Transformationelle Programmentwicklung

- Beweise in herkömmlicher Mathematik:

$$(a+b)^2 = (a+b)(a+b) = a(a+b) + b(a+b) = a^2 + 2ab + b^2$$

- Äquivalenzumformung, evtl. mit Bedingungen

- ... oder Implikation.

- Transformationelle Programmentwicklung: Umformung von *Programmen*

Transformationelle Programmentwicklung: Ein Beispiel

```
fib n   == if n<= 1 then 1 else fib(n-1) + fib(n-2)
——————— ⇓ ———————————————
fib n   == f n 1 0
f n a b == a*(fib n)+ b*(fib (n+1))
——————— ⇓ ——————————————— Unfolding
f n a b == a*(fib n)+ b*(if n+1<= 1 then 1 else (fib n)+(fib (n-1)))
——————— ⇓ ———————————————
f n a b == if n+1<= 1 then a*(fib n)+b
                      else a*(fib n)+ b*((fib n)+(fib (n-1)))
——————— ⇓ ———————————————
f n a b == if n<= 0 then a+b else b*(fib (n-1))+(a+b)*(fib n)
——————— ⇓ ——————————————— Folding
fib n   == f n 1 0
f n a b == if n= 0 then a+b else f (n-1) b (a+b)
```

Transformationelle Programmentwicklung: Grundlagen I

- $SP_1 \rightsquigarrow SP_2 \rightsquigarrow \ldots \rightsquigarrow SP_n$

  SP_i Spezifikationen, $\rightsquigarrow$ Verfeinerungsrelation:

- Korrektheitserhaltende Verfeinerung: $SP_1 \models P \Rightarrow SP_2 \models P$

- Modelinklusion: $Mod(SP_2) \subseteq Mod(SP_1)$

- $SP_i \rightsquigarrow SP_{i+1}$ durch Anwendung von *Transformationsregeln*.

Transformationelle Programmentwicklung: Grundlagen II

Eine *Transformationsregel* $\forall P_1, \ldots, P_n.\ A \Rightarrow I \rightsquigarrow O$ besteht aus

- Parametern $P_1, \ldots, P_n$,
- Eingabemuster I,
- Ausgabemuster O,
- Anwendungsbedingungen A

Transformationelle Programmentwicklung: Grundlagen III

Anwendung einer Transformationsregel $\forall\, P_1, \ldots, P_n.\ A \Rightarrow I \leadsto O$ auf Spezifikation SP

1. Instantiierung der Parameter (durch Benutzer)
2. Eingabemuster wird auf Subterm T in $SP = C[t]$ gepaßt: Substitution σ mit $\sigma(I) = T$
3. Neue Spezifikation ist $SP' \stackrel{def}{=} C[\sigma(O)]$ mit $SP \leadsto SP'$
4. Instantiierte Anwendungsbedingungen $\sigma(A)$ werden *Beweisverpflichtungen*

Verschiedene Transformationsregeln

- Logische Regeln: Fold, Unfold

  $A = B \Rightarrow A \rightsquigarrow B$
  $A = B \Rightarrow B \rightsquigarrow A$

- *Designregeln*: Spezifikationen ausführbar(er) machen:

  Einbettung *(Embedding)*,
  Endliche Differenz (*Finite Differencing*),
  Nachbedingungsspaltung *(Split of Postcondition)*,
  Dekomposition,
  ... (siehe Partsch, *Specification and Transformation of Programs*)

Transformationsregeln: Einbettung

$\forall E, K.$

```
fun f x == ?H x
```

$\Downarrow$ { `y= ?H x --> y= K x E`

```
fun f x   == g x E
fun g x y == K x y

f :: M ==> N
H :: M ==> N
g :: M* Y ==> N
E :: Y
K :: M* Y ==> N
```

Beispiele: Fibonacci, Fakultät.

Transformationsregeln: Einbettung, generalisiert

$\forall\, E, K, i, j.$

```
fun f x == ?H x
```

$$\frac{\texttt{fun f x == ?H x}}{\begin{array}{l}\texttt{fun f x   == j (g (i x) E)}\\ \texttt{fun g x y == K x y}\end{array}} \Downarrow \quad \{\ \texttt{y= ?H x --> y= j (K (i x) E)}$$

```
fun f x   == j (g (i x) E)
fun g x y == K x y
```

```
f :: M ==> N
H :: M ==> N
g :: X* Y ==> Z
E :: Y
K :: X* Y ==> Z
i :: M ==> X
j :: Z ==> N
```

Transformationsregeln: Fallunterscheidung

$\forall P.$
```
@y. ?R x y
```
$$\overline{}\Downarrow\overline{}$$
```
if P x y then @y. ?R x y else @y. ?R x y
```

$\forall Z.$
```
if ?P then ?X else ?Y
```
$$\overline{}\Downarrow\overline{}\{$$ `?P & (z= X) --> z= Z`
```
if ?P then ?Z else ?Y
```

$\forall Z.$
```
if ?P then ?X else ?Y
```
$$\overline{}\Downarrow\overline{}\{$$ `~ ?P & (z= Y) --> z= Z`
```
if ?P then ?X else Z
```

Transformationsregeln: Dekomposition

$\forall\, Q, c'.$

$$\frac{\texttt{@y. ?P (?c x) y}}{\texttt{c' (@z. Q x z)}} \Downarrow \begin{cases} \texttt{Q x z --> ?P (?c x) (c' z)} \\ \texttt{? y. ?P (?c x) y --> ? z. Q x z} \end{cases}$$

```
c  :: M ==> N
P  :: (N* P) set
c' :: R ==> P
Q  :: (M* R) set
```