NSWI183: Sémantika programů

8. Dafny III.

Jan Kofroň

MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

Department of
Distributed and
Dependable
Systems

DAFNY

- Viděli jsme základní koncepty a abstrakce programovacího jazyka DAFNY
- Dnes se seznámíme s dalšími technikami pro specifikaci v tomto prostředí
  - s ohledem na dokazování vlastností

LEMMA

- V některých případech se hodí dát dokazované vlastnosti jméno, abychom mohli důkaz použít na více místech programu
- A přesně k tomu slouží v DAFNY koncept lemmatu
- Dobře odpovídá dekompozici při uvažování o fungování programu a dokazování jeho vlastností

Příklad

- Uvažme následující funkci vracející větší číslo než je její argument:

```
function More(x: int): int {
  if x <= 0 then 1 else More(x - 2) + 3
}
```

- Opravdu to platí?
- Ověříme to právě pomocí lemmatu:

```
lemma Increasing(x: int)
  ensures x < More(x)
```

Lemma

- Lemmata vyjadřují svoje „znění“ právě pomocí postconditions (`ensures`)
- Stejně jako funkce mohou mít i preconditions, omezující případy, kdy lze lemma použít
- Abychom lemma v nějakém místě použili (na konkrétní parametry), je nutné ho zavolat
  - stejně jako normální funkci
- Lemmata jsou vlastně ghost funkce, tedy nekompilují se, ale můžeme je volat pro účely dokazování

Calc

- Pro přehlednější zápis důkazů, zejména vhodný pro důkazy aritmetických vlastností, můžeme použít příkaz `calc`:

```
calc {
       (x + y) * (x - y);
  ==   // distributivita * vzhledem k + a -
       x*x - x*y + y*x - y*y;
  ==   // komutativita násobení
       x*x - x*y + x*y - y*y
  ==   // x*y - x*y = 0
       x*x - y*y;
}
```

- Zde můžeme kromě rovnosti použít i nerovnosti (samozřejmě jen jedním směrem, aby to dávalo smysl)

Calc

- Někdy musíme specifikovat, proč lze daný krok provést, a to pomocí assertu nebo jiného lemmatu:

```
calc {
    ...
    x + Mult(x, y - 1);
== { MultCommutative(x, y - 1); }
    x + Mult(y - 1, x);
    ...
}
```

- A to je způsob, jaký známe z dokazování běžných matematických tvrzení

Verifikace na pozadí

- Pokud použijeme IDE (VSCode), verifikátor na pozadí ověřují každý výraz, který napíšeme, průběžně
- To ale platí jen v případě, že obsah celého souboru je syntakticky a typově správný
- Snadno pak vidíme, jestli jsou jednotlivé kroky v pořádku, například rekurzivní použití indukční hypotézy

Znalosti verifikátoru

- Systém Dafny zná některá triviální aritmetická fakta, například komutativitu sčítání, ale u složitějších, i když zjevných vlastností musíme explicitně důkaz napsat
- To je užitečné k uvědomění si, co jsou vlastně opravdu základní operace a skutečnosti, a co je třeba explicitně dokázat

Čtvrtá domácí úloha

- V Dafny naprogramujte Dijkstrův algoritmus pro hledání nejkratších cest ze zadaného vrcholu grafu:
  1. navrhněte vhodné datové struktury
  2. implementujte algoritmus
  3. dokažte co nejvíc (užitečných) vlastností metod a funkcí tvořících implementaci – ideálně to, že algoritmus skutečně nalezne nejkratší cestu ze zadaného vrcholu
- Termín pro odevzdání je **14. 2. 2025, 23:59**.