Introduction to Symbolic and Statistical NLP in Scheme

Damir Ćavar
dcavar@unizd.hr

ESSLLI 2006, Malaga

July/August 2006

© 2006 by Damir Ćavar

Course Issues

- **Goals:**
  - Introduction to Scheme (DrScheme, MzScheme)
  - Implementation of simple counting algorithms
  - Implementation of parsing algorithms
- **Prerequisites:**
  - Some idea of computation, linguistics . . .

Course Issues

- Web-site with course material:
  - http://web.mac.com/dcavar/ESSLLI2006/
- Practical part:
  - Online coding and discussion during the class session
  - *Repetitorium*: extra-lab session if/when possible with re-implementation, questions, extensions
  - Questions, suggestions, corrections

Agenda

- Introduction to Scheme
- Statistics (counting, N-gram models)
- Parsing (Simple to-down and bottom-up, chart parser)
- Clustering (K-Means, Expectation Maximization)

Introduction to Scheme

- Installing and running Scheme
  - DrScheme IDE
- Using MzScheme
  - interactively
  - scripting

Readings

- Documentation with DrScheme:
  - Teach Yourself Scheme in Fixnum Days (by D. Sitaram)
  - Revised^5 Report on the Algorithmic Language Scheme
- Free online books and tutorials
  - The Scheme Programming Language [Dybvig(2003)]

Starting Scheme

- Command line or IDE
- Command line:

```
Damirs:~ dcavar$ mzscheme
Welcome to MzScheme version 351, Copyright (c) 2004-2006 PLT Scheme Inc.
> 4 + 2
4
> #<primitive:+>
> 2
> (+ 4 2)
6
>
```

Command line

- Exit the interactive scheme interpreter:
  - Unix: `Ctrl-D`
  - Windows: `Ctrl-Z`
  - Commands:

```
> (exit)
```

Interaction

- Hello-world example:

```
> (display "Hello world")
Hello world> (newline)

> (begin
(display "Hello world")
(newline))
Hello world

> (printf "Hello world\n")
Hello world
>
```

Interaction

- `hello1.ss` from within the interactive interpreter:

```
> (load "hello1.ss")
Hello world!
>
```

- via command-line and file:

```
Damirs:~ dcavar$ mzscheme -r hello1.ss
Hello world!
Damirs:~ dcavar$ mzscheme --script hello1.ss
Hello world!
Damirs:~ dcavar$
```

Interaction

- For help on command line parameters:

```
Damirs:~ dcavar$ mzscheme -h
```

Calculating with Scheme

```
> (+ 5 4)
9
> (* 5 3)
15
> (/ 6 2)
3
> (- 7 3)
4
> (* (- 4 2) 5)
10
> (/ 6 4)
1 1/2
> (/ 6.0 4.0)
1.5
```

Arithmetic

- Examples: `boolean.ss`
  - `#t` = true
  - `#f` = false
  - type: `boolean?`
  - negation: `not`

Arithmetic

- Examples: `arithmetic1-4.ss`
  - procedures: `+ - *` \ ...
  - comparisons: `eqv? = > < >= <=`
  - types: `number? complex? real? rational? integer?`

Characters

- Examples: `char.ss`
  - type: `char?`
  - comparison: `char=? char>? char<? char>=? char-ci=?`
  - conversion: `char-downcase char-upcase`

Symbols

- Examples: `symbols1-2.ss`
  - Naming convention: sequences of characters
  - Not self evaluating
  - type: `symbol?`
  - global variable: `(define x 1)`
  - change: `(set! x 2)`

Variables

- Dynamically typed
  - Types do not have to be declared in the program.
  - Types of variables can change during program flow, i. e. integers can become strings or lists and vice versa.
- Garbage collection
  - No allocation and memory handling for variables and their content from the programmers perspective.

Sequences

- Examples: `sequences1-2.ss`
  - Mutable ordered sequences of all data types
  - Strings, Vectors, Dotted pairs, and Lists

Type Conversion

- Example: `types.ss`
  - `char->integer integer->char`
  - `string->list`
  - `number->string string->number`
  - `symbol->string string->symbol`

Procedures

- Example: `procedures.ss`
  - `define lambda`
  - parameters and return values

Hash-tables

- Example: `hash-table.ss`
  - Not ordered storage for key-value pairs (touples)
  - Efficient

Flow Control

- Conditions
- Loops
- Input and Output
- → now with practical example

References

[Dybvig(2003)] R. Kent Dybvig. *The Scheme Programming Language*. The MIT Press, Cambridge, MA, third edition edition, October 2003. ISBN 0-262-54148-3. URL `http://www.scheme.com/tspl3/`.