Wiederholung: Rekursion

Beispiel: Potenzfunktion z^n

- Rekursionsgleichungen:
  $x^0 = 1$
  $x^n = x \cdot x^{n-1}$ für $n > 0$
- $7^3 = 7 \cdot (7^2) = 7 \cdot (7 \cdot 7^1) = 7 \cdot (7 \cdot (7 \cdot 7^0)) = 7 \cdot (7 \cdot (7 \cdot 1))$
- Verbinden mit Hilfe eines Konditionals:
  $x^n =$ if $n > 0$ then $x \cdot x^{n-1}$ else 1
-
```
fun potenz (x:int, n:int) :  int =
          if n>0 then x*potenz(x,n-1) else 1
```

Selbstanwendung

Wiederholung: Rekursion

```
fun potenz (x:int, n:int) :  int =
        if n>0 then x*potenz(x,n-1) else 1
```

Ausführungsprotokoll:

```
potenz(4,2) = if 2 > 0 then 4* potenz(4,2-1) else 1
= if true then 4 * potenz(4,2-1) else 1
= 4 * potenz(4, 2-1)
= 4 * potenz(4,1)
= 4 * (if 1 > 0 then 4*potenz(4,1-1) else 1)
= 4 * (if true then 4*potenz(4,1-1) else 1)
= 4 * (4* potenz(4,1-1))
= 4 * (4* (if 0 > 0 then 4*potenz(4, 0-1) else 1))
= 4 * (4* (if false then 4*potenz(4, 0-1) else 1))
= 4*(4*1)
= 4*4
= 16
```

Wiederholung: Rekursion

```
fun potenz (x:int, n:int) :  int =
        if n>0 then x*potenz(x,n-1) else 1
```

- Verkürztes Ausführungsprotokoll:

  potenz(4,2) = 4 * potenz(4,1)
  = 4 * (4 * potenz(4,0))
  = 4 * (4 * 1)
  = 16

- Rekursionsfolge:

  potenz(4,3) $\longrightarrow$ potenz(4,2) $\longrightarrow$ potenz(4,0)

Wiederholung: Divergenz

Ein Programm kann

- regulär terminieren:
  Ausführung endet nach endlich vielen Schritten:
  `3 div 5`
- divergieren: Ausführung endet nie:
  `fun p (x:int) : int = p x  val y = p 5`
- abbrechen aufgrund von Benutzer: `Strg` `C`
- abbrechen aufgrund von Laufzeitfehler: `3 div 0`
- abbrechen wegen Speichererschöpfung:
  `fun q (x:int) : int = 0 + q x  val y = q 5`

Wiederholung: Natürliche Quadratwurzeln

$$\lfloor \sqrt{n} \rfloor = \max\{k \in \mathrm{N} \mid k^2 \leq n\} = \min\{k \in \mathrm{N} \mid k^2 > n\} - 1$$

- Idee: Fange bei $k = 1$ an, rechne solange bis $k^2 > n$
  - Hilfsfunktion: $w : \mathrm{N} \times \mathrm{N} \to \mathrm{N}$
    $w(k, n) = \min\{i \in \mathrm{N} \mid i \geq k \text{ und } i^2 > n\}$.
  - Rekursionsgleichungen:
    - $w(k, n) = k$ für $k^2 > n$
    - $w(k, n) = w(k + 1, n)$ für $k^2 \leq n$

Standardstrukturen

- Math.sqrt: real → real
- Math.sin: real → real
- Math.asin: real → real
- Math.exp: real → real
- Math.pow: real * real → real
- Math.ln: real → real
- Math.log10: real → real
- Math.pi: real
- Math.e: real

- Real.fromInt: int → real
- Real.round: real → int
- Real.floor: real → int
- Real.ceil: real → int

Syntaktische Gleichungen: Ausdrücke

- <Ausdruck> ::=
  <atomarer Ausdruck>
  | <Anwendung>
  | <Konditional>
  | <Tupelausdruck>
  | <Let-Ausdruck>
  | (<Ausdruck>)

- <atomarer Ausdruck> ::=
  <Konstante>
  | <Bezeichner>

- <Anwendung> ::=
  <Operatoranwendung>
  | <Prozeduranwendung>
  | <Projektion>

Syntaktische Gleichungen: Ausdrücke

- <Operatoranwendung> ::=
  <einstelliger Operator> <Ausdruck>
  | <Ausdruck> <zweistelliger Operator> <Ausdruck>

- <Konditional> ::=
  if <Ausdruck> then <Ausdruck> else <Ausdruck>

- <Tupel-Ausdruck> ::=
  (<Ausdruck> , ... , <Ausdruck>)

- <Let-Ausdruck> ::= let <Program> in <Ausdruck> end

Syntaktische Gleichungen: Deklarationen

- <Programm> ::= <Deklaration> ...<Deklaration>
- <Deklaration> ::=
  <Val-Deklaration>
  | <Prozedurdeklaration>
- <Val-Deklaration> ::= val <Val-Muster> = <Ausdruck>
- <Val-Muster> ::=
  <Bezeichner>
  | (<Bezeichner> ... <Bezeichner>)
- <Prozedurdeklaration> ::=
  fun <Bezeichner> <Argumentmuster> = <Ausdruck>
  | fun <Bezeichner> <Argumentmuster> : <Typ>
  = <Ausdruck>
- <Argumentmuster> ::= (<Argumentspezifikation>
  ... <Argumentspezifikation>)
- <Argumentspezifikation> ::= <Bezeichner> : <Typ>

Syntaktische Gleichungen: Typen

- <Typ> ::=
  <atomarer Typ>
  | <Prozedurtyp>
  | <Tupeltyp>
  | (<Typ>)

- <atomarer Typ> ::=
  int
  | real
  | bool
  | unit

- <Prozedurtyp> ::= <Typ> -> <Typ>

- <Tupeltyp> ::= <Typ> * ... * <Typ>