

3. Übungsblatt

Ausgabe: 28.04.16

Abgabe: 12.05.16

In diesem Aufgabenblatt wollen wir eine Erweiterung von `cat`, dem C0-Analysewerkzeug, implementieren, welche Beweise im Floyd-Hoare-Stil prüfen kann. Das benötigte Rahmenwerk finden Sie im Git-Repository der Veranstaltung.

3.1 Check This Proof!

15 Punkte

Die Beweise werden als *Annotationen* (speziell formatierte Kommentare, die nur für `cat` und nicht für den Compiler eine Bedeutung tragen) am Programm vermerkt. Abb. 1 zeigt als Beispiel das vollständig annotierte Programm zur Berechnung des ganzzahligen Teilers.

Wir wollen jetzt unser Analysewerkzeug so erweitern, dass es prüft, ob diese Annotationen den Regeln des Floyd-Hoare-Kalküls entsprechen. Grob gesprochen heißt das:

- Stehen zwei Annotationen `/** { P } */` und `/** { Q } */` direkt hintereinander, entspricht dieses der Anwendung der Weakening-Regel.
- Eine Anweisung (Zuweisung, Fallunterscheidung, Schleife) zwischen zwei Annotationen entspricht der jeweiligen Regel für die Zuweisung.
- Zwei Anweisungen direkt hintereinander sind nicht erlaubt.

Die Spezifikationen werden intern durch den Datentypen `Term` repräsentiert. Eine Funktion

```
sem(ctxt: Env, sp: Expr): Term
```

übersetzt einen Ausdruck der abstrakten Syntax (`Expr`) in einen `Term`. Für `Term` gibt es ferner die Methode

```
abstract class Term {  
  subst(x: Identifier, t: Term): Term  
}
```

die in einem `Term` jedes Vorkommen des Bezeichners `x` durch den `Term` `t` ersetzt.

Kern des Beweisprüfers (proof checker) sind zwei wechselseitig rekursive Funktionen, welche für eine einzelne Anweisung, oder für eine Sequenz von Anweisungen, prüft, ob die Abfolge von annotierten Spezifikationen und Anweisungen den Regeln des Floyd-Hoare-Kalküls entspricht:

```
checkTriple(ctxt: Env, pre: Term, s: Stmt, post: Term) : Option[List[Term]]  
checkTripleStmts(ctxt: Env, pre: Term, s: List[Stmt], post: Term): Option[List[Term]]
```

Der Rückgabewert ist `None`, falls das Programm nicht regelkonform ist, und ansonsten eine Liste von `Term`en, welche *Beweisverpflichtungen* sind, die sich aus der Anwendung der Weakening-Regel ergeben.

Insgesamt implementieren wir den Beweisprüfer als eine Funktion

```
checkAll(ctxt: Env, fundefs: List[FunDef]): List[(Identifier, List[Term])]
```

welche für eine gegebene Liste von Funktionsdefinitionen eine Liste aus Tupeln mit dem Namen der Funktion und der Liste der Beweisverpflichtungen zurückgibt.

```

void quotrem(int x, int y)
/** pre  x >= 0 &&& y > 0;
    post x == q*y+ r &&& 0 <= r &&& r < y;   */
{
  int q;
  int r;

  /** { x >= 0 &&& y > 0 } */
  /** { x == x &&& 0 <= x &&& y > 0 } */
  r= x;
  /** { x== r &&& 0 <= r &&& y > 0 } */
  /** { x== 0*y+ r &&& 0 <= r } */
  q= 0;
  /** { x== q*y+r &&& 0 <= r } */
  while (r >= y) {
    /** { x== q*y+ r &&& 0 <= r &&& r >= y } */
    /** { x== q*y+ r &&& y <= r } */
    /** { x== (q+ 1)*y+ (r- y) &&& 0 <= r-y } */
    r= r- y;
    /** { x== (q+1)*y+ r &&& 0 <= r } */
    q= q+1;
    /** { x== q*y+ r &&& 0 <= r } */
  }
  /** { x == q*y+ r &&& 0 <= r &&& !(r >= y) } */
  /** { x == q*y+ r &&& 0 <= r &&& r < y } */
}

```

Abbildung 1: Vollständig annotiertes Programm zur Berechnung des ganzzahligen Teilers

3.2 Vereinfachungen

5 Punkte

Die entstehenden Beweisverpflichtungen sind zum großen Teil sehr trivial. Wir wollen sie daher etwas vereinfachen, indem wir eine Funktion

`simplify(t: Term): Term`

implementieren, die Terme wie folgt vereinfacht:

1. Mit dem Extraktor `Implies` wird eine Implikation in Prämisse `prem` und Konklusion `conc` wie folgt zerlegt:¹

```
t match { case Implies(prem, conc) => ... }
```

Prämissen und Konklusion werden jetzt mit den folgende Schritten vereinfacht:

- (a) Ein Term t in konjunktiver Form $t = t_1 \wedge t_2 \wedge \dots \wedge t_n$ wird mit der Funktion

```
conjunctions(t: Term): Option[List[Term]]
```

in eine Liste von Konjunktion dekomponiert. Die einzelnen Konjunkte werden wie folgt vereinfacht:

- i. Explizite Negationen auflösen, d.h. aus $! a == b$ wird $a != b$ usw.;
 - ii. Die Relationen größer und größer-gleich werden durch kleiner und kleiner-gleich ersetzt;
 - iii. Reflexive Gleichungen und Ungleichungen (d.h. $a== a$, $a <= a$) werden durch `trueT` ersetzt;
- (b) Danach wird die Liste als ganzes vereinfacht, indem Vorkommen von `trueT` gestrichen, und mehrfache auftretende Konjunkte durch ein einzelnes ersetzt werden.
 - (c) Optional können jetzt noch Vereinfachungen wie $A \leq B \wedge B \leq A$ zu $A = B$ implementiert werden.

¹Ähnliche Extraktoren `And`, `Or`, `Not`, `Equals`, `NotEquals`, `Less`, `LEquals`, `Greater`, `GreaterEquals` für Konjunktion, Disjunktion usw. finden sich in `Logic.scala`.

2. Die vereinfachte Liste von Konjunktion der Konklusion wird vereinfacht, indem alle Konjunkte, die auch in den Prämissen vorkommen, gestrichen werden.
3. Danach wird aus den Listen von Konjunkten der Prämisse und Konklusion wieder eine Implikation (Funktion `implies` aus `Logic.scala`). Ist die Liste der Konjunkte der Konklusion leer, wurde die Beweisverpflichtung gezeigt, und kann entfallen.

Änderungen:

- 28.04.16 *Initiale Version.*
- 29.04.16 *Erläuterung zu Extraktor-Funktionen hinzugefügt.*