NSWI183: Sémantika programů

1. Úvod

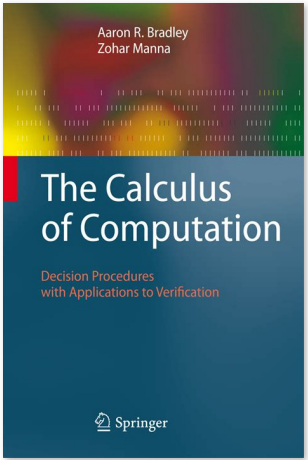
Jan Kofroň

MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

Department of
Distributed and
Dependable
Systems

Organizace kurzu

- Přednáška jednou týden
- Podmínka pro získání zápočtu:
  - Vypracování tří malých domácích úloh a jedné složitější
  - Odevzdání v termínu (v průběhu semestru)
- Kurz vytvořen podle knihy:
  A. R. Bradley, Z. Manna: The Calculus of Computation
- A podle Dafny tutorialů Rustana Leina



Sylabus

- Stručný přehled výrokové logiky a predikátových logik prvního řádu
- Složitost a rozhodnutelnost fragmentů
- Princip indukce a její využití při dokazování vlastností programů
- Pi – jednoduchý imperativní programovací jazyk
- PiVC – nástroj pro dokazování vlastností programů
- Částečná správnost (partial correctness)
- Úplná správnost (total correctness)
- Metody při specifikaci a dokazování vlastností
- Používání systému Dafny – prakticky použitelný nástroj

- … a hlavně hodně příkladů

Výroková logika (propositional logic)

- Základní logika pro zápis formulí
- Přímá korespondence s běžným životem (konjunkce, disjunkce, implikace, ekvivalence, negace)
- Jen Booleovské proměnné a logické spojky

Jazyk

- Nechť P je neprázdná spočetná množina výrokových proměnných (atomů, prvovýroků):

$$P = \{p, p_1, p_2, ...\}$$

- Jazyk výrokové logiky (nad P) obsahuje symboly:
  - Výrokové proměnné z P
  - Logické spojky $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
  - Závorky (,)

- Plus budeme používat logické konstanty $\bot$ jako zkratku za $p \wedge \neg p$ a $\top$ za $p \vee \neg p$, kde $p \in P$

FORMULE

Formule výrokové logiky se vytvářejí aplikací následujících pravidel:

1. Každá výroková proměnná z P je výrokovou formulí
2. Jsou-li p, q výrokové formule, pak následující jsou rovněž formule:

$$(\neg p), (p \wedge q), (p \vee q), (p \rightarrow q), (p \leftrightarrow q)$$

3. Každá výroková formule vznikne konečným počtem aplikací pravidel 1. a 2.

SÉMANTIKA VÝROKOVÉ LOGIKY

- Uvažujeme pouze dvě hodnoty – pravda ($\top$, 1) a nepravda ($\bot$, 0)
- Sémantika logických spojek je dána pravdivostní tabulkou

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

- Hodnota výroku (výrokové formule) je pak dána ohodnocením prvovýroků a opakovanou aplikací pravdivostní tabulky od nejjednodušších podformulí k původní formuli

Axiomy výrokové logiky

- K dokazování splnitelnosti (existuje ohodnocení prvovýroků takové, že formule má hodnotu pravda) nebo pravdivosti (při každém ohodnocení prvovýroků má formule hodnotu pravda) formule můžeme použít různé přístupy (např. metoda tabla nebo Hilbertovský systém)
- Z přednášky o logice znáte metodu tabla
- Axiomy výrokové logiky Hilbertovského systému jsou:

$$(p \to (q \to p)) \qquad \text{(A1)}$$

$$\Big((p \to (q \to r)) \to ((p \to q) \to (p \to r))\Big) \qquad \text{(A2)}$$

$$((\neg p \to \neg q) \to (q \to p)) \qquad \text{(A3)}$$

Odvozovací pravidla

- Odvozovací pravidla slouží spolu s axiomy při dokazování vět (tvrzení) výrokové logiky
- Nejsou nutně fixní, ale spolu s uvedenými axiomy budeme používat následující dvě:

Věta o dedukci:	$T \models p \to q \leftrightarrow T \cup \{p\} \models q$, kde T je množina formulí	(VD)
Modus ponens:	Z p a $p \to q$ odvodíme q	(MP)

METODY DŮKAZU

- Přímý důkaz – s použitím kombinace axiomů a odvozovacích pravidel
  - U implikace můžeme místo $A \rightarrow B$ dokazovat $\neg B \rightarrow \neg A$, pokud je to snazší
- Důkaz sporem – vhodný pro implikace, předpokládáme že platí antecedent (levá část) a neplatí konsekvent (pravá část)
- Důkaz indukcí – zobecnění vlastnosti pro mnoho případů
  - Detaily později

Logiky prvního řádu (First Order Logics – FOL)

- Proměnné výrokové logiky jsou Booleovské – mají hodnotu TRUE nebo FALSE
  - … nebo jsou volné – žádnou hodnotu (zatím) nemají
- Proměnné predikátové logiky prvního řádu mohou mít i „zajímavější“ hodnoty
  - Celá čísla, reálná čísla
- … a operace nad nimi
  - Porovnání – rovnost, uspořádání hodnot
- … a kvantifikátory

Teorie neinterpretovaných funkcí s rovností

- Bezesporná, nerozhodnutelná, úplná teorie
- Signatura: $\Sigma_E : \{=, a, b, c, ..., f, g, h, ..., p, q, r...\}$
- Velmi obecná teorie
- Obsahuje kromě logických spojek a rovnosti i funkční symboly
- Neinterpretované = nestaráme se o konkrétní výsledek aplikace funkčních symbolů
  - Ale požadujeme, aby se chovaly jako funkce: $\forall F, x, y.\ (x = y \rightarrow F(x) = F(y))$
- Funkce mohou mít více parametrů
- Zobecňují teorii

Peanova aritmetika

- Bezesporná, nerozhodnutelná, úplná teorie
- Signatura: $\Sigma_N : \{0, 1, +, \cdot, =\}$
- Axiomy:

$0 \neq x + 1$	(P1)
$x + 1 = y + 1 \rightarrow x = y$	(P2)
$x + 0 = x$	(P3)
$x + (y + 1) = (x + y) + 1$	(P4)
$x \cdot 0 = 0$	(P5)
$x \cdot (y + 1) = x \cdot y + x$	(P6)
$\varphi(0) \wedge \forall x. (\varphi(x) \rightarrow \varphi(x + 1)) \rightarrow (\forall x. \varphi(x))$ pro všechny formule φ jazyka Σ_N	(schéma indukce)

Presburgerova aritmetika

- Bezesporná, rozhodnutelná, úplná teorie
- Signatura: $\Sigma_N : \{0, 1, +, =\}$
- Axiomy:

$0 \neq x + 1$	(PR1)
$x + 1 = y + 1 \rightarrow x = y$	(PR2)
$x + 0 = x$	(PR3)
$x + (y + 1) = (x + y) + 1$	(PR4)
$\varphi(0) \wedge \forall x.\, (\varphi(x) \rightarrow \varphi(x + 1)) \rightarrow (\forall x.\, \varphi(x))$ pro všechny formule φ jazyka Σ_N	(schéma indukce)

Teorie listů

- Vhodná pro datové struktury jako jsou seznamu v LISPu
- Signatura: $\Sigma_{CONS} : \{cons, car, cdr, atom, =\}$
- Axiomy:

$\forall x.\, x = x$ (Reflexivita)

$\forall x, y.\, x = y \rightarrow y = x$ (Symetrie)

$\forall x, y, z.\, x = y \wedge y = z \rightarrow x = z$ (Transitivita)

$\forall x_1, y_1, x_2, y_2.\, x_1 = x_2 \wedge y_1 = y_2 \rightarrow cons(x_1, y_1) = cons(x_2, y_2)$

$\forall x, y.\, x = y \rightarrow car(x) = car(y)$

$\forall x, y.\, x = y \rightarrow cdr(x) = cdr(y)$

$\forall x, y.\, x = y \rightarrow (atom(x) \leftrightarrow atom(y))$

$\forall x, y.\, car(cons(x, y)) = x$

$\forall x, y.\, cdr(cons(x, y)) = y$

$\forall x.\, \neg atom(x) \rightarrow cons(car(x), cdr(x)) = x$

$\forall x, y.\, \neg atom(cons(x, y))$

Teorie polí

- Pole jsou další častou datovou strukturou
- Podobné neinterpretovaným funkcím, ale obsah se dá modifikovat
- Signatura: $\Sigma_A : \{\cdot[\cdot], \cdot\langle\cdot \lhd \cdot\rangle, =\}$
- $a[i]$ je binární funkce reprezentující čtení pole a na indexu i
- $a\langle i \lhd v\rangle$ je ternární funkce reprezentující zápis hodnoty v na index i pole a
- $=$ je binární predikát rovnosti
- Kromě axiomů reflexivity, symetrie a transitivity z teorie listů navíc:
  $\forall a, i, j.\, i = j \to a[i] = a[j]$
  $\forall a, v, i, j.\, (i = j \to a\langle i \lhd v\rangle[j] = v)$
  $\forall a, v, i, j.\, (i \neq j \to a\langle i \lhd v\rangle[j] = a[j])$

Kombinace teorií

- Často se hodí teorie kombinovat
- Je třeba dávat pozor na rozhodnutelnost a efektivitu rozhodovacích algoritmů
  - Například povolení (existenčních) kvantifikátorů může rozhodnutelnost zkomplikovat

Hoarova logika

- Princip „precondition“ a „postcondition“ ve verifikaci programů (kontrakty)
- Základní princip (Hoarova trojice):
  - Pro stav programu P (precondition)
  - Pro stav programu Q (postcondition)
  - Příkaz (statement) nebo posloupnost příkazů C změní stav z P na Q: $\{P\}C\{Q\}$
- P a Q nemusejí nutně zachycovat kompletní stav, ale třeba jen některou zajímavou část
  - To, co chceme, aby platilo, nějaká konkrétní vlastnost stavu Q, např. ve formě formule nějaké teorie
- To, jestli Q platí, je právě předmětem verifikace
- C musí být také formálně vyjádřeno v jazyku teorie (což nemusí být úplně triviální transformace)
- Pak se dá Hoarova trojice přepsat jako $P \wedge C \rightarrow Q$

Aplikace pro verifikaci programů

Hoarova logika se hodí při verifikaci programů:

1. Každou funkci (proceduru, metodu) opatříme předpokladem *P* a konsekvencí *Q* (kontrakty)
2. Dokážeme, že když platí *P*, tak po provedení těla funkce platí *Q*
3. Ověříme návaznost jednotlivých předpokladů a konsekvencí při volání funkcí
   - V místě, kde je volána nějaká funkce, jsou splněny její předpoklady
   - Po návratu z funkce předpokládáme, že platí její konsekvence

Příklad kontraktů – Mergesort

```
@pre  true
@post sorted(rv, 0, |rv|-1)
int[] MergeSort(int[] a)
{
  return ms(a, 0, |a|-1);
}
```

```
@pre  true
@post sorted(rv, l, u) && |rv|=|a_0|
int[] ms(int[] a_0, int l, int u) {
  int[] a := a_0;
  if (l>=u) return a;
  else {
    int m := (l + u) div 2;
    a := ms(a, l, m);
    a := ms(a, m + 1, u);
    a := merge(a, l, m, u);
    return a;
  }
}
```

Příklad kontraktů – Mergesort

```
@pre  true
@post sorted(rv, 0, |rv|-1)
int[] MergeSort(int[] a)
{
  return ms(a, 0, |a|-1);
}
```

```
@pre  true
@post sorted(rv, l, u) && |rv|=|a_0|
int[] ms(int[] a_0, int l, int u) {
  int[] a := a_0;
  if (l>=u) return a;
  else {
    int m := (l + u) div 2;
    a := ms(a, l, m);
    a := ms(a, m + 1, u);
    a := merge(a, l, m, u);
    return a;
  }
}
```

Příklad kontraktů – Mergesort

```
@pre  true
@post sorted(rv, 0, |rv|-1)
int[] MergeSort(int[] a)
{
  return ms(a, 0, |a|-1);
}
```

```
@pre  true
@post sorted(rv, l, u) && |rv|=|a_0|
int[] ms(int[] a_0, int l, int u) {
  int[] a := a_0;
  if (l>=u) return a;
  else {
    int m := (l + u) div 2;
    a := ms(a, l, m);
    a := ms(a, m + 1, u);
    a := merge(a, l, m, u);
    return a;
  }
}
```

Příklad kontraktů – Mergesort

```
@pre  true
@post sorted(rv, 0, |rv|-1)
int[] MergeSort(int[] a)
{
  return ms(a, 0, |a|-1);
}
```

```
@pre  true
@post sorted(rv, l, u) && |rv|=|a_0|
int[] ms(int[] a_0, int l, int u) {
  int[] a := a_0;
  if (l>=u) return a;
  else {
    int m := (l + u) div 2;
    a := ms(a, l, m);
    a := ms(a, m + 1, u);
    a := merge(a, l, m, u);
    return a;
  }
}
```

Příklad kontraktů – Mergesort

```
@pre  true
@post sorted(rv, 0, |rv|-1)
int[] MergeSort(int[] a)
{
  return ms(a, 0, |a|-1);
}
```

```
@pre  true
@post sorted(rv, l, u) && |rv|=|a_0|
int[] ms(int[] a_0, int l, int u) {
  int[] a := a_0;
  if (l>=u) return a;
  else {
    int m := (l + u) div 2;
    a := ms(a, l, m);
    a := ms(a, m + 1, u);
    a := merge(a, l, m, u);
    return a;
  }
}
```

KONTRAKTY PRO PROGRAMOVACÍ JAZYKY

- Mnoho knihoven implementující kontrakty pro Javu, C# a další jazyky k dispozici:
  - C#, .NET obecně: CodeContracts
  - Java: Java Modeling Language (JML)
  - Python: Nagini

- Většinou ve formě anotací, tedy speciálních komentářů:

```
//@ requires 0 < amount && amount + balance < MAX_BALANCE;
//@ assignable balance;
//@ ensures balance == \old(balance) + amount;
public void credit(final int amount)
{
  this.balance += amount;
}
```

KONTRAKTY PRO PROGRAMOVACÍ JAZYKY

- Mnoho knihoven implementující kontrakty pro Javu, C# a další jazyky k dispozici:
  - C#, .NET obecně: CodeContracts
  - Java: Java Modeling Language (JML)
  - Python: Nagini

- Většinou ve formě anotací, tedy speciálních komentářů:

```
//@ requires 0 < amount && amount + balance < MAX_BALANCE;
//@ assignable balance;
//@ ensures balance == \old(balance) + amount;
public void credit(final int amount)
{
  this.balance += amount;
}
```

Co je na tom těžké?

- Není těžké napsat nějaké kontrakty, je těžké napsat kontrakty tak, aby nebyly ani příliš silné, ani příliš slabé a aby do sebe zapadaly:
  - Příliš slabé předpoklady komplikují dokazování konsekvencí
  - Příliš silné předpoklady je těžké splnit při volání funkce
  - Příliš slabé konsekvence nesplňují předpoklady pro rekurzivní volání funkcí nebo komplikují dokazování konsekvencí volajících funkcí
  - Příliš silné konsekvence je těžké dokázat
- Později uvidíme příklady

SHRNUTÍ

- Ukázali jsme si, jak se dají formalizovat některé datové struktury programovacích jazyků a jak zapsat jejich vlastnosti ve formě formulí
- Důležité vlastnosti jednotlivých logik – rozhodnutelnost, úplnost
- Pro rozhodnutelné teorie můžeme implementovat automatické rozhodovací procedury a dokazovat tak vlastnosti programů automaticky
- Pro zbytek semestru důležitá zejména teorie polí, která zahrnuje rovnost a neinterpretované funkce

Úkol na příště

Dokažte, že v teorii polí platí následující formule, nebo nalezněte protipříklad, pokud neplatí:

- $(a\langle i \triangleleft e\rangle[j] = e) \rightarrow (i = j)$
- $(a\langle i \triangleleft e\rangle\langle j \triangleleft f\rangle[k] = g \wedge j \neq k \wedge i = j) \rightarrow (a[k] = g)$

ZÁVĚREČNÉ PRÁCE

V případě zájmu o oblast verifikací a analýzy programů mě prosím kontaktujte ohledně vedení bakalářských a diplomových prací, ročníkových a výzkumných projektů!

Jan Kofroň <jan.kofron@d3s.mff.cuni.cz>

https://d3s.mff.cuni.cz/students/topics/
https://d3s.mff.cuni.cz/research/verification/