9. AuD Tafelübung T-C3

Simon Ruderich

22. Dezember 2010

wp-Kalkül

wp-Kalkül

- ermöglicht es die Korrektheit von Programmen mathematisch zu beweisen
- sinnvoll für sicherheitskritische Systeme (AKWs, etc.)
- Vorbedingung = P, Nachbedingung = Q
- wp = weakest precondition (schwächste Vorbedingung) für die Q (gerade) noch zutrifft
- Programm wird vom Ende her geprüft, Q bekannt
- prinzipiell einfache Ersetzungsregeln, bei Schleifen zusätzliche Bedingungen

Ersetzungsregeln

Zuweisungen

```
a = b + 3;
a += 3 * b;
// Q : a = 5
```

$$wp(\text{"a = b + 3; a += 3 * b"}, a = 5) \equiv$$
$$wp(\text{"a = b + 3; a = a + 3 * b"}, a = 5) \equiv$$
$$wp(\text{"a = b + 3;"}, a + 3 \cdot b = 5) \equiv$$
$$(b + 3 + 3 \cdot b = 5) \equiv$$
$$(4b = 2) \equiv$$
$$(b = 0.5)$$

Ersetzungsregeln

If-Abfragen

```
if (a > 0) {
    a = a + 3;
} else {
    a = ++a - a++ - 5;
    a *= -1;
}
// Q : a = 5
```

$$[(a > 0) \wedge wp(\text{"a = a + 3"}, a = 5)] \vee$$
$$[(a \leq 0) \wedge wp(\text{"a = ++a - a++ - 5; a *= 1"}, a = 5)] \equiv$$
$$[(a > 0) \wedge (a = 2)] \vee [(a \leq 0) \wedge (-5 = -5))] \equiv$$
$$[(a = 2) \vee (a \leq 0)]$$

Ersetzungsregeln

Switch-Case

```
switch (a) {
    case 5:
        a = 3;
    case 2:
        a = 50;
    default:
        a = 5;
}
// Q : a = 5
```

Ersetzungsregeln

Switch-Case

```
switch (a) {
    case 5:
        a = 3;
    case 2:
        a = 50;
    default:
        a = 5;
}
// Q : a = 5
```

≡

if

```
if (a == 5) {
    a = 3;
    a = 50;
    a = 5;
} else if (a == 2) {
    a = 50;
    a = 5;
} else {
    a = 5;
}
```

Algebraische Regeln

Algebraische Regeln

A, B, C sind algebraische Ausdrücke, z. B. $A \equiv (a = 5 \vee a < 3)$
a ist ein Variable

- $\wedge \mathrel{\widehat{=}} \cdot$, $\vee \mathrel{\widehat{=}} +$ (Ausklammern, Ausmultiplizieren, Priorität)
- $(A \wedge B) \vee (A \wedge C) \equiv A \wedge (B \vee C)$
- $A \vee (\neg A \wedge B) \equiv A \vee B$
- $[(a > 0) \wedge B) \vee [(a \leq 0) \wedge B] \equiv B \wedge [(a > 0) \vee (a \leq 0)] \equiv B$
- $\mathit{true} \wedge A \equiv A$
- $\mathit{false} \wedge A \equiv \mathit{false}$
- $\mathit{true} \vee A \equiv \mathit{true}$
- $\mathit{false} \vee A \equiv A$

Schleifen

Schleifen

- Schleifen sind schwerer zu prüfen
- mit Hilfe von Invariante und Variante aber möglich

Schleifeninvariante bleibt bei jedem Durchlauf konstant

Schleifenvariante wird bei jedem Durchlauf erniedrigt

Schleifen

Schleifen

- Schleifeninvariante I, Schleifenvariante V
- Code vor der Schleife A, Code in der Schleife S
- Schleifenbedingung b

1. I gilt vor der Schleife: $wp(A, I) \Rightarrow P$ (oder `true`)
2. I gilt nach jedem Schleifendurchlauf: $I \wedge b \Rightarrow wp(S, I)$
3. Q gilt nach der Schleife: $I \wedge \neg b \Rightarrow Q$
4. Schleife terminiert:
   z. B. $I \Rightarrow V \geq 0$, $I \wedge b \wedge V = z \Rightarrow wp(S, V < z)$

1.-3. erfüllt: partiell korrekt, 4. erfüllt: total korrekt

Beispiel Ersetzungen

Beispiel

```
int d = c - ++b + 1;
int e = a - --b;
b = c++ + ++d;
a = b -  e;
// Q: a > b > c
```

P : ?

Beispiel Ersetzungen

Beispiel

```
int d = c - ++b + 1;
int e = a - --b;
b = c++ + ++d;
a = b -  e;
// Q: a > b > c
```

$P: \quad a < b < c$

Beispiel Schleifen

$P \equiv n \geq 1$

```
private static final long fib(int n) {
    long fib1 = 1, fib2 = 1;
    int fib3 = n - 1;
    while (fib3 > 0 && fib3--> 0) {
        fib2 = fib1 + (fib1 = fib2);
    }
    return fib1;
}
```

$Q \equiv fib1 = F(n)$

mögliche Schleifeninvarianten

- $fib3 = fib1 + fib2$
- $fib1 = F(n - fib3) \wedge fib2 = F(n - fib3 + 1) \wedge fib3 \geq 0$
- $fib2 = fib1 + fib2 \wedge 0 \leq fib3 \leq n$

Beispiel Schleifen

$P \equiv n \geq 1$

```
private static final long fib(int n) {
    long fib1 = 1, fib2 = 1;
    int fib3 = n - 1;
    while (fib3 > 0 && fib3-- > 0) {
        fib2 = fib1 + (fib1 = fib2);
    }
    return fib1;
}
```

$Q \equiv fib1 = F(n)$

mögliche Schleifeninvarianten

- $fib3 = fib1 + fib2$
- $fib1 = F(n - fib3) \wedge fib2 = F(n - fib3 + 1) \wedge fib3 \geq 0$ ✓
- $fib2 = fib1 + fib2 \wedge 0 \leq fib3 \leq n$

Beispiel Schleifen

$P \equiv n \geq 1$

```
private static final long fib(int n) {
    long fib1 = 1, fib2 = 1;
    int fib3 = n - 1;
    while (fib3 > 0 && fib3-- > 0) {
        fib2 = fib1 + (fib1 = fib2);
    }
    return fib1;
}
```

$Q \equiv fib1 = F(n)$

mögliche Schleifenvarianten

- $fib2$
- $fib2 - fib1$
- $fib3$

Beispiel Schleifen

$P \equiv n \geq 1$

```
private static final long fib(int n) {
    long fib1 = 1, fib2 = 1;
    int fib3 = n - 1;
    while (fib3 > 0 && fib3--> 0) {
        fib2 = fib1 + (fib1 = fib2);
    }
    return fib1;
}
```

$Q \equiv fib1 = F(n)$

mögliche Schleifenvarianten

- $fib2$
- $fib2 - fib1$
- $fib3$ ✓

Fragen

Fragen?