

Specifikace se obvykle píšou v přirozených jazycích (čeština, angličtina).

Problém: chybí

- preciznost, jednoznačnost
- srozumitelnost pro počítač (automatizace)

Řešení: umělé jazyky

Predikátová logika

Příklad specifikace

Přesněji: predikátová logika prvního řádu +
teorie množin, celých čísel, seznamů, polí atd.

Čistě teoreticky, teorie množin stačí a další teorie nepřídávají sílu.

Další plán

Zbytek dnešní a příští přednášky: Základy praktické logiky

Praktický: Přizpůsobeno každodenním použití v specifikaci atd.

Na univerzitách se logika většinou učí
jako předmět pro studium základy matematiky a informatiky
(Hilbertovský kalkulus, resoluční algoritmus atd.).

To je sice

- velmi elegantní, krásné, vhodné pro studium těch základ, ale bohužel
- v praktickém životě jen málo použitelné.

Logika v každodenním životě

Už jsme viděli že logika by mohla pomoct při specifikaci algoritmů.

Já osobně používám logiku i když vařím, uklízím, atd.

Ale přiznávám že jsem trochu podivná osoba

Pokaždé když čtu noviny bych rád poslal všechny novináře do kurzu logiky (tolik je tam logických chyb)

Už jsme konstatovali že lidský mozek,
vznikl před několika tisíci lety, a tentokrát měl jiný účel
To platí zejména pro logický modul v mozku (tzv. selská logika).

Tím že cvičíme formální logiku,
se vylepšíme i selskou logiku pro každodenní život.

Cíl příštích dvou přednášek

Příští dva přednášky: Pár logických pravidel které

budou i platit pokud přespáváte 1000 let, a
se probudíte v době kdy už nikdo neví co je Java, XML atd.

Doporučení, formální detaily :
J. Velebil: "Velmi jemný úvod do matematické logiky":
`ftp://math.feld.cvut.cz/pub/velebil/y01mlo/logika.pdf`
Kapitola "Predikátová logika"

Jak vytvořit logické výrazy (Syntax)

Term (např. $2x + 3$, $\texttt{first}(\texttt{rest}(l))$, $a(i)$):

- proměnná
- funkční symboly s určitou aritou

Funkční symboly arity 0 nazýváme *konstanty*.

V každodenním životě obvykle dáváme funkčním symbolům význam:

- $+$ obvykle znamená sčítání
- `rest` obvykle znamená zbytek seznamu atd.

Navíc dáváme funkčním symbolům často typ, např:

- $+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
- $\texttt{num\_elem} : \texttt{List} \to \mathbb{N}_0$.

Jak vytvořit logické výrazy (Syntax)

Formule: (např: $\exists x \,.\, 2x + 3 > 0 \wedge x^2 \leq 0$, $empty(a(i))$)

- kvantifikátory: $\forall$, $\exists$
- logické spojky: $\vee$, $\wedge$, $\neg$
- predikáty s termy jako argument, zejména predikát $=$
- **T**, **F**

Pozor:
$\exists x \,.\, P(x) \wedge Q(x)$ vs. $\exists x \,.\, [P(x) \wedge Q(x)]$ vs. $[\exists x \,.\, P(x)] \wedge Q(x)$

V každodenním životě obvykle dáváme predikátům význam,
např. $<$ obvykle znamená "méně než", atd.

Navíc dáváme i predikátům často typ, např:

- $<$ je predikát na $\mathbb{R} \times \mathbb{R}$,
- `is_empty_list` je predikát na seznamech atd.

Jak vytvořit logické výrazy (Syntax)

Když píšeme formule musíme dávat pozor aby

- všechny funkční symboly a predikáty vždy
  měly dovolený počet argumentů, a
- se odpovídaly typy těch argumentů.

Když píšeme programy, ve většině programových jazyků to dělá pro nás compiler (kromě JavaScript, LISP, Prolog, ...)

Jinak musíme dávat pozor (z novinářštiny "Klaus je lepši prezident")

Syntaktický strom

Další pojmy

Volná/vázaná proměnná.

J. Velebil:

> *Výskyt standardní proměnné x ve formuli nazýváme vázaným, pokud při cestě od tohoto listu ke kořeni syntaktického stromu narazíme na vrchol označkovaný bud $\forall x$ nebo $\exists x$. V opačným případě nazveme tento výskyt volným. Kvantifikátor $\forall x$ nebo $\exists x$ váže všechny výskyty proměnné x, které jsou v syntaktickém stromu pod tím kvantifikátorem.*

(chyby jsou moje)

Substituce:

Budeme psát $A[x \leftarrow t]$ pro
výsledek toho že nahradíme
každý volný výskyt proměnné x
termem t ve výrazu A

Význam logických formulí (Semantika)

Interpretace dává každému predikátu a každému funkčnímu symbolu určitý význam (pozor na aritu a eventuální typy!).

Valuace/ohodnocení/kontext dává každé proměnné určitou hodnotu.

Pro interpretaci $\mathcal{I}$, valuaci σ, formuli ϕ: $\mathcal{I}, \sigma \models \phi$.

Může se přesně definovat, ale nemáme na to čas

$\models \phi$: formule ϕ obecně platí (nezávislé od interpretace/valuace).

V praxi většinou stačí znát intuitivní význam

Důležité je znát pravidla jak zacházet s logickými formuli

Zbytek dnešní přednášky: Logická pravidla která

- obecně platí (i za 1000 let)
- se můžou každodenně používat

Ekvivalence

"je ekvivalentně": formule mají stejný význam, jedna formula se může libovolně nahradit druhou (A, B, C stojí pro libovolné formule):

- A je ekvivalentně B pokud
  A a B se liší jen α-konverzí (Velebil 3.1.12)
- $A \wedge \mathbf{T}$ je ekvivalentně A
- $A \wedge \mathbf{F}$ je ekvivalentně $\mathbf{F}$
- $A \vee \mathbf{T}$ je ekvivalentně $\mathbf{T}$
- $A \vee \mathbf{F}$ je ekvivalentně A
- $A \wedge B$ je ekvivalentně $B \wedge A$
- $A \vee B$ je ekvivalentně $B \vee A$
- $A \wedge (B \vee C)$ je ekvivalentně $(A \wedge B) \vee (A \wedge C)$
- $A \vee (B \wedge C)$ je ekvivalentně $(A \vee B) \wedge (A \vee C)$
- $\neg\neg A$ je ekvivalentně A
- $\neg(A \wedge B)$ je ekvivalentně $\neg A \vee \neg B$
- $\neg(A \vee B)$ je ekvivalentně $\neg A \wedge \neg B$

Další ekvivalence

- $\exists x.\exists y.A$ je ekvivalentně $\exists y.\exists x.A$
- $\forall x.\forall y.A$ je ekvivalentně $\forall y.\forall x.A$
- $\neg\forall x.A$ je ekvivalentně $\exists x.\neg A$
- $\neg\exists x.A$ je ekvivalentně $\forall x.\neg A$
- $\forall x \,.\, A \wedge B$ je ekvivalentně $[\forall x.A] \wedge [\forall x.B]$
- $\exists x \,.\, A \vee B$ je ekvivalentně $[\exists x.A] \vee [\exists x.B]$
- $\forall x \,.\, A \vee B$ je ekvivalentně $[\forall x.A] \vee B$ pokud B neobsahuje x
- $\exists x \,.\, A \wedge B$ je ekvivalentně $[\exists x.A] \wedge B$ pokud B neobsahuje x

Pravidla ekvivalence:

- A je ekvivalentně A
- Pokud A je ekvivalentně B pak B je ekvivalentně A
- Pokud A je ekvivalentně B, a B je ekvivalentně C pak A je ekvivalentně C

Implikace, ekvivalence

$A \Rightarrow$ B je zkratka pro $\neg A \vee B$

$A \Leftrightarrow B$ je zkratka pro $A \Rightarrow B \wedge B \Rightarrow A$

Z toho plyne:

- $A \Rightarrow B$ je ekvivalentně $\neg B \Leftarrow \neg A$
- $A \Leftrightarrow B$ je ekvivalentně $\neg B \Leftrightarrow \neg A$

A teď splníme starý sen

Důkaz

Důkaz: Doklad že určitá formule ϕ platí ($\models \phi$)

Můžeme dokázat $A \Leftrightarrow B$ tím,
že dokážeme že A je ekvivalentně B.

Ale obecně to nestačí.

Příští přednáška: Obecná metoda dokazování.