Applikative Programmierung in LISP II

- **Bedingte Anweisung**
- **Anonyme Funktionen**
- **Bindung und Umgebung**
- **Variablenarten**
- **Funktionale Argumente**

Bedingte Anweisung

- **Mit COND können bedingte Anweisungen realisiert werden.**
  - (COND (T1 Programm-Stück1)
    (T2 PS-2)
    (Tn PS-n)) mit n>= 1
  - PS-i sind Folgen f1 .. fm von Formen, m>= 0
  - (Ti PS-i) heißen „Klauseln“
- **Auswertung:**
  - Auswertung der Tests in der vorgegebenen Reihenfolge bis ein Test-i - Wert ungleich NIL ergibt. In diesem Fall Auswertung des Rests der Klausel, COND-Auswertung beendet.
  - Der Wert des PS-i ist der Wert der COND-Form.
  - Sind die Werte aller Tests NIL, so ist der Wert der Bedingung NIL.
  - Ist das Programmstück der zutreffenden Klausel leer, so ist der Wert der Klausel der Wert des Tests.
- **Seiteneffekt:**
  - keine durch COND, aber durch Auswertung der T-i und PS-i möglich

Systemfunktionen
Boolesche/bedingte
Ausdrücke

Beispiele / Ausführung

- **Beispiele:**

```
(COND   ((MEMBER  EL LISTE)  LISTE)
        (T  (CONS  EL  LISTE) )
)
(COND   ((NULL  L)  NIL)
        ((EQUAL  EL  (CAR  L))  L)
        (T  (MEMBER  EL (CDR  L)))
 )
```

- **Ausführung:**

```
--->(SETQ L '(AB))
(AB)
--->(COND (( MEMBER (LAST '((AB) (CD)))  L)  'FALSCH)
          (( < (LENGTH (APPEND  '(L)  L)) 4)
                          (REVERSE  '(G I T H C I R))))
(RICHTIG)
```


Funktionen
und
Rekursion

Funktionen

- **Klassifikation**
  - **Systemfunktionen**
    - spezielle Funktionen
    - normale Funktionen
  - **benutzerdefinierte Funktionen**
    - benannte Funktionen (DEFUN)
    - anonyme Funktionen (LAMBDA-Ausdruck)

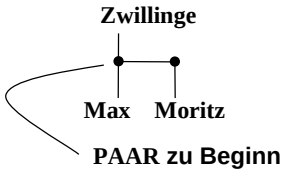
- **Wiederholung Benannte Funktionen**
  - **Definition**
    - `(DEFUN  FName  [Parlist]  [Rumpf] )`
      Auswertung: Wert ist der Funktionsname
  - **Aufruf**
    - `(FName` a_1 a_2 .. a_n `)` a_i Ausdrücke
      Auswertung: a_1, a_2 ... ausgewertet
      Bindung am Formalparameter
      Auswertung des Rumpfs

Funktionen und Rekursion

Funktionen: Beispiele

- **Beispiel:**

```
-->(DEFUN  VERTAUSCHE   (PAAR)
        (LIST (CADR  PAAR) (CAR  PAAR)))
VERTAUSCHE
-->(SETQ  ZWILLINGE  '(MAX   MORITZ))
(MAX  MORITZ)
-->(VERTAUSCHE  ZWILLINGE)
(MORITZ MAX)
```



- **Bemerkungen:**
  - Bindung (dynamisch) nur während Funktionsauswertung, aber s.u.
  - Neue Funktionsdefinition an bereits ex. Namen durch DEFUN möglich
    Systemfunktionen überschreibbar !
  - DEFUN ist spezielle Form: keiner ihrer Argumente wird ausgewertet

Funktionen und Rekursion

Unbenannte Funktionen, Lambda-Ausdrücke

- `(LAMBDA [Parlist] [Rumpf])` **analog zu DEFUN**
  - Definiert namenlose Funktion
  - Lambda-Ausdruck hat keinen Wert
  - `(Lambda-Ausdruck a1 a2 . . .an)` Auswertung wie bei DEFUN

- **Beispiel:**

```
----->(LAMBDA (PAAR)(LIST(CADR  PAAR)(CAR PAAR)))
   ; ist nicht auswertbar, jedoch
----->((LAMBDA (PAAR)(LIST(CADR PAAR)(CAR PAAR)))
            '(FRAU MANN))
(MANN  FRAU)
```

Funktionen und Rekursion

Bemerkungen: DEFUN, LAMBDA

- **DEFUN führt Namen ein**
  Aufruf und Definition beliebig weit auseinander
  ansonsten Ausführung gleich
  LAMBDA-Funktion nur an Stelle der Definition aufrufbar
- **LAMBDA historisch bedingt → Lambda-Kalkul (CHURCH 41)**
  besser AFUN für Anonymous Function
- **Auswertung eines Funktionsaufrufs heißt Lambda-Konversion**
- **Überall, wo Funktionsaufrufe stehen dürfen, dürfen auch Lambda-Ausdrücke stehen**

(Funktionsausdruck. a_1 a_n)

Name einer mit DEFUN definierten Funktion
oder Lambda-Ausdruck

Funktionen und Rekursion

Beispiele für Funktionsdeklarationen

- **Aufnehmen eines Objekts zu einer Liste, falls es noch nicht enthalten ist.**

```
(DEFUN OUR-ADJOIN (OBJEKT LISTE)
  (COND ((MEMBER  OBJEKT  LISTE)  LISTE)
        (T  (CONS  OBJEKT  LISTE))))
```

- **Erweiterung `OUR-ADJOIN`, so dass die Reihenfolge der Argumente beliebig sein darf:**

```
(DEFUN ANY-ORDER-ADJOIN (ARG-1 ARG-2)
  (COND ((LISTP  ARG-1)
         (OUR-ADJOIN  ARG-2  ARG-1))
        (T (OUR-ADJOIN  ARG-1  ARG-2))
))
```

Funktionen und Rekursion

Rekursion: Beispiel

- **Länge einer Liste:**

```
(DEFUN OUR-LENGTH(L)
  (COND ((NULL L) 0)          ; leere Liste
        (T (+1 (OUR-LENGTH (CDR L))))))
```

Laufzeitkeller auf der Listenhalde

- **Anzahl der Atome eines Ausdrucks:**

```
(DEFUN COUNT-ATOMS (L)
  (COND ((NULL L) 0)
        ((ATOM L) 1)
        (T (+ (COUNT-ATOMS (CAR L))
              (COUNT-ATOMS (CDR L))))))
```


Bindung

Parameterübergabe

- **Parameterübergabe bei Funktionsaufruf**

```
------> (SETQ FREI 3)
3

------> (DEFUN DEMO (GEBUNDEN)
          (SETQ GEBUNDEN (+ GEBUNDEN 10))
          (+ GEBUNDEN FREI))
```

- **Ausführung**

```
------> (DEMO  FREI)


------> FREI
```

Call by Value für Parameter-übergabe

Bindung

Bindung und Umgebung

- **Bindung**
  - Zuordnung von Symbolen zu Werten
- **Möglichkeiten**
  - Wertzuweisung mittels SETQ
  - Parameterbindung beim Funktionsaufruf (zeitlich begrenzt)
- **Umgebung**
  - Gesamtheit der zu einem Zeitpunkt zugänglichen Bindungen
  - Aktuelle Umgebung
- **Globale (Top-Level) Umgebung**
  - Umgebung zum Zeitpunkt des Systemstarts
  - Zuordnung zwischen Symbolen und Systemfunktionen
- **Globaler Wert eines Atoms**
  - Wert des Atoms in der globalen Umgebung

Bindung

Umgebung und Variablen

- **Definitionsumgebung einer Funktion**
  - Diejenige Umgebung, die zum Zeitpunkt der Definition aktuell ist.
- **Aufrufumgebung einer Funktion**
  - Diejenige Umgebung, die zum Zeitpunkt des Aufrufs aktuell ist.
- **Gebundene Variable**
  - Variable, die in der Parameterliste einer Funktion auftaucht.
  - Lokale Variable
- **Freie Variable**
  - Variable, die im Rumpf einer Funktion steht, aber keinen Parameter bezeichnet.

Bindung

Beispiel

```
→ (SETQ FREI  3)
3
→ (DEFUN DAMO (GEBUNDEN)
    (SETQ GEBUNDEN (+ GEBUNDEN 10))
    (+ FREI (SETQ FREI GEBUNDEN))) ; Seiteneff.
DAMO
→ (DAMO  FREI)

→ FREI
```


Bindung

Namengleichheit freier und gebundener Variablen

```
→(SETQ FREI 3)
3
→(DEFUN  DEMO1  (FREI)
    (SETQ  FREI (+ FREI 10))
    (+ FREI FREI))

DEMO1
→(DEMO1   2)
24
→FREI
3
```

- **Ablauf**
  - Zuerst Bindung FREI an 2, dann Bindung FREI an 12.
  - Nach Auswertung von DEMO1 gilt die alte Bindung wieder
  - Während der Auswertung von DEMO1 ist die alte Bindung verdeckt

Bindung

Beispiele: Bindung / Umgebung

```
⟶(DEFUN SUCCESSOR (N)
        (+ 1 N))
SUCCESSOR
⟶(SUCCESSOR 17)
18
⟶(SETQ N 10)
10
⟶(DEFUN FUNNY-SUCC (N)
        (SETQ A (+ 1 N))
        (AUX-SUCC))
FUNNY-SUCC
⟶(DEFUN AUX-SUCC ( )
        (+ 1 N))
AUX-SUCC
⟶(FUNNY-SUCC 17)
⟶A
```

Bindung N → 17 (s.u.) gilt nicht während der Auswertung von AUX-SUCC: dort ist N eine freie Variable

Bindung aus Def. Umg. von AUX-SUCC

COMMON-LISP : „statischer" Gültigkeitsbereich (Bindung)

alte LISP-Dialekte: dynamische Bindung

Bindung

LET-Ausdrücke und neue Bindungen

- **LET-Ausdruck**
  - führt neue lokale Variable ein
  - entspricht in etwa einem Block
- `(LET ((Par-1 Ausdr-1) (Par-n Ausdr-n)) ⋮ Rumpf )` **lok. Bindungen, mit denen der folg. Rumpf auszuwerten ist**

  `; Formen` $f_1 \ldots fm,\ m \geq 0$
- **Auswertung**
  - Auswertung der Ausdr-I
  - Bindung an Par-i „parallel"
  - danach Auswertung des Rumpfs
  - Wert des LET-Ausdrucks ist Wert der letzten Form, NIL für m=0
  - anschließend Aufhebung der Bindungen
  - keine Seiteneffekte durch LET, aber durch Auswertung von Ausdr-i bzw. fj

Bindung

LET vs. LAMBDA

- **LET-Ausdruck übersichtlicher als der entsprechende Lambda-Ausdruck**
  Im LISP-System werden LET-Ausdrücke auf LAMBDA-Ausdrücke zurückgeführt
- **((LAMBDA (Par-1 . . . Par-n)**
  Rumpf)
  Ausdr-1 . . . Ausdr-n)

  Bedeutung von LET-Ausdr. als funktionale Argumente (s.u.)
- **----▸(SETQ X 2)**
  2

  ----▸(LET ((X (+ 1 X))
  (Y (+ 2 X)))
  (LIST X Y))
  (3 4)

Bindung

Schachtelung und Gültigkeitsbereich

- **Schachtelung von LET-Ausdrücken bzw. Funktionen und Gültigkeitsbereich**
- ----▸
```
(LET ((X 2)
      (Z 5))
     (LET ((X (+ 1 X))
           (Y (+ 2 X)))
          (LIST X Y Z)))
(3 4 5)
```
- ----▸
```
(DEFUN FU1 (X Z)
  (DEFUN FU2 (X Y)
     (LIST X Y Z))
  (FU2 (+ 1 X)(+ 2 X)))
FU1
----▸ (FU1 2 5)
(3 4 5)
```

Funktionale Argumente

Funktionale Argumente

- **LISP- Funktionen**
  - haben die gleiche Gestalt wie LISP-Daten
  - können wie Daten behandelt und verarbeitet werden
  - insbesondere können Ausdrücke Funktionsbeschreibungen als Wert haben
-
```
(SETQ FN (COND((< X Y) 'ADJOIN)
               (T 'MEMBER)))
```
- **Funktionales Objekt**
  - Ein Ausdruck, dessen Wert eine Funktionsbeschreibung liefert.
  - Auswertung muss immer in einer Argumentposition geschehen
- **Auswertungsformen**
  - FUNCALL
  - APPLY

Funktionale Argumente

Aufruf mit verschiedenen Funktionen

- (FUNCALL Funkt-Obj a_1 . . . a_n)

Ausdruck mit Funktionsbeschreibung als Wert

Argumente für Funktionsaufruf

keine Seiteneffekte durch FUNCALL aber durch Ausw. von fO, a_i möglich

```
→(SETQ  L '(A B C)   X  3   Y  5)
5
→(LET ((FN (COND ((< X Y)'ADJOIN)
                 (T     'MEMBER))))
        (FUNCALL FN 'D  L))
(D A B C)
→(FUNCALL  'ADJOIN  'D  L)
(D A B C)
```

Funktionale Argumente

Auswertung durch APPLY

- **Auswertung funktionaler Objekte durch APPLY**
  - flexibler als FUNCALL
  - Argumente der Funktion einzeln oder zusammen in einer Liste

(APPLY fO a_1 . . . a_n a_{n+1})

Liste

falls n =0 alle Argumente in der Liste
Argument a_1, .. a_n + Elemente der Liste

- **Beispiel:**

```
(APPLY  '+ (1 2 3 4 5 6))  ; Standardfall
21
(APPLY  '+  1 2 3 '(4  5  6))
21
```


Was haben wir gelernt?

- **COMMON-LISP Ausschnitt für wertorientierte/applikative Programmierung**
- **LISP-Programm: Folge von auswertbaren Ausdrücken (Formen) mit oder ohne Seiteneffekt (Zuordnung von Variablen zu Atomen oder Listen)**
- **Interne Listenstruktur als spezielle Haldenstruktur, Auswertung durch rekursiven Abstieg (Compilation!), Gargabe Collection nötig, Listen in leftmost-son-right-sibling-Darstellung**
- **Auswertung oder Quotierung**
- **Listenoperationen: Aufbau mit CONS, Zusammenfügen APPEND, LIST**
  Zerlegen mit CAR, CDR, abgekürzte
  Zusammenfassung wie CADDR
  Spiegelung mit REVERSE
  Zugriff auf Teile mit LAST, MEMBER
- **Strukturprädikate ATOM, CONSP, NULL, LISTP, EQ, EQUAL, LENGTH, <, =, etc. Artvergleiche NUMBERP, SYMPOLP, STRINGP, TYPEP, n-äre Junktoren AND, OR, unäres NOT**
- **Bedingte „Anweisungen“ COND**
- **Funktionen: Definition und Aufruf, unbenannte Funktionen mit LAMBDA-Ausdrücken, Rekursionshandhabung, Parameterübergabe, freie/gebundene Variable, Bindung und Umgebung, LET-Ausdrücke, Schachtelung, Gültigkeitsbereich, funktionale Argumente, funktionale Objekte**

Glossar

- **Atome, Listen, Funktionen, Funktionssymbole, Argumente**
- **Variable als Verweise auf Atome oder Listen, Listenaufbau aus CONS-Zellen**
- **Seiteneffekte als Änderung der Variablenzuordnung mit SETQ, Top-Level-Schleife, Interpretation als Systemfunktion, Compilation äquivalenten Codes wegen Effizienz**
- **Listenfunktionen, Listenprädikate (Unterscheidung nicht sauber), Strukturprädikat für Atome, Ausdrücke, Listen, Bedingungen, Klauseln**
- **DEFUN, LAMBDA, LET, APPLY**
- **Namensgleichheit, freie/gebundene Variable, Bindung, Umgebung, Schachtelung, Gültigkeit**
- **funktionale Argumente, Bindung verschiedener Funktionen**