NSWI183: Sémantika programů

13. Dafny VI – Invarianty datových struktur

Jan Kofroň

MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

Department of
Distributed and
Dependable
Systems

Invarianty datových struktur

- Datové typy a typy obecně nám umožňují kontrolovat program nad rámec syntaxe
- Víme, že proměnná typu `List` může vypadat jako `Nil` nebo `Cons(head, tail)`
- To je užitečné, ale my můžeme jít ještě dál a specifikovat další (detailnější) vlastnosti týkající se typů
  - například u červeno-černého stromu můžeme pomocí invariantů specifikovat povolené počty červených a černých uzlů
- Invarianty jsou velmi důležité pro správný design programů a jejich správnost

PRIORITNÍ FRONTA

- Jako datovou strukturu pro tuto přednášku budeme používat *prioritní frontu*
- Prioritní fronta umožňuje přístup k minimálnímu, ne nutně prvnímu prvku:

```
module PriorityQueue {
  type PQueue

  function Empty(): PQueue
  predicate IsEmpty(pq: PQueue)
  function Insert(pq: PQueue, y: int): PQueue
  function RemoveMin(pq: PQueue): (int, PQueue)
    requires !IsEmpty(pq)
}
```

PRIORITNÍ FRONTA

- Naše definice prioritní fronty nedefinuje reprezentaci prvků (jestli to bude seznam, strom, ...)
- Proto je nutné mít možnost testovat prázdnost fronty pomocí predikátu `IsEmpty` (nebo jinak)
- Abychom mohli dále specifikovat, co vracejí jednotlivé funkce a predikáty, potřebujeme nějakou abstrakci naší prioritní fronty ve smyslu reprezentace jednotlivých prvků
  - máme několik možností (uspořádaný seznam, multimnožina (multiset), ...)
  - každá reprezentace má své výhody a nevýhody, my zvolíme multimnožinu

Abstrakce

```
ghost function Elements(pq: PQueue): multiset<int>

lemma EmptyCorrect()
  ensures Elements(Empty()) == multiset{}

lemma IsEmptyCorrect(pq: PQueue)
  ensures IsEmpty(pq) <==> Elements(pq) == multiset{}

lemma InsertCorrect(pq: PQueue, y: int)
  ensures Elements(Insert(y, pq)) == Elements(pq) + multiset{y}

lemma RemoveMinCorrect(pq: PQueue)
  requires !IsEmpty(pq)
  ensures var (y, pq') := RemoveMin(pq);
          IsMin(y, Elements(pq)) &&
          Elements(pq') + multiset{y} == Elements(pq)

ghost predicate IsMin(y: int, s: multiset<int>) {
  y in s && forall x :: x in s ==> y <= x
}
```

EXPORT

- Specifikace je z hlediska potenciálních klientů skoro hotová, musíme ještě definovat, co chceme exportovat:

```
module PriorityQueue {
  export
    provides PQueue, Empty, IsEmpty, Insert, RemoveMin
    provides Elements
    provides EmptyCorrect, IsEmptyCorrect
    provides InsertCorrect, RemoveMinCorrect
    reveals IsMin

  ...
}
```

Návrh datové struktury

- Pro implementaci prioritní fronty použijeme haldu
  - halda je binární strom, který reprezentuje prvky ve vnitřních uzlech
  - umožňuje tak odebírat minimum v čase $\mathcal{O}(\log n)$, **pokud** je strom vyvážený, tedy každý levý a pravý podstrom téhož prvku mají přibližně stejnou velikost
  - ke klasifikaci použijeme pojem *Braunova stromu* – levý a pravý podstrom jsou stejně velké nebo levý podstrom obsahuje o jeden prvek více než pravý

- Máme tedy tři vlastnosti:
  1. prvky ve vnitřních uzlech
  2. vlastnost haldy – prvek v uzlu je menší než prvky v podstromech
  3. vyváženost – velikosti levého a pravého podstromu se liší nejvýše o 1
- Z modulu PriorityQueue exportujeme predikát `Valid`, který platí pro prioritní frontu implementovanou jako `BraunTree`, pokud splňuje všechny tři výše uvedené vlastnosti:

```
ghost predicate Valid(pq: PQueue)
```

Návrh datové struktury

Upravíme předchozí lemmata tak, aby byla aplikovatelná jen na validní stromy, protože ostatní případy nás nezajímají:

```
lemma EmptyCorrect()
  ensures var pq := Empty();
    Valid(pq) && Elements(Empty()) == multiset{}

lemma IsEmptyCorrect(pq: PQueue)
  requires Valid(pq)
  ensures IsEmpty(pq) <==> Elements(pq) == multiset{}

lemma InsertCorrect(pq: PQueue, y: int)
  requires Valid(pq)
  ensures var pq' := Insert(pq, y);
    Valid(pq') && Elements(Insert(pq, y)) == Elements(pq) + multiset{y}

lemma RemoveMinCorrect(pq: PQueue)
  requires Valid(pq) && !IsEmpty(pq)
  ensures var (y, pq') := RemoveMin(pq);
  Valid(pq') && IsMin(y, Elements(pq)) && Elements(pq') + multiset{y} == Elements(pq)
```

Implementace datové struktury

- Nyní už máme specifikaci i návrh hotový a můžeme přistoupit k implementaci
- To obnáší definice jak operací nad datovou strukturou, tak predikátů a lemmat
  - detaily v přiložených materiálech
- Například predikát `Valid`:

```
ghost predicate Valid(pq: PQueue) {
  IsBinaryHeap(pq) && IsBalanced(pq)
}
```

Implementace prioritní fronty

Další kurzy na MFF – Navazující magisterské studium

- NSWI101: Modely a verifikace chování systémů (ZS)
  - Základní principy a algoritmy model checkingu
  - Modelování chování systémů a jejich následná verifikace
- NSWI132: Analýza programů a verifikace kódu (LS)
  - Zaměřen na praktickou zkušenost s nástroji pro verifikaci kódu (C/C++, Java, C#)
  - Principy a algoritmy pro analýzu zdrojového kódu
- NAIL094: Rozhodovací procedury a verifikace (LS)
  - Zaměřen na SAT a SMT solvery, algoritmy, optimalizace
- **Bakalářské a diplomové práce, softwarové a výzkumné projekty**
  - Pokud Vás téma verifikací zaujalo, neváhejte mě kontaktovat!