

Praktische Algorithmen der Bioinformatik und Computerlinguistik mit Lisp

Teil 1: Teaser

6.5.2021

Who is who

- ➔ Prof. Dr. Hans A. Kestler, Institut für Medizinische Systembiologie
- ➔ Dr. Tilman Becker, Deutsches Forschungszentrum für künstliche Intelligenz
- ➔ Dr. Axel Fürstberger, Institut für Medizinische Systembiologie
- ➔ Dr. Johann Kraus, Institut für Medizinische Systembiologie
- ➔ Dr. Ludwig Lausser, Institut für Medizinische Systembiologie
- ➔ Dr. Julian Schwab, Institut für Medizinische Systembiologie
- ➔ M. Sc. Marietta Hamberger, Institut für Medizinische Systembiologie
- ➔ Lisp Alien



lisperati.com

Teaser

JAVA

```
public class Quicksort {

    public static void quicksort(double[] arr) {
        quicksort(arr, 0, arr.length - 1);
    }

    public static void quicksort(double[] arr, int start, int end) {
        if (start < end) {
            int part = partition(arr, start, end);
            quicksort(arr, start, part);
            quicksort(arr, part+1, end);
        }
    }

    public static int partition(double[] arr, int start, int end) {
        double pivot = arr[start];
        int left = start - 1;
        int right = end + 1;
        while (left < right) {
            do {
                left++;
            } while (arr[left] < pivot);
            do {
                right--;
            } while (arr[right] > pivot);
            if (left < right) {
                double tmp = arr[left];
                arr[left] = arr[right];
                arr[right] = tmp;
            }
        }
        return right;
    }

    public static void main(String[] args) {
        double[] arr = {7, 2, 5, 4, 1, 0, 9, 8, 3, 6};
        quicksort(arr);
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
}
```

Clojure

```
(defn quicksort [[x & xs]]
  (let [smaller (filter #(< % x) xs)
        bigger (filter #(>= % x) xs)]
    (when x
      (lazy-cat (quicksort smaller) [x] (quicksort bigger)))))

(defn main []
  (let [arr [7 2 5 4 1 0 9 8 3 6]]
    (println (quicksort arr))))
```

Ziele dieser Vorlesung

“A language that doesn’t affect the way you think about programming is not worth knowing.”

Alan Perlis, 1st recipient of Turing award

Einblick in die funktionale Programmierung

- + Kennenlernen und Überblick
- + konkrete fachliche Inhalte

Programmierparadigmen

Imperatives Programmieren:

- Lineare Abfolge von Befehlen.
- Zustandsorientiertes Programmieren.
- Imperatives Programm beschreibt Wie etwas gemacht wird.

Objektorientiertes Programmieren:

- Daten werden in Klassen gekapselt.
- Ein Programm ist eine Menge interagierender Objekte.

Funktionales Programmieren:

- Programme sind Mengen von Funktionsdefinitionen.
- Zur Laufzeit findet eine Neukombination und Transformation der Funktionen statt.
- Funktionaler Code beschreibt Was gemacht wird.

Thematik der Vorlesung

Aspekte funktionaler Programmiersprachen und ihrer Anwendung

- Funktionales Programmieren
- Lazy Evaluation
- Anonyme Funktionen
- Funktionen höherer Ordnung
- Typsysteme
- Concurrency
- ...

Funktionale Programmiersprachen Lisp

- Haskell
- Clojure
- Scala
- Ocaml
- ...

Bearbeitung anhand konkreter Programmieraufgaben

Programmieraufgaben

Linguistik und DNA

1. Türme von Hanoi
2. Countdown Problem
3. Sortieralgorithmen
4. Dynamic Programming
5. ...

Data Mining

1. Klassifikation
2. Clusterung
3. Simulation boolescher Netze
4. Evolutionäre Algorithmen
5. ...

LISP

```
(defun helloWorld () (print "Hello World!"))
```

- LISP = List Processing
- Vorgestellt 1958 von John McCarthy
- Entwickelt für die Verarbeitung symbolischer Information
- 1994 Einführung des ANSI Standards für Common Lisp
- Konzipiert als funktionale Programmiersprache
- Erweitert um prozedurale und objekt-orientierte Strukturen (CLOS)
- www.sbcl.org
- Editor: [Emacs mit Slime](#)
- Practical Common Lisp (Buch): www.gigamonkeys.com/book

LISP: Basics

Atome:

- **Zahlen:** 123, 0.432, 12/34, ...
- **Zeichenketten:** "Hello World", ...
- **Symbole:** A, B, C, ...

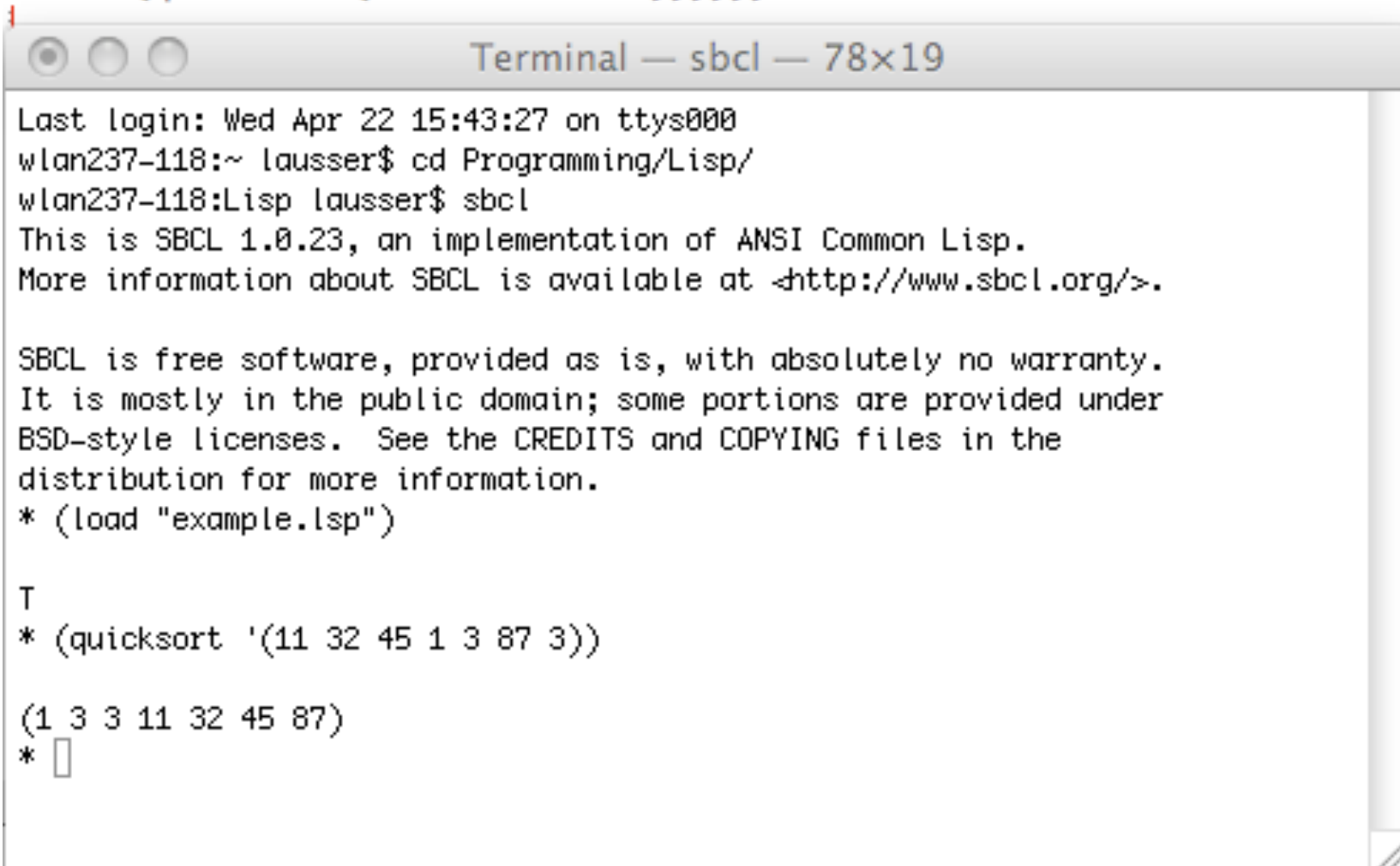
Listen: '(1 2 3), '(1 A "name"), ...

Funktion:

- (+ 1 2) => 3
- (NOT T) => NIL
- (car '(1 2 3)) => 1

LISP: Quicksort

```
(defun quicksort (lis) (if (null lis) nil
  (let* ((x (car lis)) (r (cdr lis)) (fn (lambda (a) (< a x))))
    (append (quicksort (remove-if-not fn r)) (list x)
      (quicksort (remove-if fn r))))))
```

A terminal window titled "Terminal — sbcl — 78x19" showing the execution of the quicksort function. The window has a standard macOS-style title bar with three buttons (red, yellow, green) on the left. The text inside the terminal shows the user navigating to the Programming/Lisp directory, starting SBCL, and then loading and running the quicksort function on a list of numbers. The output shows the sorted list.

```
Terminal — sbcl — 78x19
Last login: Wed Apr 22 15:43:27 on ttys000
wlan237-118:~ lausser$ cd Programming/Lisp/
wlan237-118:Lisp lausser$ sbcl
This is SBCL 1.0.23, an implementation of ANSI Common Lisp.
More information about SBCL is available at <http://www.sbcl.org/>.

SBCL is free software, provided as is, with absolutely no warranty.
It is mostly in the public domain; some portions are provided under
BSD-style licenses.  See the CREDITS and COPYING files in the
distribution for more information.
* (load "example.lisp")

T
* (quicksort '(11 32 45 1 3 87 3))

(1 3 3 11 32 45 87)
* 
```

Haskell

Aus der Werbung:

Haskell is an advanced purely functional programming language. An open source product of more than twenty years of cutting edge research, it allows rapid development of robust, concise, correct software. With strong support for integration with other languages, built-in concurrency and parallelism, debuggers, profilers, rich libraries and an active community, Haskell makes it easier to produce flexible, maintainable high-quality software.

(www.haskell.org)

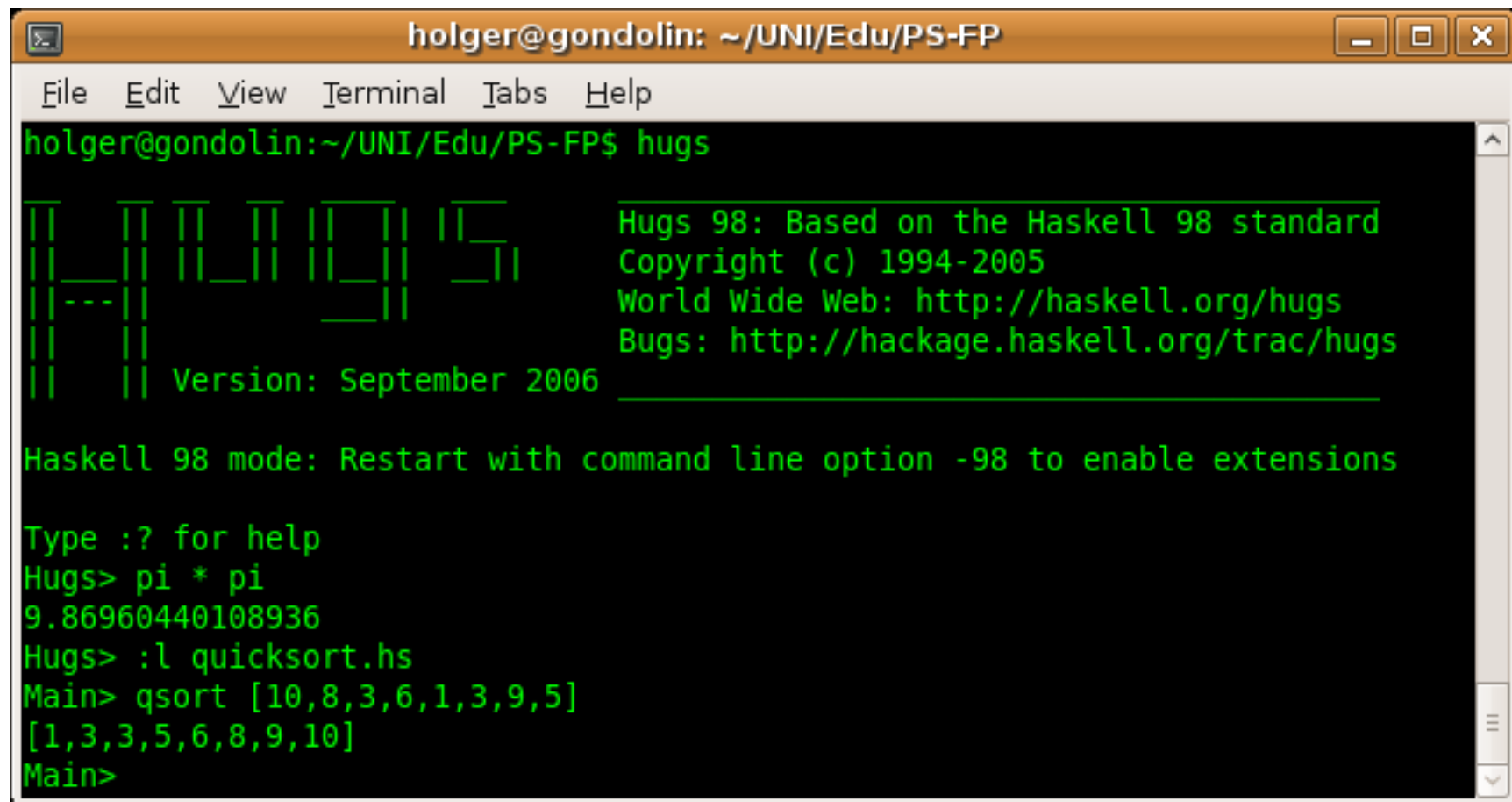
Charakteristika von Haskell:

- **Purely functional:** keine Seiteneffekte
- **Lazy evaluation:** "nicht-strikte" Auswertung – Ausdrücke werden erst dann ausgewertet, wenn das Ergebnis gebraucht wird. Ermöglicht z. B. unendliche Datenstrukturen.
- **Strong typing:** Der Typ von Variablen, Funktionen, Ausdrücken ist statisch festgelegt und wird automatisch inferiert. Fehler können bereits zur Compile-Zeit festgestellt werden.

Haskell

Haskell-Implementierungen:

- HUGS Interpreter
- GHC Compiler, auch als Interpreter GHCi



```
holger@gondolin: ~/UNI/Edu/PS-FP
File Edit View Terminal Tabs Help
holger@gondolin:~/UNI/Edu/PS-FP$ hugs

  ||  ||  ||  ||  ||  ||  ||
  ||__||  ||__||  ||__||  ||
  ||--||      ||__||  ||
  ||  ||      ||
  ||  || Version: September 2006

Hugs 98: Based on the Haskell 98 standard
Copyright (c) 1994-2005
World Wide Web: http://haskell.org/hugs
Bugs: http://hackage.haskell.org/trac/hugs

Haskell 98 mode: Restart with command line option -98 to enable extensions

Type :? for help
Hugs> pi * pi
9.86960440108936
Hugs> :l quicksort.hs
Main> qsort [10,8,3,6,1,3,9,5]
[1,3,3,5,6,8,9,10]
Main>
```

Haskell Quicksort

```
qsort [] = []  
qsort (x:xs) = qsort smaller ++ [x] ++ qsort larger  
  where smaller = filter (<x) xs  
        larger  = filter (>=x) xs
```

Highlights:

- eingebaute Unterstützung von **Listen** als Standard-Datenstrukturen inkl. Operationen (kein explizites Verwalten von Pointern nötig)
- Funktionsdefinitionen durch **pattern matching**
- Funktionen sind "first-class citizens": können Argument oder Ergebnis anderer Funktionen sein – **higher-order functions**
- **Currying** und **partielle Applikation**

Clojure

- Ist ein neuer Lisp Dialekt.
- Wird seit 2007 von Rich Hickey entwickelt.
- Ist eine dynamische funktionale Programmiersprache basierend auf der Java VM.
- Bietet einen einfachen Zugriff auf Java und unterstützt optional Typdeklarationen.
- Hat unveränderliche und persistente Basisdatenstrukturen.
- Bietet lazy evaluation.
- Hat ein software transactional memory system und ermöglicht dadurch einfaches, sicheres und korrektes multi-thread Programmieren.
- Website: <http://clojure.org>

Clojure

Concurrency

```
(def counter (ref 0))  
  
(defn inc-counter []  
  (dosync (alter counter inc)))
```

Java Interaktion

```
(defn sqrt [x]  
  (. java.lang.Math (sqrt x)))
```

Clojure

Quicksort

```
(defn qsort [[x & xs]]  
  (let [smaller (filter #(< % x) xs)  
        bigger (filter #(>= % x) xs)]  
    (when x  
      (lazy-cat (qsort smaller) [x] (qsort bigger)))))
```


Scala

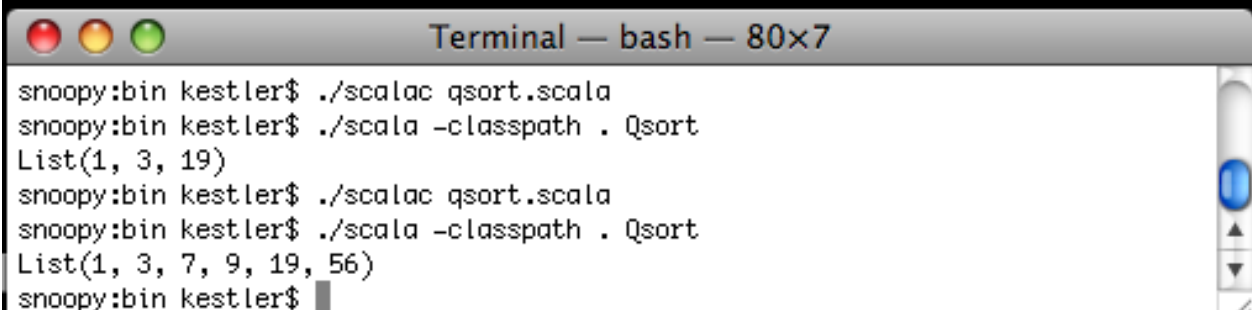
Scalable Language

- Wird seit 2001 von der Gruppe um Martin Odersky an der EPFL entwickelt
- Compiliert zu Bytecode für die JVM
- Vollständige Interoperabilität mit allen Java Bibliotheken
- Fusion objektorientierter und funktionaler Konzepte
- Strong typing
- Website: <http://www.scala-lang.org>
- Alles ist ein Objekt, einschließlich, Zahlen und Funktionen
- Alle Klassen aus `java.lang` werden automatisch importiert, andere müssen explizit importiert werden

Scala

Quicksort in Scala

```
object Qsort {  
  def qsort(lst: List[Int]): List[Int] =  
    lst match {  
      case Nil => Nil  
      case pivot::tail => qsort(tail.filter(_ < pivot)) ::: pivot :: qsort(tail.filter(_ >= pivot))  
    }  
  
  def main(args: Array[String]) {  
    val a=List(3, 19, 1, 56, 7, 9)  
    println(qsort(a))  
  }  
}
```



Terminal — bash — 80x7

```
snoopy:bin kestler$ ./scalac qsort.scala  
snoopy:bin kestler$ ./scala -classpath . Qsort  
List(1, 3, 19)  
snoopy:bin kestler$ ./scalac qsort.scala  
snoopy:bin kestler$ ./scala -classpath . Qsort  
List(1, 3, 7, 9, 19, 56)  
snoopy:bin kestler$
```

Ocaml

Objective Categorically Abstract Machine Language:

- Wird am INRIA entwickelt (erschienen 1996)
- Verwendung von C- und Fortran-Bibliotheken möglich
- Unterstützt funktionale, imperative und objektorientierte Konzepte
- Statisch typisiert
- Website: <http://caml.inria.fr>
- Compiler für Bytecode und Maschinencode (in der Effizienz vergleichbar mit C++ - Code)
- Interaktiver Interpreter zum Testen von Quellcode

```
# print_string "Hello World\n";;  
Hello World  
- : unit = ()
```

Ocaml

Variablen und Funktionen im Toplevel

Objective Caml version 3.10.2

```
# let zahl = 3+3;;  
val zahl : int = 6  
# let quadrat x = x*x;;  
val quadrat : int -> int = <fun>  
# quadrat zahl;;  
- : int = 36  
# quadrat 3.5;;  
Characters 8-11:  
  quadrat 3.5;;  
    ^^^
```

This expression has type float but is here used with type int

```
# let quadrat2 x = x*.x;;  
val quadrat2 : float -> float = <fun>  
# quadrat2 3.5;;  
- : float = 12.25  
#
```

Ocaml

Quicksort in Ocaml

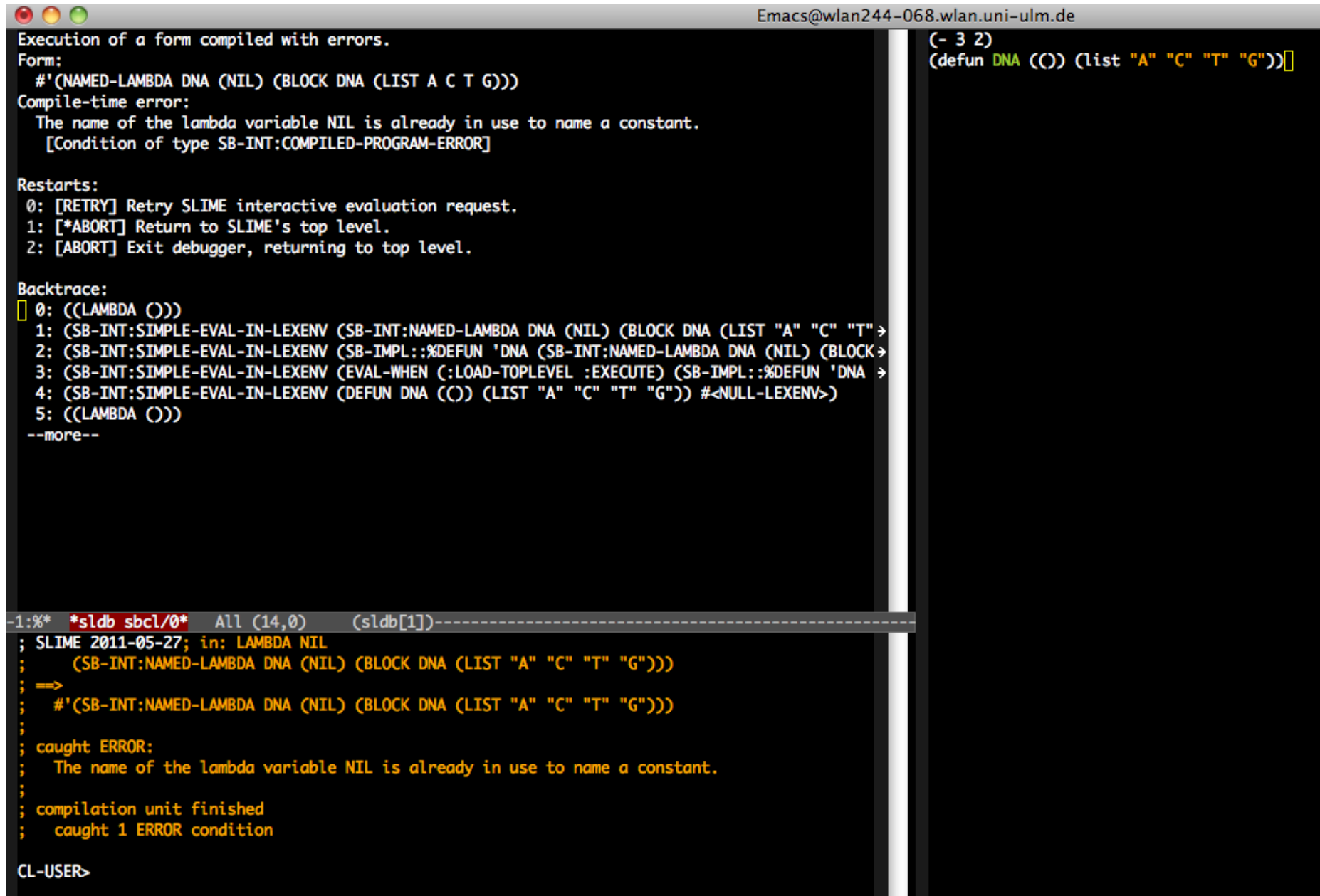
```
let rec quicksort = function
| [] -> []
| test :: rest ->
    let is_less x = x < test in
    let left, right = List.partition is_less rest in
    quicksort left @ [test] @ quicksort right
```

Objective Caml version 3.10.2

```
# #use "ocaml.ml";;
val quicksort : 'a list -> 'a list = <fun>
# quicksort [3;2;55;9;4];;
- : int list = [2; 3; 4; 9; 55]
# □
```

Editoren

Emacs



The screenshot shows an Emacs window with a dark background. The title bar at the top reads "Emacs@wlan244-068.wlan.uni-ulm.de". The main text area displays the following content:

```
Execution of a form compiled with errors.
Form:
  #'(NAMED-LAMBDA DNA (NIL) (BLOCK DNA (LIST A C T G)))
Compile-time error:
  The name of the lambda variable NIL is already in use to name a constant.
  [Condition of type SB-INT:COMPILED-PROGRAM-ERROR]

Restarts:
0: [RETRY] Retry SLIME interactive evaluation request.
1: [*ABORT] Return to SLIME's top level.
2: [ABORT] Exit debugger, returning to top level.

Backtrace:
[] 0: ((LAMBDA ()))
1: (SB-INT:SIMPLE-EVAL-IN-LEXENV (SB-INT:NAMED-LAMBDA DNA (NIL) (BLOCK DNA (LIST "A" "C" "T" "G"))
2: (SB-INT:SIMPLE-EVAL-IN-LEXENV (SB-IMPL::%DEFUN 'DNA (SB-INT:NAMED-LAMBDA DNA (NIL) (BLOCK
3: (SB-INT:SIMPLE-EVAL-IN-LEXENV (EVAL-WHEN (:LOAD-TOPLEVEL :EXECUTE) (SB-IMPL::%DEFUN 'DNA
4: (SB-INT:SIMPLE-EVAL-IN-LEXENV (DEFUN DNA (()) (LIST "A" "C" "T" "G"))) #<NULL-LEXENV>)
5: ((LAMBDA ()))
--more--
```

Below the main text area, there is a status bar with the text: "-1:%* *sldb sbcl/0* All (14,0) (sldb[1])-----".

At the bottom of the window, there is a buffer containing the following text:

```
; SLIME 2011-05-27; in: LAMBDA NIL
; (SB-INT:NAMED-LAMBDA DNA (NIL) (BLOCK DNA (LIST "A" "C" "T" "G")))
; ==>
; #'(SB-INT:NAMED-LAMBDA DNA (NIL) (BLOCK DNA (LIST "A" "C" "T" "G")))
;
; caught ERROR:
; The name of the lambda variable NIL is already in use to name a constant.
;
; compilation unit finished
; caught 1 ERROR condition

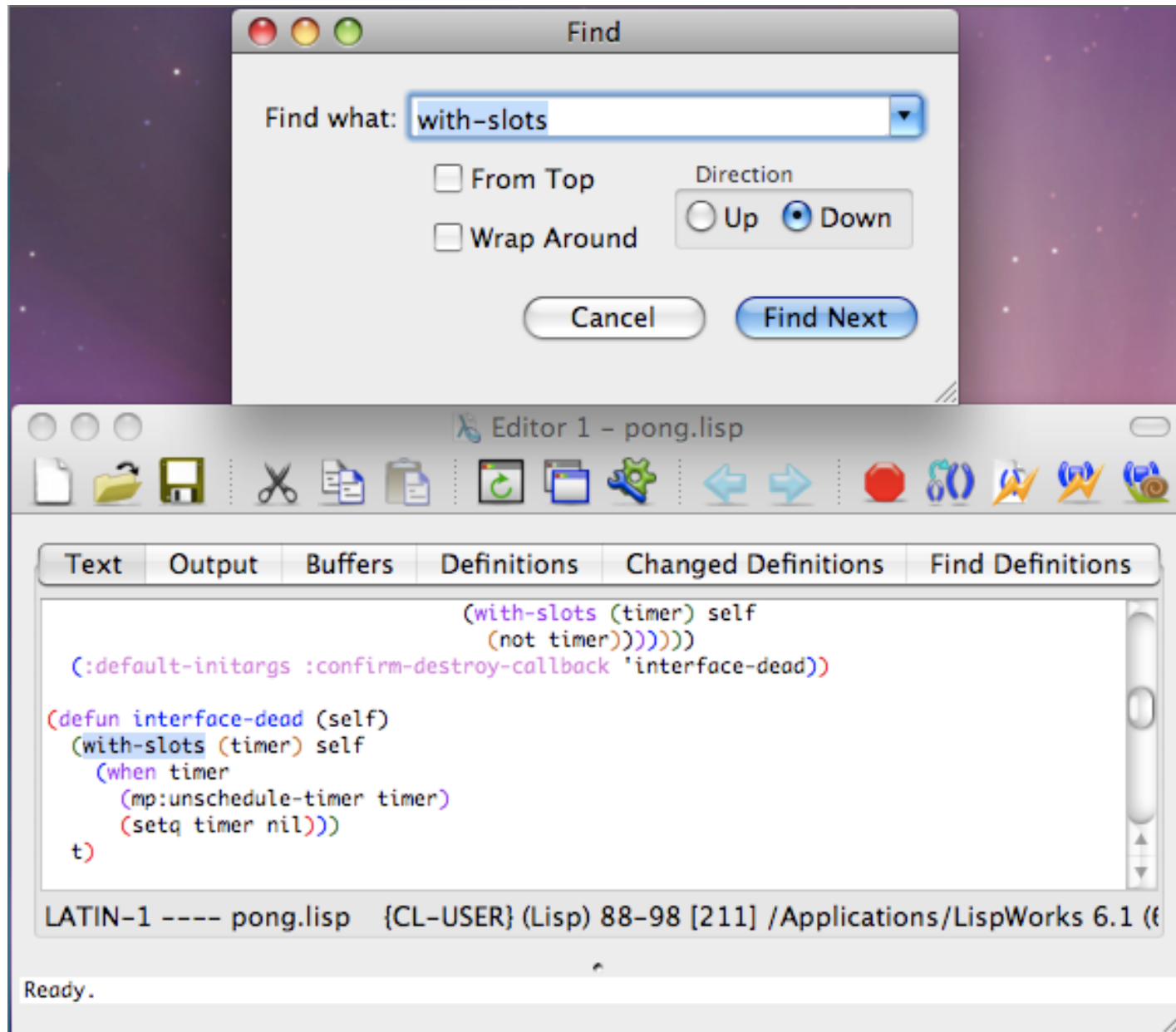
CL-USER>
```

On the right side of the window, there is a small panel showing the current form being evaluated:

```
(- 3 2)
(defun DNA (()) (list "A" "C" "T" "G"))[]
```

Editoren

LispWorks



Editoren

Allegro

The screenshot displays the International Allegro CL 8.2 Release IDE interface. The main window shows the source code for a file named `c:\temp\adder.cl`. The code defines two functions: `add` and `factorial`.

```
* adder movesize projectstyle

(in-package :user)

(defun add (one two)
  (+ one two))

(defun factorial (x)
  (if (= x 1)
      x
      (* x (factorial (1- x)))))
```

Several other windows are open, providing additional information:

- Inspect:** Shows the internal structure of the `(adder 3 :foo) cons` object. The table lists the object's class as `#<built-in-class cons>` and its type as `cons`. The object's slots are `0` (value: `adder`), `1` (value: `3`), and `2` (value: `:foo`).
- Trace - Listener 1:** Displays the trace history for the `adder` function. The history shows a sequence of calls: `1 factorial`, `2 factorial`, `3 factorial`, `4 factorial`, `5 adder (still inside ...)`, `6 adder`, and `7 adder`. The arguments for the `adder` function are `3` and the values returned are `6`.
- Debug Window:** Shows the current state of the debugger. The `Listener 1` tab is active, displaying the error message: `Error: ':foo' is not of the expected type 'number'`. The `Level 1` tab shows the current stack frame, with the `adder` function being executed. The `exp` (expression) is `(adder 3 :foo)` and the `local-0` (local variable) is `(adder 3 :foo)`. The `Debugger` window shows the current state of the debugger, with the `adder` function being executed. The `Debugger` window shows the current state of the debugger, with the `adder` function being executed.
- Subclass Graph for grid-widget:** A diagram showing the relationships between different widget classes. The `grid-widget` class is the base class for `class-grid`, `inspector-grid`, and `patch-grid`. The `class-grid` class is further specialized into `slot-editing-class-grid`.

The bottom status bar indicates the current state of the IDE, showing `MODIFIED` and `cl-user`.

Ablauf der Vorlesung

1. Was ist Lisp?

- Geschichtliches
- Lisp im Vergleich
- Shootouts
- List processing

2. Lisp basics

- Kontrollstrukturen
- Funktionen
- Speichersparendes Programmieren
- Editoren
- Debugging
- Kompilieren

Ablauf der Vorlesung

3. Funktionales Programmieren

- Funktionen als Werte
- Funktionen höherer Ordnung
- Map-, Reduce-Funktionen
- Anonyme Funktionen
- Parameterlisten

4. Imperatives Programmieren

- Variablenbindung mit let
- Dynamische Bindung
- Ein- und Ausgabe
- Destruktive Operationen

Ablauf der Vorlesung

5. Typsystem

- Dynamische Typisierung
- Basistypen
- Arrays
- Hashtabellen

6. Objektorientiertes Programmieren

- Einführung in CLOS
- Generische Funktionen
- Mehrfachvererbung
- Multimethoden

Ablauf der Vorlesung

Sonstiges

- Paketsysteme
- Parallele Programmierung
- Being Lazy
- Tail recursion