Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Florian Franzmann, Martin Hoffmann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

4. Juni 2012

Überblick

ANSI/ISO C Specification Language (ACSL)

- Beschreibungssprache für C-Programme
- Annotationen für C-Funktionen
- „Design by Contract“
- Vor-/Nachbedingungen, Invarianten, Varianten
  $\leadsto$ wie in AUD
- verifizierbar mit frama-c
- heute nur der für die Aufgabe relevante Teil der Sprache
- mehr in der Vorlesung

Sprachelemente

- Annotationen als Kommentare im Quellcode
  $\leadsto$ so eng gefasst wie möglich
- `/*@ ... */`
- Aussagenlogik wie in C (`&& || !`)
- außerdem Implikation `==>`, Äquivalenz `<==>`, ...
- Quantorenlogik `\forall`, `\exists`

```
\forall integer k; 0 <= k < n ==> a[k] == k * k;
```

- mathematische Typen

  `integer` ganze Zahlen $\leadsto \mathbb{Z}$

  `real` reelle Zahlen $\leadsto \mathbb{R}$

  `boolean` `\true` und `\false`
- C-Datentypen

Wahrheitswerte

- Wahrheitswerte in C: Integer $\begin{cases} \text{falsch} & \text{wenn } 0 \\ \text{wahr} & \text{sonst} \end{cases}$
- das muß in ACSL dann auch entsprechend geprüft werden, d. h. `x == 0` bzw. `x != 0`
- Alternative: `enum` verwenden und auf `x == true` bzw. `x == false` testen

bool.h

```
1 typedef enum _bool {
2   false = 0,
3    true = 1
4 } bool;
```

Mathematische Funktionen

Mathematische Funktionen

```
1  integer \min(integer x, integer y);
2  integer \max(integer x, integer y);
3
4  real \min(real x, real y);
5  real \max(real x, real y);
6
7  real \pow(real x, real y);
8  real \sqrt(real x);
9
10 real \sin(real x);
11 real \cos(real x);
```

und noch viele andere $\rightsquigarrow$ Frama-C-ACSL-Handbuch

für die Übungsaufgabe nicht relevant

Bestandteile von Verträgen

`requires` Vorbedingung, muß beim Funktionsaufruf erfüllt sein

`assigns` Beschränkung von Schreibzugriffen
nicht vorhanden $\Rightarrow$ keine Einschränkung

`ensures` Nachbedingung
$\rightsquigarrow$ muß nach Rückkehr der Funktion erfüllt sein

`behavior` Fallunterscheidung

`\result` Rückgabewert der Funktion

Verträge

Schnittstelleneigenschaften

```
1  /*@
2     requires x >= 0;          Vorbedingung
3     assigns \nothing;         Funktion weisst nichts zu
4     ensures \result >= 0;     Nachbedingung
5     ensures \result * \result <= x;
6     ensures x < (\result + 1)
7                 * (\result + 1);
8   */
9  int32_t is_sqrt(int32_t x);
```

```
1
2  /*@
3     requires \valid(p);  p muss gültiger Zeiger sein
4     assigns *p;          Wert von p wird zugewiesen
5     ensures *p == \old(*p) + 1;  old greift auf Wert vor Aufruf zu
6   */
7   void incstar(int *p);
```

Verhalten

Fallunterscheidungen

```
1  /*@ ...
2      behavior p_changed:
3        assumes n > 0;        wird bei n > 0 aktiv
4        requires \valid(p);
5        assigns *p;
6        ensures *p == n;
7      behavior q_changed:
8        assumes n <= 0;       wird bei n ≤ 0 aktiv
9        requires \valid(q);
10       assigns *q;
11       ensures *q == n;
12   */
13 void f(int n, int *p, int *q) {
14   if (n > 0) *p = n;
15   else       *q = n;
16 }
```

Vollständigkeit von Verhalten I

complete

```
1  /*@
2     requires R;
3     behavior b1:
4       assumes A1;
5       ...
6     behavior bn:
7       assumes An;
8       ...
9     ....
10    complete behaviors b1, ..., bn;
11  */
```

- Dann gilt: `R ==> (A1 || ... || An)`
- ⇝ mindestens ein Verhalten trifft zu
- Kurzform: `complete behaviors;`

Vollständigkeit von Verhalten II

disjoint

```
1 /*@
2    disjoint behaviors b1, ..., bn;
3  */
```

- Dann gilt: `R ==> !(A1 && ... && An)`

⇝ nicht alle Verhalten gleichzeitig, mindestens zwei disjunkte

- Kurzform: `disjoint behaviors;`

Prädikate – Kapselung von Logik

```
1  /*@ ...
2     ensures \result >= 0;
3     ensures \result * \result <= x;
4     ensures x < (\result + 1) * (\result + 1);
5   */
6  int32_t is_sqrt(int32_t x);
```

```
1  /*@ predicate is_sqrt(int input, int output)
2        = output >= 0
3          && output * output <= input
4          && input < (output + 1) * (output + 1);
5   */
6  /*@ ...
7     ensures is_sqrt(x, \result);
8   */
9  int32_t is_sqrt(int32_t x);
```

Korrektheitsbeweis für Schleifen

Invariante ändert sich *nicht* von Schleifendurchlauf zu Schleifendurchlauf

Variante wird in jedem Schleifendurchlauf *dekrementiert*, hat nichtnegativen Startwert $\leadsto$ Terminierungsbeweis

`assigns` Aussagen über Seiteneffekte

`I` $\leadsto$ Invariante

`while(c) s;` `I` gelte für die Seiteneffekte von `c` gefolgt von `s`

`for(init; c; step) s;` `I` gelte für die Seiteneffekte von `c` gefolgt von `s` gefolgt von `step`

`do s while(c);` `I` gelte für die Seiteneffekte von `s` gefolgt von `c`

Schleifenannotationen

Varianten, Invarianten, Seiteneffekte

```
1  /*@
2     loop invariant 0 <= i <= n;
3     loop invariant \forall integer k; 0 <= k < i
4                    ==> b[k] == a[n - 1 - k];
5     loop   variant n - i;
6     loop assigns i, b[0..i - 1];
7   */
8  for (size_t i = 0; i < n; ++i) {
9    b[i] = a[n - 1 - i];
10 }
```

Lesenswertes zu ACSL/jessie/frama-c

- http://krakatoa.lri.fr/jessie.pdf
- http://frama-c.com
- http://frama-c.com/download/user-manual-Nitrogen-20111001.pdf
- http://frama-c.com/download/acsl-implementation-Nitrogen-20111001.pdf

Frama-C

- Frama-C bietet zwei Wegzeuge für WP-Beweisführung

  Jessie älter, besser ausgereift, nicht so schön zu bedienen

  wp schöne Benutzeroberfläche, nicht so leistungsfähig

Einmal ausführen, damit frama-c/why3 die Werkzeuge findet

```
1 % why3config
```

Jessie-Aufruf

```
1 % frama-c -jessie \
2           -jessie-atp why3ide \
3           -cpp-extra-args="-Iinclude" \
4           src/traffic_light.c
```

WP-Aufruf

```
1 % frama-c-gui -cpp-extra-args="-Iinclude" \
2               src/traffic_light.c
```

Table of Contents

Aufgabenstellung

1. Ampel-Programm verifizieren
   - für jede Funktion mindestens ein `assigns`, `ensures` und `requires`
2. `lower_bound()` verifizieren
   - Annotationsrümpfe von uns vorgegeben
   - für die Schleife: Invarianten, Variante und `assigns`-Aussage

Wichtig

Annotationen so eng wie möglich gefasst!

Funktionalität von `lower_bound()`

- Vorbedingung: Array ist aufsteigend sortiert
- Binärsuche auf Array
  - Suchraum wird in jedem Schritt halbiert
  - falls mittleres Element größer als Vergleichswert $\leadsto$ suche links weiter
  - sonst suche rechts davon
  - wiederhole bis Mächtigkeit der Suchmenge 1

`lower_bound()` liefert größten Index zurück, für den gilt

Alle Array-Elemente mit niedrigerem Index sind kleiner als der übergebene Wert

Implementierung `lower_bound()`

```
1  size_t lower_bound(const int *a, size_t n, int val)
2  {
3    size_t left = 0, right = n, middle = 0;
4    while (left < right) {
5      middle = left + (right - left) / 2;
6      if (a[middle] < val) {
7        left = middle + 1;
8      } else {
9        right = middle;
10     }
11   }
12   return left;
13 }
```

```
Wert a[k] | 1  4  5  8  9 11 14  18  21  22  23
Index k   | 0  1  2  3  4  5  6   7   8   9  10
                          /
lower_bound(a, 11, 10) == 5
```

Hilfestellung zur `lower_bound()`-Funktion I

- Annotationen zu `reverse_copy()` ähnlich zu `lower_bound()`

Formale Spezifikation `reverse_copy()`

```
1  /*@
2     predicate IsValidRange(int *a, integer n) =
3           (0 <= n) && \valid_range(a, 0, n - 1);
4   */
5  /*@
6     requires IsValidRange(a, n);
7     requires IsValidRange(b, n);
8     assigns b[0..(n - 1)];
9     ensures \forall integer i; 0 <= i < n
10                    ==> b[i] == a[n - 1 - i];
11  */
12 void reverse_copy(const int *a, size_t n, int *b);
```

Hilfestellung zur `lower_bound()`-Funktion II

Implementierung `reverse_copy()`

```
1  void reverse_copy(const int *a, size_t n, int *b) {
2    /*@
3       loop invariant 0 <= i <= n;
4       loop invariant \forall integer k; 0 <= k < i
5                      ==> b[k] == a[n - 1 - k];
6       loop   variant n - i;
7       loop assigns i, b[0..i - 1];
8     */
9     for (size_t i = 0; i < n; ++i) {
10      b[i] = a[n - 1 - i];
11    }
12 }
```

Fragen?