Validierungstechniken

Hybride Programmiersprachen
Daniel Krompass
Berlin, 2009

Singuläre Validierung durch Typdefinition

```
# Jedes Datum ist eine Ganzzahl mit spezifischen Eigenschaften
public static type (D,d):=Date as Int      (oder imports)

    # Die Menge aller Daten ist gegeben durch.
    public static func (~) x:=Int -> Bool
        return (x == 0) || (IsYear a=x.Right 4, 4) &&
        (IsMonth b=x.Right 2, 2) && IsDay a, b, x.Right 2

    # Für ein Element gilt die Gleichheit.
    public func (~) x:=Int -> Bool
        return x == d

    # Für gültige Werte aus Year,  Month und Day ist ein Element gegeben durch:
    public func ctor a:=D.Year, b:=D.Month, c:=D.Day -> D
        return D (c.Right 4) + (b.Right 2) + a.Right 2

    # Andernfalls nimmt das Element einen initialen Wert an.
    public func ctor any -> D
        return D 0
```

Singuläre Validierung durch Typdefinition

```
public static type (D,d):=Date as Int

   ...

   public type Year as Int
      public static func (~) x:=Int -> Bool
         return x >= 0

   public type Month as Int
      public static func (~) x:=Int -> Bool
         return x.Between 1, 12

   public type Day as Int
      public static func (~) c:=Int -> Bool
         return c.Between 1, 31

   ...
```

- Die atomare Validierung ist oftmals nur eine Abschätzung von Wertebereichen.

Parametrische Validierung durch Typdefinition

```
public static type (D,d):=Date as Int

    public static func (~) x:=Int -> Bool
        return x.Between 0, 99991231

    public func (~) x:=Int -> Bool
        return x == d

    public func ctor a:=D.Year, b:=D.Month, (c:=D.Day a, b) -> D
        return D (c.Right 4) + (b.Right 2) + a.Right 2

    public func ctor any -> D
        return D 0

    public static func IsValidDay a:=Int, b:=Int, c:=Int -> Bool
    where a >= 0 && b.Between 1, 12
        ...

    public static func IsValidDay any -> Bool
    return False
```

Parametrische Validierung durch Typdefinition

```
public static type (D,d):=Date as Int

   ...

   public type Year as Int
      public static func (~) x:=Int -> Bool
         return x >= 0

   public type Month as Int
      public static func (~) x:=Int -> Bool
         return x.Between 1, 12

   public type Day a:=D.Month, b:=D.Year as Int
      public static func (~) c:=Int -> Bool
         return D.IsValidDay a, b, c

   ...
```

Parametrische Validierung mit Nachbedingung

```
public static type (D,d):=Date as Int

    public static func (~) x:=Int -> Bool
        return x.Between 0, 99991231

    public func (~) x:=Int -> Bool
        return x == d

    public func ctor a:=Int, b:=Int, c:=Int -> D
    where a >= 0 && (b.Between 1, 12) && IsValidDay a, b, c
        return D (c.Right 4) + (b.Right 2) + a.Right 2

    public func ctor any -> D
        return D 0

    ...
```

Operative (Casting-) Validierung

```
public static type (D,d):=Date as Int

    public static func (~) x:=Int -> Bool
        return x.Between 0, 99991231

    public func (~) x:=Int -> Bool
        return x == d

    public func ctor a:=Int, b:=Int, c:=Int -> D
        a = Year a
        b = Month b
        c = Day a, b, c
        return D (c.Right 4) + (b.Right 2) + a.Right 2

    public func ctor any -> D
        return D 0

    ...
```

Operative (Casting-) Validierung

```
public static type (D,d):=Date as Int

   public type Y:=Year as Int
      public static func (~) x:=Int -> Bool
         return x >= 0
      public ctor a:=Int -> Y
         return a.Abs

   public type M:=Month as Int
      public static func (~) x:=Int -> Bool
         return x.Between 1, 12
      public ctor b:=Int -> M
         return Math.Min (Math.Max 1, b), 12

   public type X:=Day a:=D.Month, b:=D.Year as Int
      public static func (~) c:=Int -> Bool
         return D.IsValidDay a, b, c
      public ctor a:=D.Month, b:=D.Year, c:=Int -> X
         ...
```