

# Verlässliche Echtzeitsysteme

## Übungen zur Vorlesung

Formale Verifikation mit Frama-C

Phillip Raffeck, Florian Schmaus, Simon Schuster

Friedrich-Alexander-Universität Erlangen-Nürnberg  
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)  
<https://www4.cs.fau.de>

Sommersemester 2020

## Programme zur Softwareverifikation

- Astrée
  - Frama-C
  - VeriFast
  - Dafny
  - SeaHorn
  - Coq
  - Isabelle/HOL
  - Agda
  - Idris



```

Dafny


```

method Int MyInt()
  ensures 3 * x + 1 <= a <= 4 * x + 1
  ensures MyInt(x) == 4 * Int(x) - 1
  {
    x := 3;
    a := 3 * x + 1;
    while x < a do
      a := 4 * x + 1;
      x := x + 1;
    end;
    return a;
  }
}

```


```



# Programme zur Softwareverifikation

- Astrée
- Frama-C
- VeriFast
- Dafny
- SeaHorn
- Coq
- Isabelle/HOL
- Agda
- Idris



...

- Bestimmt die schwächste notwendige Vorbedingung  $wp(S, Q)$ 
  - Für ein gegebenes imperatives Programmsegment  $S$
  - Um die ebenfalls gegebene Nachbedingung  $Q$  sicherzustellen
  - Dieser Sachverhalt wird beschrieben durch:  $P \Rightarrow wp(S, Q)$ 
    - Lässt sich die schwächste notwendige Vorbedingung  $wp(S, Q)$  aus der gegebenen Vorbedingung  $P$  folgern?



- Bestimmt die schwächste notwendige Vorbedingung  $wp(S, Q)$ 
  - Für ein gegebenes imperatives Programmsegment  $S$
  - Um die ebenfalls gegebene Nachbedingung  $Q$  sicherzustellen
  - Dieser Sachverhalt wird beschrieben durch:  $P \Rightarrow wp(S, Q)$ 
    - Lässt sich die schwächste notwendige Vorbedingung  $wp(S, Q)$  aus der gegebenen Vorbedingung  $P$  folgern?
- Das WP-Kalkül ist eine Rückwärtsanalyse
  - Sie beginnt mit der Nachbedingung und durchläuft das Programmsegment in umgekehrter Reihenfolge
  - „Sozusagen“ umgekehrter Einsatz der Regeln des Hoare-Kalküls



- Bestimmt die schwächste notwendige Vorbedingung  $wp(S, Q)$ 
  - Für ein gegebenes imperatives Programmsegment  $S$
  - Um die ebenfalls gegebene Nachbedingung  $Q$  sicherzustellen
  - Dieser Sachverhalt wird beschrieben durch:  $P \Rightarrow wp(S, Q)$ 
    - Lässt sich die schwächste notwendige Vorbedingung  $wp(S, Q)$  aus der gegebenen Vorbedingung  $P$  folgern?
- Das WP-Kalkül ist eine Rückwärtsanalyse
  - Sie beginnt mit der Nachbedingung und durchläuft das Programmsegment in umgekehrter Reihenfolge
  - „Sozusagen“ umgekehrter Einsatz der Regeln des Hoare-Kalküls
- Jeder Anweisung wird eine Prädikattransformation zugewiesen
  - Abbildung: Nachbedingung  $\mapsto$  notwendige schwächste Vorbedingung
  - Eine rückwärtige symbolische Ausführung des Programmsegments



- Bestimmt die schwächste notwendige Vorbedingung  $wp(S, Q)$ 
  - Für ein gegebenes imperatives Programmsegment  $S$
  - Um die ebenfalls gegebene Nachbedingung  $Q$  sicherzustellen
  - Dieser Sachverhalt wird beschrieben durch:  $P \Rightarrow wp(S, Q)$ 
    - Lässt sich die schwächste notwendige Vorbedingung  $wp(S, Q)$  aus der gegebenen Vorbedingung  $P$  folgern?
- Das WP-Kalkül ist eine Rückwärtsanalyse
  - Sie beginnt mit der Nachbedingung und durchläuft das Programmsegment in umgekehrter Reihenfolge
  - „Sozusagen“ umgekehrter Einsatz der Regeln des Hoare-Kalküls
- Jeder Anweisung wird eine Prädikattransformation zugewiesen
  - Abbildung: Nachbedingung  $\mapsto$  notwendige schwächste Vorbedingung
  - Eine rückwärtige symbolische Ausführung des Programmsegments
- Frama-C setzt WP-Kalkül ein: Nachweis der Schritte erfolgt über verschiedene Theorembeweiser und Löser: Alt-Ergo, Coq, Why3, Z3, ...

# Beispiel: WP-Kalkül Maximumsfunktion

WP : ?

if (a > b)

result = a;

else

result = b;

return result ;



# Beispiel: WP-Kalkül Maximumsfunktion

WP : ?

Auswertungsreihenfolge

if (a > b)

result = a;

else

result = b;

return result ;

Q :

\result  $\geq$  b  $\wedge$  \result  $\geq$  a

$$wp(\text{return } x, Q) \equiv Q[\backslash \text{result}/x]$$



# Beispiel: WP-Kalkül Maximumsfunktion

WP : ?

Auswertungsreihenfolge

if (a > b)

result = a;

else

result = b;

$result \geq b \wedge result \geq a$

return result ;

Q :

$\backslash result \geq b \wedge \backslash result \geq a$

$$wp(\text{return } x, Q) \equiv Q[\backslash result/x]$$



# Beispiel: WP-Kalkül Maximumsfunktion

WP : ?

Auswertungsreihenfolge

Q :

`if (a > b)`

`result = a;`

$a > b \Rightarrow result \geq b \wedge result \geq a$

`else`

`result = b;`

$\neg(a > b) \Rightarrow result \geq b \wedge result \geq a$

$result \geq b \wedge result \geq a$

`return result;`

$\backslash result \geq b \wedge \backslash result \geq a$

$wp(\text{if } b \text{ then } S1 \text{ else } S2, Q) \equiv$   
 $(b \Rightarrow wp(S1, Q)) \wedge (\neg b \Rightarrow wp(S2, Q))$



# Beispiel: WP-Kalkül Maximumsfunktion

WP : ?

Auswertungsreihenfolge

Q :

`if (a > b)`

`result = a;`

$a > b \Rightarrow result \geq b \wedge result \geq a$

`else`

$\neg(a > b) \Rightarrow b \geq b \wedge b \geq a$

`result = b;`

$\neg(a > b) \Rightarrow result \geq b \wedge result \geq a$

$result \geq b \wedge result \geq a$

`return result;`

$\backslash result \geq b \wedge \backslash result \geq a$

$wp(x = y, Q) \equiv Q[x/y]$



# Beispiel: WP-Kalkül Maximumsfunktion

WP : ?

Auswertungsreihenfolge

Q :

`if (a > b)`

$a > b \Rightarrow a \geq b \wedge a \geq a$

`result = a;`

$a > b \Rightarrow result \geq b \wedge result \geq a$

`else`

$\neg(a > b) \Rightarrow b \geq b \wedge b \geq a$

`result = b;`

$\neg(a > b) \Rightarrow result \geq b \wedge result \geq a$

$result \geq b \wedge result \geq a$

`return result;`

$\backslash result \geq b \wedge \backslash result \geq a$

$wp(x = y, Q) \equiv Q[x/y]$



# Beispiel: WP-Kalkül Maximumsfunktion

WP :

$$(a > b \Rightarrow a \geq b \wedge a \geq a) \wedge (\neg(a > b) \Rightarrow b \geq b \wedge b \geq a)$$

Auswertungsreihenfolge

Q :

`if (a > b)`

$$a > b \Rightarrow a \geq b \wedge a \geq a$$

`result = a;`

$$a > b \Rightarrow result \geq b \wedge result \geq a$$

`else`

$$\neg(a > b) \Rightarrow b \geq b \wedge b \geq a$$

`result = b;`

$$\neg(a > b) \Rightarrow result \geq b \wedge result \geq a$$

$$result \geq b \wedge result \geq a$$

`return result;`

$$\backslash result \geq b \wedge \backslash result \geq a$$

$$wp(\text{if } b \text{ then } S1 \text{ else } S2, Q) \equiv (b \Rightarrow wp(S1, Q)) \wedge (\neg b \Rightarrow wp(S2, Q))$$



# Beispiel: WP-Kalkül Maximumsfunktion

WP :

$$\Leftrightarrow \begin{array}{l} (a > b \Rightarrow a \geq b \wedge a \geq a) \wedge (\neg(a > b) \Rightarrow b \geq b \wedge b \geq a) \\ (a > b \Rightarrow a \geq b) \wedge (a \leq b \Rightarrow b \geq a) \end{array}$$

`if (a > b)`

$$a > b \Rightarrow a \geq b \wedge a \geq a$$

`result = a;`

$$a > b \Rightarrow result \geq b \wedge result \geq a$$

`else`

$$\neg(a > b) \Rightarrow b \geq b \wedge b \geq a$$

`result = b;`

$$\neg(a > b) \Rightarrow result \geq b \wedge result \geq a$$

$$result \geq b \wedge result \geq a$$

`return result;`

$$\backslash result \geq b \wedge \backslash result \geq a$$

Auswertungsreihenfolge

Q :



# Beispiel: WP-Kalkül Maximumsfunktion

WP :

$$(a > b \Rightarrow a \geq b \wedge a \geq a) \wedge (\neg(a > b) \Rightarrow b \geq b \wedge b \geq a)$$

$$\Leftrightarrow (a > b \Rightarrow a \geq b) \wedge (a \leq b \Rightarrow b \geq a)$$

$\Leftrightarrow \top$  // Gilt ohne Vorbedingung

`if (a > b)`

$$a > b \Rightarrow a \geq b \wedge a \geq a$$

`result = a;`

$$a > b \Rightarrow result \geq b \wedge result \geq a$$

`else`

$$\neg(a > b) \Rightarrow b \geq b \wedge b \geq a$$

`result = b;`

$$\neg(a > b) \Rightarrow result \geq b \wedge result \geq a$$

$$result \geq b \wedge result \geq a$$

`return result;`

$$\backslash result \geq b \wedge \backslash result \geq a$$

Auswertungsreihenfolge

Q :



- erlaubt Nachweise funktionaler Eigenschaften mittels wp-Kalkül
  - Prädikatenlogik erster Stufe
    - $\forall$ : \forallall
    - $\exists$ : \exists
    - $\Rightarrow$ : Implikation, ==>
    - Die aus C bekannten aussagenlogischen Verknüpfungen: !, &&, ||, ==, !=, ...
- Vor-/Nachbedingungen als Kommentare direkt im C-Code vor der betreffenden Funktion
  - //@ oder /\*@ \*/
- Vorbedingung: requires
- Nachbedingung: ensures
- Variablen, die durch die Funktion verändert werden: assigns
  - inklusive Arraybereiche: a[start..end]
  - Spezialfall \nothing
- Ergebnis des Funktionsaufrufs: \result



# Beispiel: Maximumsfunktion

```
S:int maximum(int a,int b) {  
    int result = INT_MIN;  
  
    if(a > b)  
        result = a;  
    else  
        result = b;  
  
    return INT_MAX;  
}
```



# Beispiel: Maximumsfunktion

*P* : wahr

```
S:int maximum(int a,int b) {
    int result = INT_MIN;

    if(a > b)
        result = a;
    else
        result = b;

    return INT_MAX;
}
```



# Beispiel: Maximumsfunktion

$P$  : wahr

```
S: int maximum(int a, int b) {  
    int result = INT_MIN;  
  
    if(a > b)  
        result = a;  
    else  
        result = b;  
  
    return INT_MAX;  
}
```

$Q$  :  $\text{result} \geq a \wedge \text{result} \geq b$



# Beispiel: Maximumsfunktion

$P$  : wahr

```
S: int maximum(int a, int b) {  
    int result = INT_MIN;  
  
    if(a > b)  
        result = a;  
    else  
        result = b;  
  
    return result;  
}
```

$Q$  :  $\text{result} \geq a \wedge \text{result} \geq b \wedge$   
 $(\text{result} == a \vee \text{result} == b)$



# Beispiel: Maximumsfunktion

*P* : wahr

```
S: int maximum(int a, int b) {  
    int result = INT_MIN;  
  
    if(a > b)  
        result = a;  
    else  
        result = b;  
  
    return result;  
}
```

*Q* :  $\text{result} \geq a \wedge \text{result} \geq b \wedge$   
 $(\text{result} == a \vee \text{result} == b)$

```
/*@  
  assigns \nothing;  
  
  ensures \result >= a;  
  ensures \result >= b;  
  ensures \result == a || \result == b;  
*/  
int maximum(int a, int b) {  
    int result = INT_MIN;  
    if (a > b) {  
        result = a;  
    } else {  
        result = b;  
    }  
    return result;  
}
```



```
// Swap array[0] with pointer other, do not modify the rest of array
```

```
void swap_first_element(int *array, size_t length, int *other) {
    int tmp = array[0];
    array[0] = *other;
    *other = tmp;
}
```



```
// Swap array[0] with pointer other, do not modify the rest of array

/*@ requires \valid(array) && \valid(other);
   requires \base_addr(array) != \base_addr(other);
   assigns array[0], *other;
   ensures array[1 .. (length - 1)] == \old(array[1 .. (length - 1)]);
   ensures *other == \old(array[0]);
   ensures array[0] == \old(*other); */
void swap_first_element(int *array, size_t length, int *other) {
    int tmp = array[0];
    array[0] = *other;
    *other = tmp;
}
```

- Übergebene Zeiger sind gültig: `\valid(ptr)`
- Speicherobjekte hinter Zeigern überlappen sich nicht:  
`\base_addr(ptr1) != \base_addr(ptr2)`
- `assigns`: Nur bestimmte (außen sichtbare) Variablen werden verändert
- `ensures`: Zusammenhänge zwischen Werten vor (`\old(var)`) und nach (`var`) der Ausführung anfordern



- Gesucht:  $wp(\text{while } (B) \{S\}; Q)$
- Wann ist die Berechnung korrekt?
  - Stellt  $Q$  her  $\leadsto$  Schleifen**invariante**
  - Terminiert  $\leadsto$  Schleifen**variante**
- Schleifenvariante: Streng monoton fallender, **nichtnegativer** Ausdruck  
In Frama-C: `//@ loop variant length - i;`
- Schleifeninvariante: Ausdruck, der vor der Schleife als nach jedem Durchlauf des Schleifenkörpers B gilt  
In Frama-C: `//@ loop invariant i % 2 == 0;`
- Ferner: Schleife überschreibt nur bestimmte Werte  
In Frama-C: `//@ loop assign i, j, *ptr;`
- Beispiel:  
`int i = 0;  
while(i < 30) { i++; }`
  - Schleifenvariante:  $30 - i \geq 0$
  - Schleifeninvariante:  $0 \leq i \leq 30$

Die Schleifenvariante gilt vor, während und nach der Schleife

Nach der Schleife ist die Schleifenbedingung falsch  
 $\leadsto \neg(i < 30) \wedge (0 \leq i \leq 30) \leadsto i == 30$



# Beispiel: Maximum in Liste finden

```
S:int findMax(int *a, size_t length) {
    int max = a[0];
    for (size_t i = 0; i < length; i++) {
        if (a[i] > max) {
            max = a[i];
        }
    }
    return max;
}
```



# Beispiel: Maximum in Liste finden

$P: a \neq \text{NULL} \wedge \text{length} > 0$

```
S:int findMax(int *a, size_t length) {
    int max = a[0];
    for (size_t i = 0; i < length; i++) {
        if (a[i] > max) {
            max = a[i];
        }
    }
    return max;
}
```

$Q: \forall k. k \geq 0 \wedge k < \text{length}$

$\Rightarrow a[k] \leq \text{\result}$

$\exists k. k \geq 0 \wedge k < \text{length}$

$\Rightarrow a[k] == \text{\result}$

```
/*@
requires \valid(a);
requires \valid(a +(0 .. length));
requires length > 0;
assigns \nothing;
ensures \forall size_t i;
    0 <= i < length ==> a[i] <= \result;
ensures \exists size_t i;
    0 <= i < length ==> a[i] == \result;
*/
int findMax(int *a, size_t length) {
    int max = a[0];
    size_t index = 0;
/*@
loop invariant \forall size_t k;
    0 <= k < i ==> a[k] <= max;
loop invariant a[index] == max;
loop invariant 0 <= index < i;
loop invariant \exists size_t k;
    0 <= k < i ==> a[k] == max;
loop assigns max, index, i;
loop variant length - i;
*/
    for (size_t i = 1; i < length; i++) {
        if (a[i] > max) {
            max = a[i];
            index = i;
        }
    }
    return max;
}
```



- „Wiederverwendbare“ aussagenlogische Formulierung
  - ~ logische Prädikate: predicate
- ~ „Funktionen“ auf ACSL-Ebene, erleichtern Formulierung von Bedingungen
- Beispiel: Maximum einer Sequenz

```
/*@ predicate is_max_of_seq(int max, int *seq, int start, int end) =
    \forall int i; start <= i < end ==> max >= seq[i];
 */
/*@ ...
   ensures is_max_of_seq(\result, a, (int)0, length);
 ...
 */
int findMax(int *a, int length) {
...
    /*@ loop invariant is_max_of_seq(max, a, (int)0, i);
    ...
    */
    for (int i = 0; i < length; i++) {
    ...
    }
    return max;
}
```



# Frama-C: Datenstrukturen beschreiben

- Invarianten über der Datenstruktur definieren

- Jede Methode:

- Setzt Gültigkeit der Invarianten voraus
- Erhält Sie nach Ausführung

~ Bei Kapselung: Korrektheit der Datenstruktur sichergestellt

- Beispiel: Konto

```
struct account {  
    int balance;  
    int credit_limit;  
};  
/*@ requires \valid(a);  
    ensures valid_account(a); */  
void init(struct account *a) {  
    a->balance = 0; a->credit_limit = 0;  
}
```

- Invariante: Kreditrahmen wird nie verletzt:

```
/*@ predicate valid_account(struct account *a) =  
    \valid(a) // Gültiger Zeiger  
    && a->balance >= a->credit_limit; // Kreditrahmen nicht verletzt  
*/
```

- Alle Methoden erfordern und erhalten die Invariante:

```
/*@ requires valid_account(a);  
    requires amount > 0;  
    ensures valid_account(a); */  
bool withdraw(struct account *a, int amount) {  
    /* ... */  
    return true;  
}  
/*@ requires valid_account(a);  
    requires amount > 0;  
    ensures valid_account(a); */  
void deposit(struct account *a, int amount) {  
    /* ... */  
}
```

~ Solange das Konto nur per withdraw und deposit modifiziert wird, kann der Kreditrahmen nie verletzt werden



- Oft gelten Nachbedingungen nur für bestimmte Eingabewerte

```
/*@ requires i != INT_MIN;
   ensures i < 0 ==> \result == -i;
   ensures i >= 0 ==> \result == i; */
int abs(int i) {
    if (i >= 0) return i;
    else         return -i;
}
```

- Behaviors beschreiben Verhalten in bestimmten Kontexten:

- assumes: Voraussetzungen, die das Verhalten aktiviert
- ensures: Nachbedingung, die die Funktion dann einhält

*Eingabe ist negativ*

behavior negative:

assumes  $i < 0$ ;

ensures  $\text{\result} == -i$ ;

*Eingabe ist positiv*

behavior positive:

assumes  $i \geq 0$ ;

ensures  $\text{\result} == i$ ;

- complete behaviors: Die Beschreibung ist vollständig  
Behaviors beschreiben das Verhalten für alle Aufrufkontakte
- disjoint behaviors: Die beschriebenen Verhalten überlappen sich nicht



# Frama-C: Beispiel Behaviors

```
#include <limits.h>

/*@ requires i != INT_MIN;
   behavior negative:
     assumes i < 0;
     ensures \result == -i;
   behavior positive:
     assumes i >= 0;
     ensures \result == i;
   complete behaviors;
   disjoint behaviors; */
int abs(int i) {
  if (i >= 0) return i;
  else         return -i;
}
```



- Die Frama-C-Gui bietet keinen Editor an!
- Reihenfolge der Annotationen z.T. relevant. Empfehlung:

## *Funktionen*

- requires
- assigns
- ensures

## *Schleifen*

- loop invariants
- loop assigns
- loop variants

- assert() wird als nicht terminierend angenommen  
~> Frama-C assert verwenden: //@ assert x == 1;

- Mehrere Annotationen immer in einen gemeinsamen Block /\*@ \*/

- Weitere Informationen:

- Frauenhofer Fokus: ACSL-By-Example:

<https://github.com/fraunhoferfokus/acsl-by-example>

- A. Blanchard: Introduction to C program proof with Frama-C and its WP plugin:  
<https://allan-blanchard.fr/publis/frama-c-wp-tutorial-en.pdf>





Edsger W. Dijkstra.

Guarded commands, nondeterminacy and formal derivation of programs.

*Communications of the ACM*, 18(8):453–457, August 1975.

