NSWI183: Sémantika programů

8. Dafny I.

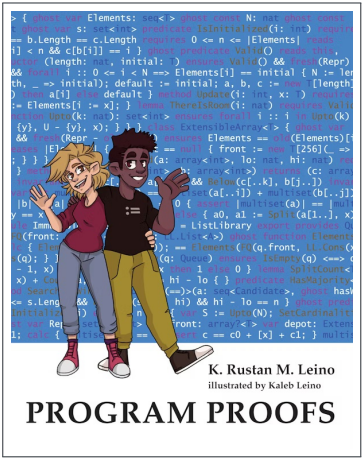
Jan Kofroň

MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

Department of
Distributed and
Dependable
Systems

Motivace

- Naučili jsme se základní principy pro ověřování vlastností kódu s použitím jednoduchého jazyka Pɪ a jeho anotací
- V následujících přednáškách se podrobně podíváme na prakticky použitelný systém – Dafny – pro reálné programovací jazyky a naučíme se ho používat
- Přednášky jsou inspirovány knihou a sérií přednášek od Rustana Leina



Verifikace skutečných programů

- Vývoj programů je náročný a programy obsahují chyby
- Je mnoho způsobů, jak chyby hledat a jak jim předcházet (viz předchozí přednáška)
- Pokud chceme k problému detekce a odstraňování chyb přistoupit systematicky, můžeme použít metody verifikace – to jsme viděli pomocí použití PIVC
  - Dříve (osmdesátá a devadesátá léta minulého století) se vlastnosti programů ověřovaly ručně – pomocí ručně konstruovaných důkazů
  - To trvalo velmi dlouho (roky), proto se např. v bankovnictví používaly některé systémy i více než deset let
    - ... až žádní zaměstnanci banky neuměli systémy udržovat
  - Dnes máme naštěstí k dispozici nástroje, které nám s důkazy pomohou

Příklady verifikovaných systémů

- Brzdící systém linky 14 pařížského metra (B)
- Mikrokernel operačního systému seL4 (Haskell, Isabelle/HOL, C)
- Překladač CompCert (Coq)
- Ironclad (Dafny)
- IronFleet (Dafny)

DAFNY – PŘIPOMENUTÍ

- DAFNY je programovací jazyk s podporou verifikace pomocí kontraktů
- Umožňuje vytvářet kód, který je následně ověřen vůči specifikaci
- Umožňuje nám *přeložit* kód do několika běžných programovacích jazyků:
  - C#
  - Java
  - JavaScript
  - Python
  - Go

Příklad – QuickSort I.

```
method qsort(a: array<int>, l: nat, u: nat)
  requires a.Length > 1
  requires 0 <= l <= u <= a.Length
  requires partitioned(a, 0, l, l, u)
  requires partitioned(a, l, u, u, a.Length)

  ensures sorted(a, l, u)
  ensures forall i: nat :: 0 <= i < l ==> a[i] == old(a[i])
  ensures forall i: nat :: u <= i < a.Length ==> a[i] == old(a[i])
  ensures partitioned(a, 0, l, l, u)
  ensures partitioned(a, l, u, u, a.Length)

  decreases u - l
  modifies a
{
  if l >= u - 1 {
  } else {
    var p_pivot := partition(a, l, u);
    qsort(a, l, p_pivot);
    qsort(a, p_pivot + 1, u);
  }
}
```

Příklad – QuickSort II.

```
predicate partitioned(a: array<int>, k: nat, l: nat, m: nat, n: nat)
  reads a
{
  forall i: nat, j: nat :: k <= i < l <= m <= j < n <= a.Length ==> a[i] <= a[j]
}

predicate sorted(a: array<int>, l: nat, u: nat)
  reads a
  requires 0 <= l <= u <= a.Length
{
  forall j, k :: l <= j <= k < u ==> a[j] <= a[k]
}

predicate eq(a: array<int>, b: array<int>, l: nat, u: nat)
  reads a
  reads b
  requires 0 <= l <= u <= a.Length
  requires a.Length == b.Length
{
  forall i: nat :: l <= i < u ==> a[i] == b[i]
}
```

Použití

Při psaní kódu lze verifikovat kontrakty, což můžeme nahlížet jako formu ladění během vývoje:

- `dafny verify qsort.dfy`

Když je kód dokončený a verifikovaný, můžeme ho přeložit do cílového jazyka:

- `dafny build --target=py --no-verify qsort.dfy`

A následně v cílovém jazyku spustit:

- `python __main__.py`

Pro vývoj lze použít plugin pro VSCode, který budeme používat při přednáškách

DAFNY – PŘEHLED

- Začneme úplně základními informacemi o DAFNY programech a z příkladů se budeme postupně dozvídat další
- DAFNY je imperativní objektový jazyk, podporující interakci s několika dalšími běžnými jazyky
- Motivací existence DAFNY je psaní programů, jejichž důležité vlastnosti můžeme přímo formálně ověřit a poté programy přeložit do některého běžného jazyka a spustit

ZÁKLADNÍ ANOTACE

- V principu se v DAFNY používají stejné anotace jako v PI
  - DAFNY je ale daleko složitější jazyk, anotací je tedy víc
- Klíčová slova odpovídající anotacím:
  - `requires`
  - `ensures`
  - `invariant`
  - `decreases`
  - `reads`
  - `modifies`
  - `ghost`
  - …

Anotace, precondition

- Specifikace anotací se v Dafny píše dovnitř metod a funkcí, ne vně jako v Pi
- Anotace se ověřuje staticky, tedy před spuštěním programu, za běhu se už v programu nevyskytuje

- Klíčové slovo `requires`:
  - Jedná se o klíčové slovo pro specifikaci *precondition*
  - Odpovídá anotaci `@pre` v Pi
  - Typicky zachycuje omezení vstupních parametrů

```
method Triple(x: int) returns (r: int)
  requires x % 2 == 0
  ensures  r == 3 * x
```

Postcondition

- Pro specifikaci *postcondition* se používá klíčové slovo `ensures`
- Odpovídá anotaci `@post` v Pi
- Typicky zachycuje vlastnosti návratové hodnoty
  - Metoda v Dafny může vracet n-tici návratových hodnot
- V Dafny je možné, aby metody měly vedlejší efekty, proto je aplikace `ensures` širší než v Pi
- Pomocí `old()` se lze odkázat k hodnotě proměnné v momentě vstupu do metody

```
method UpdateElements(a: array<int>)
  requires a.Length == 10
  modifies a
  ensures  old(a[4]) < a[4]
```

Invarianty

- Invarianty cyklů se v Dafny specifikují pomocí klíčového slova `invariant`
- Píší se před začátek těla cyklu (pod jeho „hlavičku“) jako v Pi a mají stejný význam – invariant musí platit před každým vstupem do těla cyklu a po poslední iteraci

```
while x % 2 == 1
  invariant 0 <= x <= 100
```

Terminace

- Terminace, ať už v případě rekurzivního volání nebo cyklů, je důležitá vlastnost programů
- Pro specifikaci fundované funkce se používá klíčové slovo `decreases`
- Pokud není funkce pro dokázání terminace specifikována, Dafny vygeneruje standardní výraz:
  - pro rekurzivní volání n-tici tvořenou parametry metody v použitém pořadí
  - pro cyklus s podmínkou `E < F` nebo `E <= F`: `F - E`
  - pro cyklus s podmínkou `E != F`: `if E < F then F - E else E - F`
- Tyto vygenerované terminační funkce jsou ve většině případů postačující

```
function SeqSum(s: seq<int>, lo: int, hi: int): int
  requires 0 <= lo <= hi <= |s|
  decreases hi - lo
{
  if lo == hi then 0 else s[lo] + SeqSum(s, lo + 1, hi)
}
```

METODY

- Metody jsou základní stavební kameny programu v DAFNY
- Mají vstupní a výstupní parametry a tělo složené z posloupnosti příkazů
- Před tělem metody se mohou nacházet anotace
- Hlavní metoda se jmenuje `Main()`

```
method sumdif(x: int, y: int) returns(sum: int, dif: int)
  ensures sum == x + y
  ensures dif == x - y
{
  sum := x + y;
  dif := x - y;

  // return x + y, x - y;
}
```

Příklady – trojnásobek a maximum a suma

FUNKCE

- Funkce jsou dalším základním prvkem v DAFNY
- Používají se k popsání sémantiky nějaké operace (funkce v matematickém smyslu) co nejjednodušším (funkcionálním) způsobem, bez ohledu na výpočetní složitost daného zápisu – „jde jen o výsledek“
- Funkce umožňují ověřovat výsledky výpočtů metod
- I když je můžeme zahrnout do implementace, často je k nim přistupováno jako ke specifikaci, která se za běhu programu nepoužívá

```
function Fib(n: nat): nat
{
  if n < 2 then n else Fib(n-1) + Fib(n-2)
}
```

GHOST VS. KOMPILOVANÝ KÓD

- DAFNY umožňuje psát kód, který se není součástí běhové verze programu
- Takovému kódu se říká *ghost* kód (funkce, metoda, proměnná, …)
- Kód, který se opravdu vykonává a je součástí běhové verze programu, se říká *compiled*
- Kompilovaný kód nesmí záviset na *ghost* kódu

```
ghost function Fib(n: nat): nat
{
  if n < 2 then n else Fib(n-1) + Fib(n-2)
}
```

Příklad – Fibonacciho posloupnost

*Jako **Fibonacciho posloupnost** je v matematice označována nekonečná posloupnost přirozených čísel, začínající 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, … (čísla nacházející se ve Fibonacciho posloupnosti jsou někdy nazývána Fibonacciho čísla), kde každé číslo je součtem dvou předchozích.*

[Wikipedie]

Shrnutí

- Viděli jsme základ programovacího jazyka Dafny
- V principech specifikace je blízký jazyku Pi
- Příště se podíváme na složitější příklady