5.– 8. September 2011
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Polyglott

Trends in Programmiersprachen

Michael Wiedeking

MATHEMA Software GmbH

A# Ada APL ASP ASP.NET: ASM to IL AsmL Basic BETA Boo BlueDragon C C# Cω C++ Cat Ceylon CIL Closure Cobol CQL CULE.NET E# Eiffel F# Forth Fortran Groovy Haskell IronPython Java JScript.NET Lego.NET Lisp L# leXico LOLCODE Lua.NET M# Mercury Metaphor MixNet Mondrain Nermerle Oberon Ook# OMeta# Pan# Pascal PerlNET PHP Prolog Ruby Python Scala Smalltalk S# Scala SML.NET Spec# Tachy TickleSharp Zonnon

Zweckmäßig

```
 10  INPUT "Geben Sie bitte Ihren Namen ein"; A$
 20  PRINT "Guten Tag, "; A$
 30  INPUT "Wie viele Sterne möchten Sie?"; S
 35  S$ = ""
 40  FOR I = 1 TO S
 50      S$ = S$ + "*"
 55  NEXT I
 60  PRINT S$
 70  INPUT "Möchten Sie noch mehr Sterne?"; Q$
 80  IF LEN(Q$) = 0 THEN GOTO 70
 90  L$ = LEFT$(Q$, 1)
100  IF (L$ = "J") OR (L$ = "j") THEN GOTO 30
110  PRINT "Auf Wiedersehen";
```

Funktional

$$f : (\mathbb{Z}, \mathbb{Z}) \to \mathbb{Z}$$

$$f : (x, y) \mapsto x + y$$

- let tuple = (42, "Hello world!");;

*val tuple : int * string*

- let (num, str) = tuple;;

val num : int val str : string

- **type** Expr =

| Binary **of** string * Expr * Expr

| Variable **of** string

| Constant **of** int;;

- **let** v = Binary("+", Variable "x", Constant 10);;

val v : Expr

```
➢ let rec eval x = match x with
| Binary(op, l, r) –>
                let (lv, rv) = (eval l, eval r)
                if (op="+") then    lv + rv
                elif (op = “–") then          lv – rv
                else failwith "Unknonw operator!“
| Variable(var) –> getVariableValue var
| Constant(n) –> n;;
val eval : Expr –> int
```

- **let** nums = [1; 2; 3; 4; 5];;

val nums : list<int>

- **let rec** sum list = **match** list **with**

| h::tail –> (sum tail) + h

| [] –> 0

val sum : list<int> –> int

- **let rec** sumAux acc list = **match** list **with**

```
    | h::tail      -> sumAux (acc + h) tail
    | []           -> acc
```

val sum : int -> list<int> -> int

- **let** sum list = sumAux 0 list

val sum : list<int> -> int

- **let** createAdder n = (**fun** arg –> n + arg);;

val createAdder : int –> int –> int

- **let** add10 = createAdder 10;;

val add10 : int –> int

- add10 32;;

val it : int = 42

- **let** add a b = a + b;;

val add : int –> int –> int

- **let** add10 = add 10;;

val add10 : int –> int

- **let** nums = [1; 2; 3; 4; 5];;

val nums : list<int>

- **let** odds = List.filter (**fun** n –> n%2 <> 0) nums;;

val odds : list<int> = [1; 3; 5]

- **let** squares = List.map (**fun** n –> n * n) odds;;

val squares : list<int> = [1; 9; 25]

- **let** nums = [1; 2; 3; 4; 5];;

val nums : list<int>

- **let** odds_plus_ten = nums
  |> List.filter (fun n –> n%2 <> 0)
  |> List.map (add 10);;

val odds_plus_ten : list<int> = [11; 13; 15]

- (fst >> String.uppercase) ("Hello world", 123);;

val it : string = "HELLO WORLD"

- **let** data = [("Jim", 1); ("John", 2); ("Jane", 3)];;

*val data : (string * int) list*

- data |> List.map (fst >> String.uppercase);;

val it : string list = ["JIM"; "JOHN"; "JANE"]

Lesbar

LOLCODE – Hai World

```
HAI

CAN  HAS  STDIO?

VISIBLE  "HAI WORLD!"

KTHXBYE
```

```
HAI
CAN HAS STDIO?
I HAS A VAR
IM IN YR LOOP
UP VAR!!1
VISIBLE VAR
IZ VAR BIGGER THAN 10? KTHXBYE
IM OUTTA YR LOOP
KTHXBYE
```

LOLCODE – Dateizugriff

```
HAI
CAN HAS STDIO?
PLZ OPEN FILE "LOLCATS.TXT"?
AWSUM THX
VISIBLE FILE
O NOES
INVISIBLE "ERROR!"
KTHXBYE
```

LOLCODE – Eingabe

```
HAI

CAN  HAS  STDIO?

I  HAS  A  VAR

GIMMEH  VAR

VISIBLE  "You said "  N  VAR  N  " !!"

KTHXBYE
```

Deklarativ

<TAG\b[^>]*>(.*?)</TAG>

Grammatik

```
prog:   stat+ ;

stat
    :   expr NL
    |   ID '=' expr NL
    |   NL ;

expr:
    multExpr (
            '+' multExpr |
            '–' multExpr
    )* ;

multExpr:
    atom ('*' atom)* ;
```

Prolog – Fakten schaffen

```
mann(adam).
mann(tobias).
mann(frank).
frau(eva).
frau(daniela).
frau(ulrike).
```

```
vater(adam, tobias).
vater(tobias, frank).
vater(tobias, ulrike).
mutter(eva, tobias).
mutter(daniela, frank).
mutter(daniela, ulrike).
```

```
mann(adam).
mann(tobias).
mann(frank).
frau(eva).
frau(daniela).
frau(ulrike).
vater(adam, tobias).
vater(tobias, frank).
vater(tobias, ulrike).
mutter(eva, tobias).
mutter(daniela, frank).
mutter(daniela, ulrike).
```

```
?- mann(tobias).

yes.

?- frau(tobias).

no.

?- mann(heinrich).

no.

?- frau(heinrich).

no.
```

```
mann(adam).
mann(tobias).
mann(frank).
frau(eva).
frau(daniela).
frau(ulrike).
vater(adam, tobias).
vater(tobias, frank).
vater(tobias, ulrike).
mutter(eva, tobias).
mutter(daniela, frank).
mutter(daniela, ulrike).
```

```
?- frau(X).
X=eva
X=daniela
X=ulrike
no.
```

```
mann(adam).
mann(tobias).
mann(frank).
frau(eva).
frau(daniela).
frau(ulrike).
vater(adam, tobias).
vater(tobias, frank).
vater(tobias, ulrike).
mutter(eva, tobias).
mutter(daniela, frank).
mutter(daniela, ulrike).
```

- ```
  grossvater(X, Y) :-
      vater(X, Z),
      vater(Z, Y).
  ```

- ```
  grossvater(X, Y) :-
      vater(X, Z),
      mutter(Z, Y).
  ```

```
mann(adam).
mann(tobias).
mann(frank).
frau(eva).
frau(daniela).
frau(ulrike).
vater(adam, tobias).
vater(tobias, frank).
vater(tobias, ulrike).
mutter(eva, tobias).
mutter(daniela, frank).
mutter(daniela, ulrike).
```

```
?- grossvater(
     adam,
     ulrike
).

yes.

?- grossvater(
     X,
     frank
).

X=adam
```

```
mann(adam).
mann(tobias).
mann(frank).
frau(eva).
frau(daniela).
frau(ulrike).
vater(adam, tobias).
vater(tobias, frank).
vater(tobias, ulrike).
mutter(eva, tobias).
mutter(daniela, frank).
mutter(daniela, ulrike).
```

```
vorfahr(X, Z) :-
    elternteil(X, Z).
vorfahr(X, Z) :-
    elternteil(X, Y),
    vorfahr(Y, Z).
elternteil(X, Y) :-
    mutter(X, Y);
    vater(X, Y).
```

```
familie(heinz,jutta,[peter,laura]).

familie(karl,gertrud,[]).


?- familie(X, _, []).

X=karl
```

```
weg(Ziel,Ziel,Zustandsliste) :-
    write(Zustandsliste),nl.                  // Ziel erreicht
weg(Start,Ziel,Zustandsliste) :-              // Es gibt einen Weg Start -> Ziel, wenn ...
    operator(Op),                             // ... es einen Operator gibt, ...
    anwendbar(Op,Start),                      // ... der im Startzustand anwendbar ist, ...
    fuehrt_zu(Op,Start,Neu),                  // ... zu einem neuen Zustand führt, ...
    not(member(Neu,Zustandsliste)),           // ... noch nie da war ...
    zulaessig(Neu),                           // ... und zulässig ist, ...
    weg(Neu,Ziel,[Neu|Zustandsliste]).        // ... und es von dort
                                              // einen Weg zum Ziel gibt.
```

```
weg(Ziel, Ziel, Ortsliste):-
    write(Ortsliste), nl.
weg(Start, Ziel, Ortsliste):-
    strasse(Start, Neu),
    not(member(Neu, Ortsliste)),
    weg(Neu, Ziel, [Neu | Ortsliste]).
```

```
weg(Ziel,Ziel,Ortsliste,Strecke) :-
    write(Ortsliste), nl, write(Strecke), nl.
weg(Start, Ziel, Ortsliste, Strecke) :-
    findall(Ort, strasse(Start, Ort), Neuliste),
    bewerte(Neuliste, Start, Strecke, Ziel, BewerteteListe),
    sort(BewerteteListe, SortierteListe),
    member([_, Sgesamt, Neu], SortierteListe),
    not(member(Neu, Ortsliste)),
    weg(Neu, Ziel, [Neu | Ortsliste],  Sgesamt).
```

Spezifisch

Ook! – Design-Kriterien

- Eine Programmiersprache sollte
  schreib- und lesbar für Orang-Utans sein,
- Die Syntax sollte
  - einfach sein
  - leicht zu merken sein
  - Worte wie Monkey oder Affe vermeiden
- Bananen sind gut

Ook! – Syntax-Elemente

- Ook.
- Ook?
- Ook!

• Ook. Ook.	Wert der aktuellen Zelle um 1 erhöhen
• Ook! Ook!	Wert der aktuellen Zelle um 1 verringern
• Ook. Ook?	Zelle nach rechts gehen
• Ook? Ook.	Zelle nach links gehen
• Ook! Ook?	Schleifenanfang – die Schleife durchlaufen, solange Wert der aktuellen Zelle ungleich 0 ist
• Ook? Ook!	Schleifenende – beendet die Schleife, wenn Wert der aktuellen Zelle gleich 0 ist
• Ook! Ook.	Wert der aktuellen Zelle ausdrucken
• Ook. Ook!	Wert von der Tastatur in aktuelle Zelle einlesen

Ook! – „Hello world”

Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook.
Ook! Ook. Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook?
Ook! Ook! Ook? Ook! Ook? Ook. Ook. Ook. Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook. Ook! Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook. Ook! Ook.
Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook.
Ook? Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook! Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook.
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook. Ook! Ook.

Ook! – Implementierung

• Ook. Ook?	++ptr;
• Ook? Ook.	--ptr;
• Ook. Ook.	++*ptr;
• Ook! Ook!	--*ptr;
• Ook! Ook.	putchar(*ptr);
• Ook. Ook!	*ptr = getchar();
• Ook! Ook?	while (*ptr) {
• Ook? Ook!	}

Einfach

$$(5 + 3) * (7 + 2)$$

Forth – Umgekehrt gepolt

$$(5 + 3) * (7 + 2)$$

$$5\ 3\ +\ 7\ 2\ +\ *\ .$$

Forth – Funktionsumfang

- DUP $\ldots\ x \rightarrow \ldots\ x\ x$
- SWAP $\ldots\ x\ y \rightarrow \ldots\ y\ x$
- ROT $\ldots\ x\ y\ z \rightarrow \ldots\ y\ z\ x$
- OVER $\ldots\ x\ y \rightarrow \ldots\ x\ y\ x$
- PICK $\ldots\ x\ y\ z\ 2 \rightarrow \ldots\ x\ y\ z\ x$
- DROP $\ldots\ x\ y \rightarrow \ldots\ x$

Forth – Anwendung

- hupe *einschalten*
- ventil *öffnen*
- 12 *sekunden warten*
- ventil *schließen*
- hupe *ausschalten*

Sicher

```
class NonNull {
    string aString = "Hello";
    string anotherString;
    string? maybeString;
    public NonNull() {
        anotherString = "World";
    }
    public int GetCharCount() {
        return aString.Length + maybeString.Length;
    }
}
```

string? *maybeString*;

… *maybeString*.Length … // Type Error

(*maybeString* != null ? *maybeString*.Length : 0)

((string) *maybeString*).Length

```
int Sqrt(int x)
    requires x >= 0;
    ensures Sqr(result) <= x && x < Sqr(result + 1);
{
    int r = 0;
    while ((r + 1) * (r + 1) <= x)
        invariant r * r <= x;
    {
        r++;
    }
    return r;
}
```

```
class Counter {
    int x;
    public void Inc()
        ensures old(x) < x;
    {
        x = 2 * x; // Error
    }
}
```

```
bool LinearSearch(int[] a, int key)
    ensures result == exists { int i in (0 : a.Length); a[i] == key };
{
    int n = a.Length;
    do
        invariant 0 <= n && n <= a.Length;
        invariant forall { int i in (n: a.Length); a[i] != key };
    {
        // Hier die Liste nacheinander durchsuchen
    }
    return n >= 0;
}
```

```
bool LinearSearch(int[] a, int key) {
    return exists { int x in a; x == key };
}
```

```
public void Swap(int[] a, int i, int j)
    requires 0 <= i && i < a.Length;
    requires 0 <= j && j < a.Length;
    modifies a[i], a[j];
    ensures a[i] == old(a[j]) && a[j] == old(a[i]);
{
    int tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
}
```

Eigenwillig

OO-Dilemma

```
class A {
    virtual public string ToString() {
        return "(" + … + ")";
    }
}

class B : A {
    override public string ToString() {
        return … super.ToString() …;
    }
}
```

```
Person: (#
   name: @text;
   display:< (# do name[]->out.puttext; inner #)
#)

Employee: Person (#
   salary: @integer;
   display::< (# do salary->out.putint #)
#)
```

```
Person: (#
    name: @text;
    display:< (# do name[]->out.puttext; inner #)
#)

Employee: Person (#
    salary: @integer;
    display::< (# do salary->out.putint #)
#)

display: (#
    do name[] -> out.puttext; salary -> out.putint
#)
```

Domänenspezifisch

```
virtual int* FromTo(int b, int e){
    for (int i = b; i <= e; i++) {
        yield return i;
    }
}

foreach(int i in FromTo(0, 10)) {
    Console.WriteLine(i);
}
```

Cω – „Unendliche“ Streams

```
virtual int* From(int n){
   for (;;) {
      yield return n++;
   }
}
```

Cω – Konkatenation

```
From(0).{return 2 * it;}.ToString().{
    Console.WriteLine(it);
};
```

```
string* greeting = {
    yield return "Hello";
    yield return "World!";
};
```

```
decimal GetTotalByZip(XmlDocument doc, int zip) {
    decimal total = 0;
    string xpathQuery = "orders/order[Zip='"+zip+"']/item";
    foreach (XmlElement item in doc.SelectNodes(xpathQuery)) {
        XmlNode price = item.SelectSingleNode("price");
        XmlNode qty = item.SelectSingleNode("quantity");
        decimal p = Decimal.Parse(price.InnerText);
        decimal q = Decimal.Parse(qty.InnerText);
        total += p * q;
    }
    return total;
}
```

```
public decimal GetTotalByZip(Orders orders, int zip) {
    float total = 0;
    foreach (Item item in orders.order[Zip == zip].item) {
        total += item.price * item.quantity;
    }
    order.total = total;
}
```

```
select ProductId, Sum(UnitPrice * Quantity) as Total
    from o in db.Orders
    inner join rd in db.OrderDetails
    on o.OrderID == rd.OrderID
    where o.ShipPostalCode == zip
    group by rd.ProductID
```

Nebenläufig/Parallel

```
async postEvent(EventInfo data) {
    // large method body
}
```

```
public class Buffer {
    public string Get() & public async Put(string s) {
        return s;
    }
}
```

```
Buffer buffer = new Buffer();

buffer.Put("blue");

buffer.Put("sky");

Console.Write(buffer.Get() + buffer.Get());
```

Liefert *bluesky* oder *skyblue*

```
public class Buffer {
    public string Get() & public async Put(string s) {
        return s;
    }
    public string Get() & public async Put(int n) {
        return n.ToString();
    }
}
```

```
public class OneCell {
    public OneCell() {
        empty();
    }
    public void Put(object o) & private async empty() {
        contains(o);
    }
    public object Get() & private async contains(object o) {
        empty();
        return o;
    }
}
```

Und sonst?

Vielen Dank!