NSWI183: Sémantika programů

6. Strategie pro psaní anotací

Jan Kofroň

MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

Department of
Distributed and
Dependable
Systems

Anotace – částečná správnost

- Cílem, jak jsme již viděli, je nějakým způsobem dát do vztahu vstup funkce – parametry – a výstup funkce – návratovou hodnotu
- Pokud v anotaci funkce vyjádříme jen tento vztah, typicky to k dokázání nestačí, protože nám chybí zachycení sémantiky funkce, tedy
  - Invarianty cyklů
  - Anotace volaných funkcí
- Uvidíme, jakým způsobem najít správné (dostatečně silné) anotace

Konstrukce induktivních anotací

- Vytváření anotací vyžaduje lidský vhled do příslušné metody, stejně jako její vlastní implementace
- Jazyk anotací má i své limity – syntakticky správná formule musí padnout do rozhodnutelného fragmentu dané teorie
- Anotace je nejlépe psát současně s kódem, většinou potom nebereme v úvahu cíl verifikace a snažíme se je psát dostatečně obecně
- Anotace, zejména anotace funkcí, mají i dokumentační funkci
  - Typicky zachycují, co funkce dělá, nebo její důležité vlastnosti
- Invarianty cyklů musejí být induktivní, ne jen invariantní
  - Což je těžší úkol

Jak na psaní anotací

- Při vytváření anotací je dobré nejprve začít se základními fakty u invariantů cyklů
  - Meze pro index, meze pro pole
- Tato fakta jsou typicky potřeba, aby anotace byly induktivní

Příklad – Lineární vyhledávání

```
for
    @ L: l <= i && i <= u + 1
    (int i := l; i <= u; i := i + 1)    {
        if (a[i] = e) return true;
    }
```

Příklad – BubbleSort

```
for @ L1: -1 <= i < |a|
    (int i := |a| - 1; i > 0; i := i - 1) {
    for @ L2: 0 < i < |a| && 0 <= j <= i
        (int j := 0; j < i; j := j + 1) {
        if (a[j] > a[j + 1]) {
            int t := a[j];
            a[j] := a[j + 1];
            a[j + 1] := t;
        }
    }
}
```

Invarianty cyklů

- Je třeba nezapomínat, že invariant cyklu platí před každou iterací cyklu a zároveň po poslední iteraci
  - Před první iterací se předpokládá stav po inicializaci řídící proměnné cyklu
  - Po poslední iteraci zejména neplatí podmínka cyklu
- Je třeba mít na paměti okrajové případy
  - Proměnná může být záporná, pole může mít nulovou délku, atp.
- Napsat správný induktivní invariant může být obtížné a leckdy záleží na formuli, kterou chceme dokázat

Metoda předpokladů (Precondition method)

- Jedná se o název strategie pro psaní invariantů systematickým způsobem
- Někdy se jí říká „metoda zpětné substituce“ (backward substitution) nebo „metoda zpětné propagace“ (backward propagation)
- Jedná se o heuristiku, neboli metodologii, ne o algoritmus
  - Kdyby to byl algoritmus, mohli bychom ho implementovat a anotace generovat automaticky, což jak víme nelze (halting problem)
- Stále je potřebný lidský vhled do problému

METODA PŘEDPOKLADŮ

Metoda se skládá z následujících kroků:

1. Nalezneme fakt F, který je známý na místě $L(@L : F)$, ale nemá podporu v anotacích vyskytujících se před tímto místem
2. Opakujeme:
   a. Spočítáme nejslabší předpoklad (weakest precondition) faktu F zpětným průchodem až k předcházejícímu invariantu cyklu nebo anotaci funkce
   b. U každého nového místa L' zobecníme nová fakta a vytvoříme formuli $F'(@L' : F')$

Příklad – Lineární vyhledávání

```
@pre  0 <= l && u < |a|
@post rv <-> exists j. (l <= j && j <= u && a[j] = e)
bool LinearSearch(int[] a, int l, int u, int e) {
for
    @ L: l <= i && i <= u + 1
    (int i := l; i <= u; i := i + 1)
    {
        if (a[i] = e)
        return true;
    }
    return false;
}
```

Příklad – Lineární vyhledávání

- Předpokládejme následující základní cestu odpovídající nenalezení hledaného prvku (základní cesta číslo 5 ve třetí přednášce):

$$@L : F_1 : l \leq i \leq u + 1$$
$$S_1 : assume\ i > u$$
$$S_2 : rv := false$$
$$@post\ F_2 : rv \leftrightarrow \exists j . l \leq j \leq u \wedge a[j] = e$$

Příklad – Lineární vyhledávání

```
@pre  0 <= l && u < |a|
@post rv <-> exists j. (l <= j && j <= u && a[j] = e)
bool LinearSearch(int[] a, int l, int u, int e) {
    for
        @ L: l <= i && i <= u + 1

        (int i := l; i <= u; i := i + 1)
    {
        @ 0 <= i && i < |a|
        if (a[i] = e)  return true;
    }
    return false;
}
```

5

Příklad – Lineární vyhledávání

- Tato základní cesta odpovídá následující verifikační podmínce:

$$\{F_1\}S_1; S_2\{F_2\} : ((l \leq i \leq u + 1) \wedge (i > u)) \rightarrow \neg(\exists j. l \leq j \leq u \wedge a[j] = e)$$

- Tato formule není platná – antecedent implikace neříká nic o obsahu pole a, můžeme tedy těžko dokázat konsekvent
- Přepišme konsekvent jako $F_2 : \forall j. l \leq j \leq u \rightarrow a[j] \neq e$
  - „Pokud funkce vrátí false, všechny prvky pole a jsou v rozsahu $[l, u]$ různé od e“
- Tenhle fakt stále nemá podporu v předcházejících anotacích
- Spočítáme tedy nejslabší předpoklad formule F_2 pro předcházející invariant cyklu

Příklad – Lineární vyhledávání

$$
\begin{aligned}
& wp(F_2, S_1; S_2) \\
\leftrightarrow\ & wp(wp(F_2, rv := false), S_1) \\
\leftrightarrow\ & wp(F_2\{rv \mapsto false\}, S_1) \\
\leftrightarrow\ & wp(F_2\{rv \mapsto false\}, assume\, i > u) \\
\leftrightarrow\ & i > u \rightarrow F_2\{rv \mapsto false\} \\
\leftrightarrow\ & i > u \rightarrow \forall j.\, l \leq j \leq u \rightarrow a[j] \neq e
\end{aligned}
$$

Příklad – Lineární vyhledávání

- Podformule $(i > u)$ poslední formule $(i > u) \rightarrow (\forall j.\, l \leq j \leq u \rightarrow a[j] \neq e)$ a troška intuice nás vede k drobnému zobecnění: $\forall j.\, l \leq j < i \rightarrow a[j] \neq e$
- A to je náš induktivní invariant cyklu, spolu s mezemi pro proměnnou i:

$$(l \leq i \leq u) \wedge (\forall j.\, l \leq j < i \rightarrow a[j] \neq e)$$

- Obecně je dobrá strategie pro zobecnění anotací nahradit termy s pevnou hodnotou (u) termy, jejichž hodnota se v jednotlivých iteracích cyklu liší (i)

Indexování polí

- Stejný postup lze použít pro nalezení vhodných formulí omezujících hodnoty proměnných pro přístup do polí
- Postup však bývá jednodušší, protože víme, jaké proměnné musíme omezit a že hledáme nerovnosti
  - V případě invariantu cyklu obecně dopředu nevíme, jak má formule vypadat – potřebujeme zachytit sémantiku kódu ve větším detailu

Příklad – BubbleSort

```
@pre  true
@post sorted(rv, 0, |rv| - 1)
int[] BubbleSort(int[] a_0) {
  int[] a := a_0;
  for @ L1: -1 <= i && i < |a|
      (int i := |a| - 1; i > 0; i := i - 1) {
      for @ L2: 0 < i && i < |a| && 0 <= j && j < i
          (int j := 0; j < i; j := j + 1) {
          if (a[j] > a[j + 1]) {
            int t := a[j];
            a[j] := a[j + 1];
            a[j + 1] := t;
  }  }    }
  return a;
}
```

Příklad – BubbleSort

- Chceme nyní rozšířit anotace – invarianty cyklů – tak, abychom dokázali postcondition, tedy $sorted(rv, 0, |rv| - 1)$
- Odpovídající základní cesta je:

$$@L_1 : G : ?$$
$$S_1 : assume\ i \leq 0;$$
$$S_2 : rv := a;$$
$$@post\ F : sorted(rv, 0, |rv| - 1)$$

- Pokud spočítáme nejslabší předpoklad (weakest precondition) $wp(F, S_1, S_2)$, dostaneme formuli:

$$F' : i \leq 0 \rightarrow sorted(rv, 0, |rv| - 1)$$

Příklad – BubbleSort

- Formule F' nám (ne příliš překvapivě) říká, že po opuštění vnějšího cyklu je pole setříděné
- Řídící proměnná vnějšího cyklu se snižuje od $|a| - 1$ do 0
- Aplikujeme tedy stejný přístup jako v předchozím případě (nahrazení konstantních termů termy, jejichž hodnota se mění mezi jednotlivými iteracemi) a dostáváme:

$$G : sorted(a, i, |a| - 1)$$

- G triviálně platí při prvním vstupu do cyklu a i po každé iteraci
- Invariant cyklu by tedy měl vypadat takhle:

$$@L_1 : -1 \leq i \leq |a| \wedge sorted(a, i, |a| - 1)$$

Příklad – BubbleSort

- Dalším krokem je propagace invariantu z vnějšího cyklu do invariantu vnitřního cyklu
- Zde si již s pouhým výpočtem nejslabšího předpokladu nevystačíme a musíme využít vhledu do fungování funkce
- Nejslabší předpoklad pro vnitřní cyklus je $sorted(a, i, |a| - 1)$
  - Ten ale nezachycuje všechna potřebná fakta
- Po každé iteraci je hodnota na indexu j ta největší, na kterou jsme dosud narazili, neboli:

$$F : partitioned(a, 0, j - 1, j, j)$$

- Tuto novou informaci je třeba opět propagovat do anotace vnějšího cyklu

Příklad – BubbleSort

```
@pre  true
@post sorted(rv, 0, |rv| - 1)
int[] BubbleSort(int[] a_0) {
  int[] a := a_0;
  for @ sorted(a, i, |a| - 1) && partitioned(a, 0, i, i+1, |a|-1)
      (int i := |a| - 1; i > 0; i := i - 1) {
      for @ partitioned(a, 0, i, i + 1, |a| - 1) && 1 <= i &&
            0 <= j && j <= i && sorted(a, i, |a| - 1) &&
            partitioned(a, 0, j - 1, j, j)
          (int j := 0; j < i; j := j + 1) {
          if (a[j] > a[j + 1]) {
            int t := a[j];
            a[j] := a[j + 1];
            a[j + 1] := t;
  }  }    }
  return a;
}
```

Obecná strategie

- Obecná strategie využívá metodu předpokladů jako jeden ze svých kroků
- Nejprve rozložíme specifikaci funkce na jednotlivé atomické vlastnosti a pak pro každou vlastnost opakujeme následující postup:
  1. Zachytíme základní fakta
  2. Opakujeme:
     a. Použijeme metodu předpokladů k propagaci anotací
     b. Zformalizujeme informaci získanou vhledem do sémantiky funkce
- K tomu, co nám ještě chybí zachytit v anotacích, může posloužit protipříklad, který je výsledkem verifikace

Příklad – QuickSort

Shrnutí

- I když nám metoda předpokladů a obecná strategie mohou být do jisté míry vodítkem při psaní anotací, stále je nutné používat a formalizovat vhled do fungování funkcí
- Automatické metody pro konstrukci invariantů a anotací funkcí existují, ale nemohou fungovat ve všech případech z důvodu nerozhodnutelnosti problému
- Nejlepším způsobem, jak se naučit psát anotace, je samozřejmě praxe

Třetí domácí úloha

- Doplňte anotace do funkce SelectSort a ověřte správnost výpočtu v PiVC
  - Konečnost je již ověřená v druhé domácí úloze
- Termín pro odevzdání je **24. 11. 2025, 23:59**.
- Zdrojový kód funkcí naleznete na webu předmětu:

  `http://d3s.mff.cuni.cz/files/teaching/nswi183/selectsort.pi`