Universität Bremen

Korrekte Software: Grundlagen und Methoden SoSe 2016

Serge Autexier
Christoph Lüth

5. Übungsblatt

Ausgabe: 19.05.16
Abgabe: 30.05.16

5.1 *Verification Condition Generation* *10 Punkte*

In diesem Übungsblatt wollen wir das C-Analysewerkzeug `cat` um eine Funktion zur Berechnung der Verifikationsbedingungen erweitern, in Fachkreisen auch als *verification condition generation (vcg)* bekannt.

Wir konzentrieren uns erst einmal auf die Verifikationsbedingungen aus der schwächsten Vorbedingung. Die Eingabe ist dabei immer ein C-Programm, bei dem Funktionen mit Vor- und Nachbedingungen (erste optional) annotiert sind, und zwingend jede Schleife mit einer Invariante. Hier ist das notorische Beispiel:

```
int fac(int n)
/** pre 1 <= n;
    post p == faculty(n);
  */
{
 int p;
 int c;

 p= 1;
 c= 1;
 while (c<= n)
    /** inv p == faculty(c- 1) && c <= n+1;  */  {
    p= p*c;
    c= c+1;
    }

}
```

Unser Analysewerkzeug nimmt ein annotiertes Programm, und berechnet daraus die Verifikationsbedingungen mit Hilfe der approximierten schwächsten Vorbedingung (*awp*). Wir könnten dazu zwei Funktionen schreiben

```
def wvc(ctxt: Env)(s: Stmt, post: Term): List[Term]
def awp(ctxt: Env)(s: Stmt, post: Term): Term
```

die exakt den in der Vorlesung vorgestellten entsprechen (*7 Punkte*); aus Effizienzgründen, damit das Programm nicht zweimal traversiert werden muss, bietet es sich an, beide in *einer* Funktion zusammenzufassen (*10 Punkte*):

```
def awpvc(ctxt: Env)(s: Stmt, post: Term): (Term, List[Term])
```

Sie finden den entsprechenden Rahmen im git-Repository der Veranstaltung.

Die VCG ist in der Klasse WeakestPrecondition zu implementieren.

5.2 *Now Prove It!* *10 Punkte*

Das Werkzeug kann aus der Kommandozeile heraus aufgerufen werden. Die so erzeugten Beweisziele wollen wir jetzt in Isabelle beweisen:

1. Übersetzen Sie die Beweisverpflichtungen in eine Isabelle-Theorie mit sechs Beweiszielen.

   (*2 Punkte*)

2. Zeigen Sie die Beweisverpflichtungen der Funktion `quotrem` (Isabelle bringt alle benötigten arithmetischen Funktionen bereits mit).

   (*3 Punkt*)

3. Um die Beweisverpflichtugen für `fac` zu zeigen, definieren Sie eine Funktion *faculty* auf natürlichen Zahlen, welche rekursiv die Fakultät berechnet, und zeigen Sie folgendes Lemma:

   (*2 Punkte*)

   ```
   fun faculty :: "nat => nat"
   where ...

   lemma fac_unwind: "0 < m ==> faculty(m) = m*(faculty(m-1))"
   ...
   ```

4. Zeigen Sie jetzt die Beweisverpflichtungen für `fac`. Bei der zweiten Beweisverpflichtung werden Sie bemerken, dass die Invariante nicht stark genug ist. Stärken Sie die Invariante, so dass Sie auch die zweite Beweisverpflichtung zeigen können.

   (*3 Punkte*)

Hinweise:

- Wenn einfache Beweise in Isabelle nicht funktionieren liegt das meist daran, dass Isabelle den falschen Typen ableitet. Mit einer *Deklaration* der Form

  ```
  declare [[ show_types ]]
  ```

  kann Isabelle dazu gebracht werden, die Typen aller Variablen anzuzeigen.

- Der Beweis der dritten Beweisverpflichtung für die Fakultät ist langwieriger als die anderen. Hier ist es hilfreich, ein Lemma zu zeigen:

  ```
  lemma aux: "[| (c::nat) <= n+1 ; ~(c<= n) |] ==> c-1 = n"
  ```

  Ferner ist eventuell ein (mit `subgoal_tac "..."`) manuell eingefügtes Beweisziel notwendig.