Marketing Functional Programming

Perceptions and Reality

R. Kent Dybvig
Cadence Research Systems
September 2004

Talk Outline

Chez Scheme: where I’m coming from

- background
- priorities
- business model

Perceptions

Other issues

Background

Implements ANSI/R^{5}RS standard Scheme with extensions

Runs on multiple architectures and operating systems

Compiles source incrementally to machine code

Version 1 completed in December 1984 (almost 20 years ago!)

Version 7 should be completed soon

Priorities

Highest priorities:

- reliability
  - reliable compiler and run-time system
  - no arbitrary limitations
  - failure recovery

Priorities

Highest priorities:

- reliability
  - reliable compiler and run-time system
  - no arbitrary limitations
  - failure recovery
- performance
  - compilation speed
  - speed of generated code
  - storage management
  - performance continuity

Priorities

Also important:

- standards compliance
- interoperability with other programs
- features
- debugging tools
- documentation

Business Model

Chez Scheme marketed to corporations, institutions

Petite Chez Scheme freely available

- serves as run-time system for distributed applications
- serves as free implementation for personal use

work constantly to improve reliability

- an ounce of prevention . . .

work constantly to improve performance

work under contract to provide major new functionality

- C, COM interfaces
- ports to new architectures and operating systems
- thread system
- etc.

Why not open source?

Open-source route is always worth considering

Potential open-source benefits:

- larger user base
- contributions from qualified developers
- cost sharing

Potential open-source downsides:

- ineffective below critical mass of users/contributors
- difficult to fund developers

Commercial Model

Sticking with commercial model:

- keeps us honest: more incentive to:
  - fill in functionality
  - maintain reliability and performance
  - eliminate rough edges
- allows us to provide better support services
- keeps us in contact with power users
  - client needs drive development
  - keeps it real

Talk Outline

Chez Scheme: where I’m coming from

Perceptions

- expressiveness of FP languages
- performance of FP languages
- impact of Java
- etc.

Other issues

Expressiveness

Perception:

FP languages are restrictive

Expressiveness

Perception:

FP languages are restrictive

Reality:

this perception is just plain wrong

most FP languages support both imperative and functional programming

most imperative languages support only imperative programming

- no dynamic memory allocation without side effects
- no bounded looping without side effects
- no tail calls without danger of stack overflow

(sadly, most FP languages also fail to guarantee proper tail recursion)

FP versus OOP

Perception:

okay, but surely OOP is better

FP versus OOP

Perception:

okay, but surely OOP is better

Reality:

OOP is nice for encapsulating state

pervasive OOP leads to ultra-imperative programming
(do this to that object, do that to this object, etc.)

most FP languages or implementations support OOP

FP languages thus support mix of paradigms

Generality

Perception:

but FP language *X* is good only for special purpose *P*

Generality

Perception:

but FP language *X* is good only for special purpose *P*

Reality:

most FP languages are general purpose languages

early associations stick, like Scheme and AI

same is true, however, for mainstream languages

- Fortran good only for number crunching
- C good only for systems programming
- Java good only for web applets
- Perl good only scripting

Performance

Perception:

but aren’t FP languages slow?

Performance

Perception:

but aren't FP languages slow?

Reality:

FP *languages* aren't inherently slow

- early *implementations* were slow
- present implementations run the gamut

higher level of abstraction makes optimization

- more difficult
- potentially more fruitful (less overspecification)

potential for big wins greater on larger programs

Performance

Perception:

but interpreted languages *must* be slow

Performance

Perception:

but interpreted languages *must* be slow

Reality:

languages aren't interpreted

- some *implementations* use interpreters
- some use compilers
- some use hybrids

this misperception might survive because

- early implementations were interpreted
- interactivity confused with interpretation

Garbage Collection

Perception:

but all garbage collected languages are slow

Garbage Collection

Perception:

but all garbage collected languages are slow

Reality:

garbage collection often outperforms explicit storage management

- partnership with compiler, run-time system
- support for inline allocation

some implementations not as good as others

performance concerns outweighed by benefits:

- no dangling pointers
- fewer memory leaks
- increased reliability, productivity

analogies: O/S scheduling, assembly versus high-level language

Conservative Collection

Perception:

so GC is good, but even C can be GC'd

Conservative Collection

Perception:

so GC is good, but even C can be GC'd

Reality:

this is true, after a fashion, with conservative collectors

conservative collectors still susceptible to

- memory leaks
- dangling pointers

conservative collectors don't enjoy same performance benefits

analogies: "lite" cigarettes, low-carb big-macs

Hardware Support

Perception:

FP languages don’t mesh well with stock hardware

Hardware Support

Perception:

FP languages don’t mesh well with stock hardware

Reality:

FP languages could use better support for:

- generic and arbitrary-precision arithmetic
- bounds checking
- tag checking (latently typed languages)

stock HW designed to support unsafe imperative languages

we adapt with clever implementation techniques

bigger concern may be virtual machines like JVM and .NET

Libraries

Perception:

FP languages lack libraries

Libraries

Perception:

FP languages lack libraries

Reality:

this has been a real problem, perhaps the major problem

strides being made in Scheme community, elsewhere

Interoperability

Perception:

FP languages don’t play well with others

Interoperability

Perception:

FP languages don’t play well with others

Reality:

paradigm and datatype mismatches do exist

many FP implementations support C interfaces

some interface with Java

onus always on the FP implementation
(how many C implementations support FP interfaces?)

ever tried to mix Haskell and Scheme?

Java

Perception:

Java will choke off demand for FP languages

Java

Perception:

> Java will choke off demand for FP languages

Reality:

> initially, this was probably true
>
> Java has, however, helped validate garbage collection (after a rocky start)
>
> may also shake up management conservatism

Purity

Perception:

FP is more about purity than usability

Purity

Perception:

FP is more about purity than usability

Reality:

FP language designers and implementors *are* purists

great pressure to “get things right”

- language design
- implementation reliability
- detecting errors

takes time and energy away from eye candy

“right” doesn’t sell as well as eye candy

Talk Outline

Chez Scheme: where I’m coming from

Perceptions

Other issues

- backing
- stability
- etc.

Institutional Backing

Big push from an 800-lb Gorilla would help

- examples: C/AT&T, Java/Sun
- not necessary: Perl does okay
- not sufficient: Ada

Software Patents

Software patents present a signficant problem

- software patents handed out for “obvious” solutions
- large companies churn out patents like mad . . .
- . . . then lay in wait for profitable opportunities
- small developers cannot afford patent process (application or defense)

Size and Stability

Big companies want to deal with other big companies

- especially true for single-source technology
- big companies shy away from FP languages
- pressure suppliers to do the same

Capitalization

Most FP implementations are undercapitalized

- this is where 800-lb gorilla could help
- typical investment problem
  - need for money to fund development and marketing
  - need development and marketing to increase demand
  - need demand to justify money
- huge time and cost to develop competitive implementation
- payoff, if any, may take many years

Inertia

People just don’t want to try new things

- some people still program in assembly language
- some still insist that Fortran IV is all they need
- reluctance part religion, part fear
- educational institutions partly to blame
- many who take the leap get pulled back

Summary

Inaccurate perceptions:

- FP is restrictive, special-purpose, and inefficient
- overcome with education, better marketing

Accurate perceptions:

- FP needs more libraries, better interoperability
- overcome with time and hard work

Underlying problems:

- lack of funding, inertia
- not clear how to overcome