Why Scheme rocks

Marek Kubica

munich-lisp

April 24, 2009

My Scheme experience

Scheming for fun

- heard about Lisp long ago
- thought ages which Lisp to choose
- decided to start with Scheme (which implementation?)

What I'm doing with Scheme

- Simple games
- Calculating my working hours
- Solving problems in a functional way
- General playthings like Haskell-style currying and useless macros

So don't ask *too* tricky questions ☺

Why Scheme?

Advantages of Scheme

- easy to pick up
- dynamically typed, garbage collected
- free and open development (free as in speech and beer)
- nice for doing first steps in functional programming
- Read Eval Print Loop (honestly, how can one live without?)
- livecoding!

Scheme is dead

SCHEME IS DEAD!

Scheme is dead

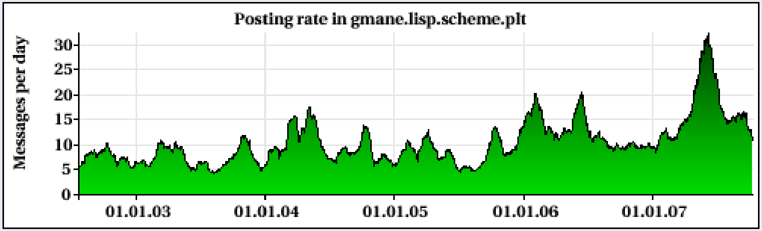
SCHEME IS DEAD!

Not true

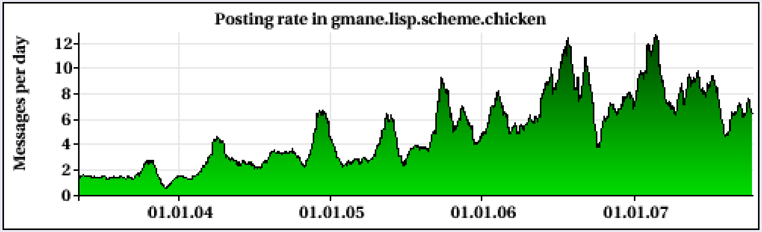
- More and more users (recent interest in functional programming)
- Evolving standards
- Many implementations

Growth

PLT Scheme



Chicken



Growth, part two

Standards

- IEEE Std 1178-1990, somewhere in 1990
- R^{5}RS, 1st August 1998
- R^{6}RS, 27th September 2007
- R^{7}RS, Steering Comitee elected

SRFIs

Scheme Requests for Implementation
`http://srfi.schemers.org/`. A collection of useful libraries that are ported to many implementations.

Growth, part three

Multiple high-quality implementations of Scheme, running on their own, targeting the JVM, CLR; compilers, interpreters

Implementations

1. PLT Scheme
2. Chicken
3. Larceny
4. Guile
5. Ikarus
6. Ypsilon
7. Gambit
8. Chez
9. Bigloo
10. Gauche
11. IronScheme
12. MIT Scheme
13. Mosh Scheme

And these are only the ones with recent releases

Livecoding

What is live coding

Writing software which creates visuals/audio interactively as an performance of art.

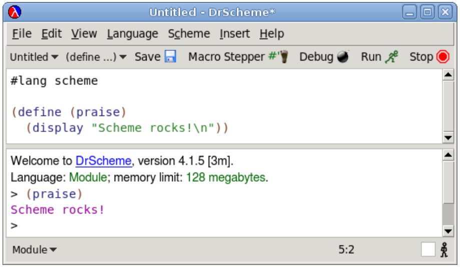
Scheme systems

Due its dynamic nature Scheme is a rather popular language

- Fluxus
- Impromptu

Care to see some videos?

Where to start?


Untitled - DrScheme*
File Edit View Language Scheme Insert Help
Untitled (define ...) Save Macro Stepper Debug Run Stop

```
#lang scheme

(define (praise)
  (display "Scheme rocks!\n"))
```

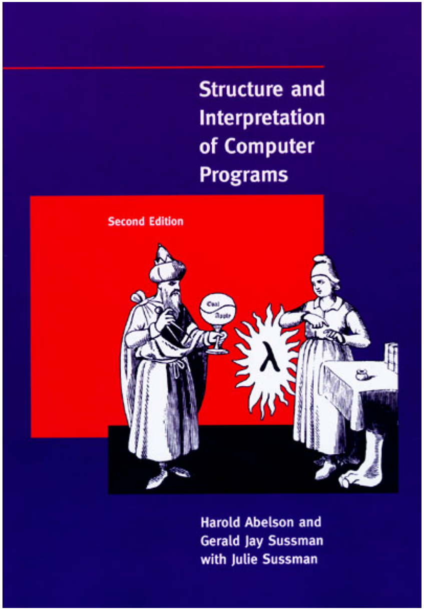
```
Welcome to DrScheme, version 4.1.5 [3m].
Language: Module; memory limit: 128 megabytes.
> (praise)
Scheme rocks!
>
```

And don't forget to pick a book!

DrScheme

- Nice editor for Scheme
- Part of PLT Scheme
- Works out-of-the-box (no configuration)
- useful for beginners
- macro-stepper
- profiling tools

Structure and Interpretation of Computer Programs


Structure and Interpretation of Computer Programs

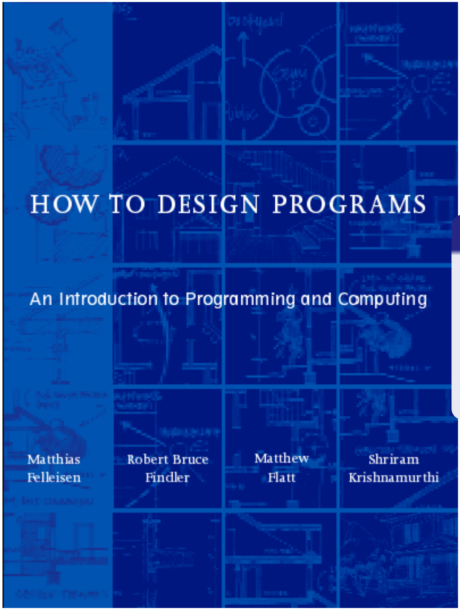
Second Edition

Harold Abelson and Gerald Jay Sussman with Julie Sussman

SICP

- A computer science classic, the *wizard book*
- full text available online from MIT
- lecture videos also available

How to Design Programs



HOW TO DESIGN PROGRAMS

An Introduction to Programming and Computing

Matthias Felleisen

Robert Bruce Findler

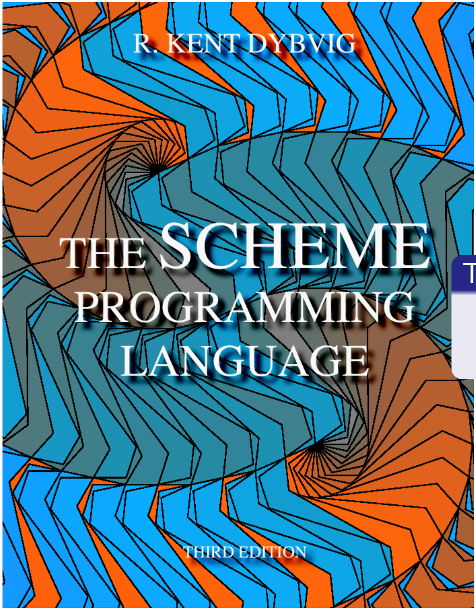
Matthew Flatt

Shriram Krishnamurthi

HtDP

- teaches many programming techniques
- from the creators of PLT
- full text available online

The Scheme Programming Language



TSPL

- describes the language
- full text available online

Wait, there's even more

Some others

- Die Macht der Abstraktion
- Concrete Abstractions
- Simply Scheme
- Teach yourself Scheme in fixnum days

Why PLT?

Advantages of PLT

- Everything-in-one package
- Extensive documentation (master index: 354 pages)
- GUI toolkit, editor, libraries, FFI, 3D support, network access, XML, documentation tools
- continuation based Web server (think Seaside)
- a package installation system, PLaneT
- friendly mailing list

Language experiments

- Typed Scheme: static type system on top of Scheme
- Lazy Scheme: Scheme with lazy evaluation

PLaneT

A central repository for PLT packages

Usage

1. Visit `http://planet.plt-scheme.org/`
2. Choose package
3. Copy-paste installation code into REPL
4. Optional: read documentation

Code

Let's get a flickr interface:

```
(require (planet dvanhorn/flickr:1:0/flickr))
```

downloads, installs and loads the package.

Macros

Code that transforms code

Code is put in, transformed by a macro, executed as regular Scheme code.

- Pattern-based transformations
- not like C macros
- syntax-case vs. syntax-rules
- PLT supports defmacro, too: (require mzlib/defmacro)

Further reading

- Documentation: `http://www.scheme.com/tspl3/syntax.html`
- Scheme vs. CL macros: `http://www.hobbit-hole.org/?p=151`

Macros, example

A postfixed Scheme using pattern-matching macros

```
(define-syntax postfixed
  (syntax-rules ()
    [(_ (operands ... operator))
     (operator (postfixed operands) ...)]
    [(_ atom) atom]))

;; all of these return 5
(postfixed 5)
(postfixed (2 3 +))
(postfixed (2 (1 2 +) +))
(postfixed ((1 1 +) (1 2 +) +))
```

Object-oriented programming

Not the preferred way to use Scheme

Pick one object system

1. Tiny-CLOS
2. Swindle
3. GOOPS
4. STklos
5. Meroon
6. YASOS
7. TinyTalk
8. OakLisp
9. BOS
10. SCOOPS
11. SOS
12. Gauche's
13. Protobj
14. Prometheus
15. ClosureTalk
16. LispMeObjects

Rough overview

`http://community.schemewiki.org/?object-systems`

Functional programming

My preciousss!

Toolbox

- anonymous functions
- first-class functions
- tail-call optimization
- map/filter/fold (in many variants)
- currying
- immutable types

Community

Cares about functional solutions to problems.

Not everything is golden, though

The cons

- Incompatibility
- Lack of libraries
- Divided community (R^{6}RS haters, PLT community, R^{4}RS lovers)
- Extensive but complex documentation
- Virtually unknown
- Many prejudices about Lisp in general
- Few free software projects that are something other than implementations ☺

Finally

Scheme ressources

- `http://schemers.org/` - lists books, documents, implementations, SRFIs, user groups (us too!)
- `http://community.schemewiki.org/` - the Scheme community wiki
- `http://schemecookbook.org/` - recipes for real-world problems
- `http://docs.plt-scheme.org` - PLT documentation
- #scheme on freenode

Thanks for listening!

If you liked the slides, send them to friends, co-workers, to let them know about Lisp in general. I tried to keep them mostly understandable without the audio.