The Mapping Dilemma

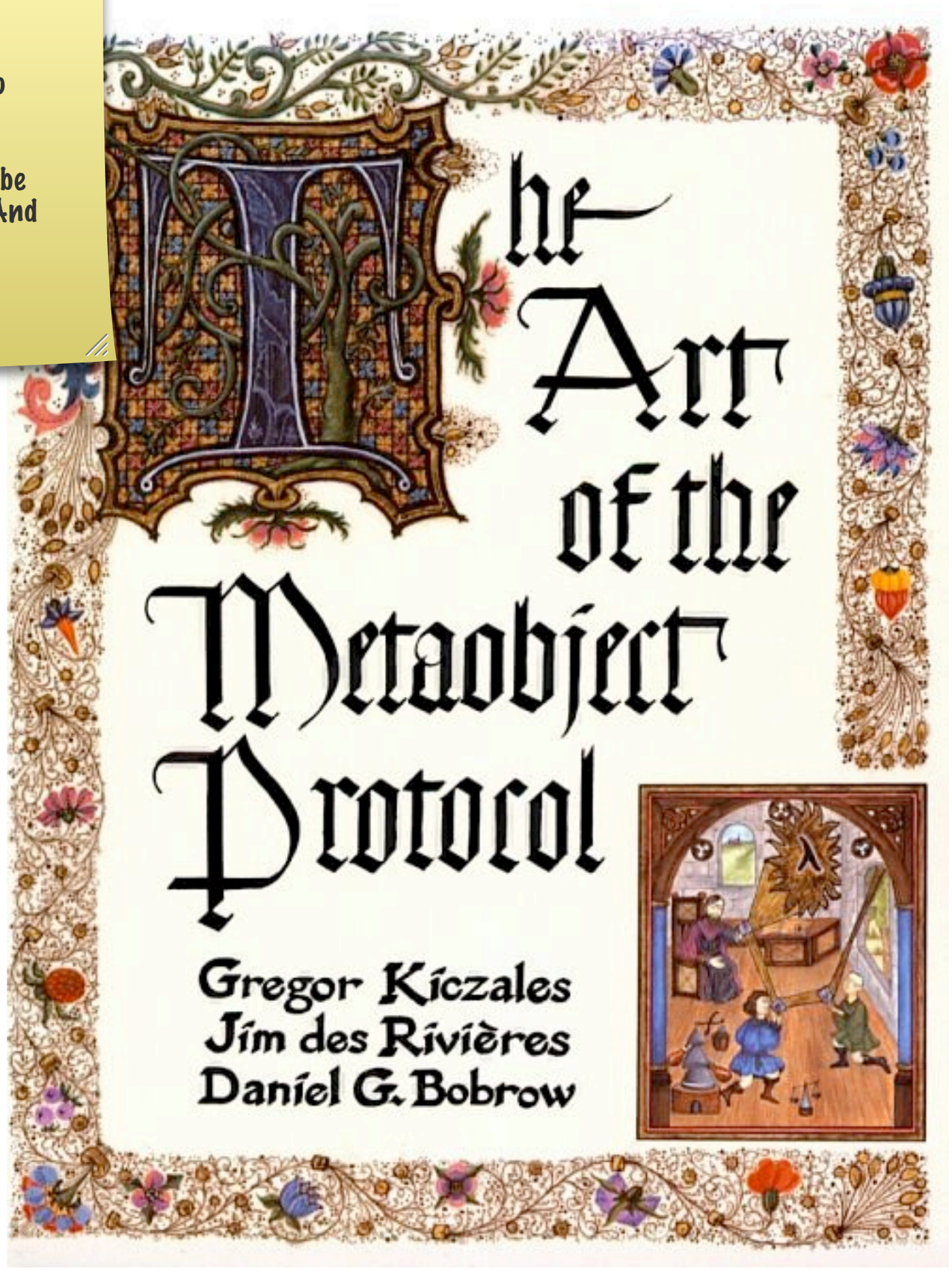
Or the ramblings of a JavaScript hacker

“Our plan and our hope was that the next generation of kids would come along and do something better than Smalltalk around 1984 or so. We all thought that the next level of programming languages would be much more strategic and even policy-oriented and would have much more knowledge about what it was trying to do. But a variety of different things conspired together, and that next generation actually didn’t show up.”

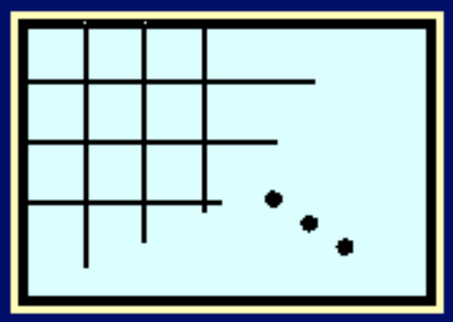
best book on object oriented programming in 10 years

Alan Kay complained about it's Lisp centric nature

I'll try not be so Lisp centric. We'll be talking about how I leverage Lisp. And perhaps you'll see the advantages.



It's Simple



cells in which to click,
display text etc.

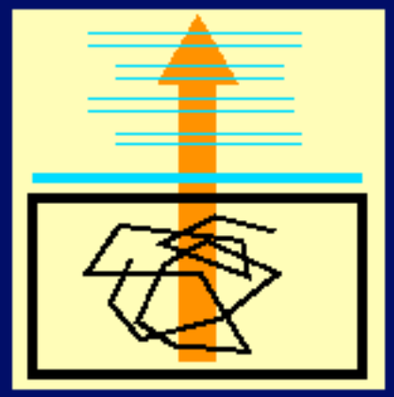
```
i: 1 → 100
j: 1 → 100
make-text-window (~~~)
```

this looks fine...

CLOS in 10 pages
fast CLOS in 350 pages

hematomas of
duplication

But. . . Performance?



the interface itself doesn't
betray the implementation . . .

But, performance
shows through

in order to control complexity, so we hide the implementation

we have to let clients control mapping decisions. but their brain is not getting any bigger.

Hiding vs Dividing

must control complexity → hide the implementation → black box abstraction → mapping decisions show through

→ must let clients control mapping decisions

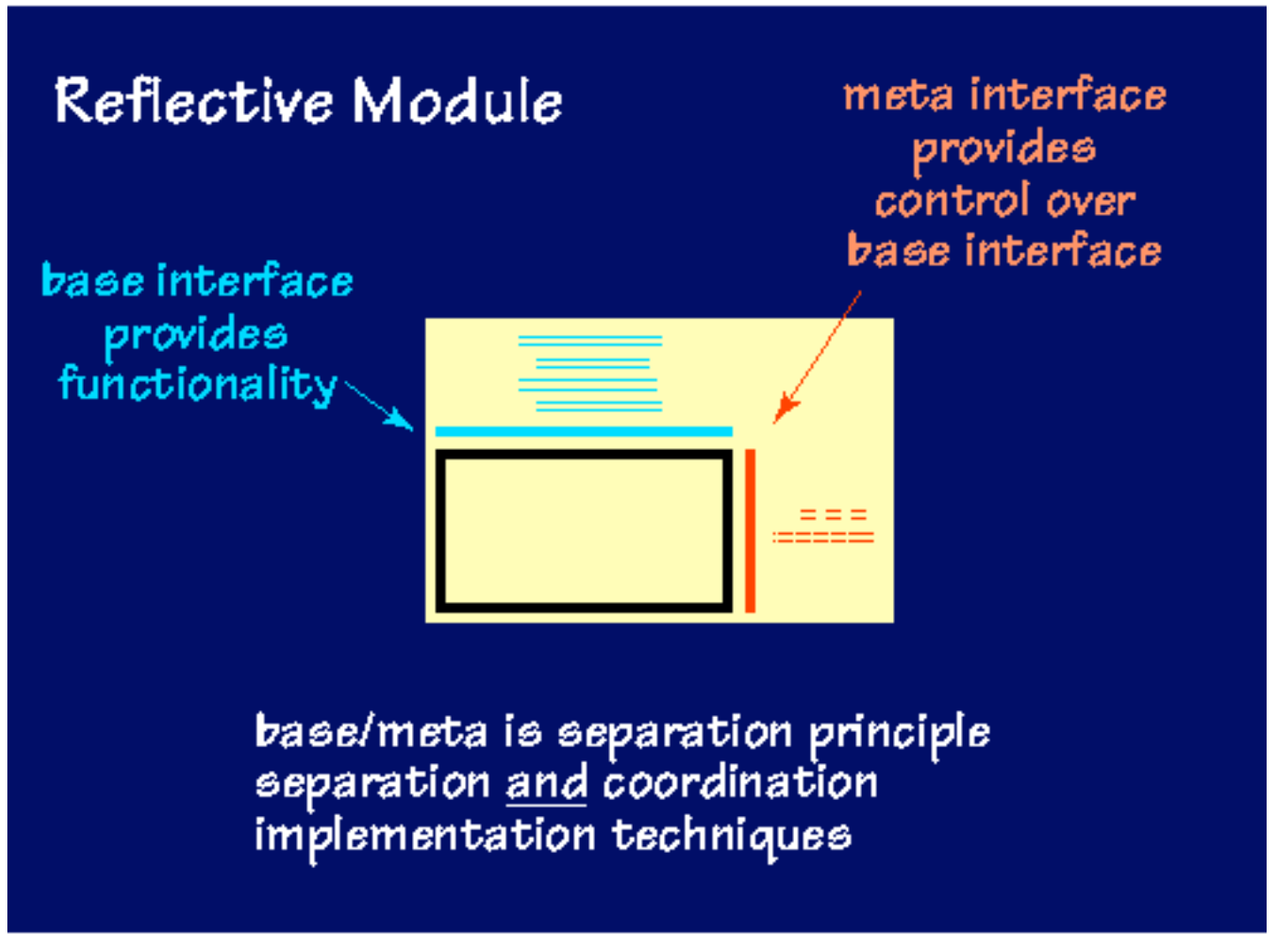
→ "If you can't conquer at least divide"

→ divide what from what?

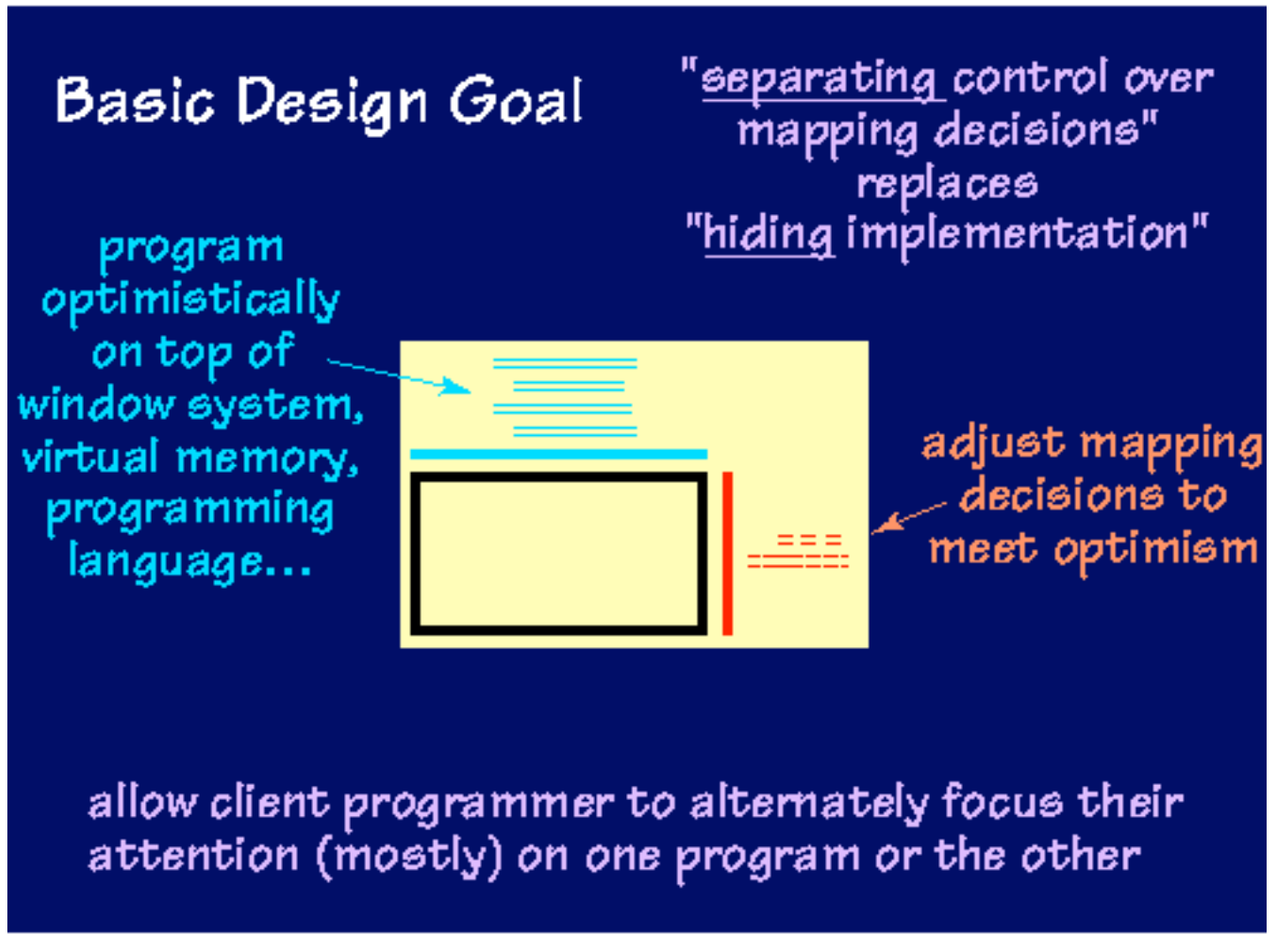
base and meta are not the same, not by a long shot

I don't think meta programming is super common but I believe principled meta protocols would allow us to develop better software

this is how CLOS solved it.



to build robust software regular and meta programming should not be separate steps.



When working in real life situations, users of high-level programming languages encounter a number of common problems. Often, these can be expressed in the form of desires: for compatibility with other related languages; for customizations to support specific applications; for adjustments to the implementation in order to improve the performance of a given application program.

Lacking mechanisms to meet these goals, programmers are driven to employ workarounds, to give up on their desires, or to abandon the language completely.

Object Oriented Programming

Functional Programming

Neither presents solutions to the problem

Nor test driven development

Nor static types

Nor MVC

Nor IDEs

Nor agile

time constraints, complexity of implementation

so we pick the 70-80% solution because we needed the solution yesterday

even though we know we'll pay for it

Worse is Better

no continuum

not judging whether they are good or bad but someone has decided ahead of time what kind of problems you have and how you should solve them, they've made some mapping decisions

you have to pick one or the other, why can't we scale between both use cases from the same framework?

not all their fault of course, jQuery is constrained by JS and UIKit ObjC

jQuery vs. UIKit

What if Worse is not Better wasn't so costly?

What if the 90% or 95% solution is reachable in a reasonable amount of time? And the 99% is in the realm of possibility

Let's design something from the viewpoint of worse is not better

Worse is not Better

I don't have any answers, but I had a bit of a revelation recently ...

A promenade

subsumes generic methods and pattern dispatch

great paper by Craig Chambers (Self)

but the really good literature on how to make this stuff efficient - ML pattern match compilation literature

predicate dispatch

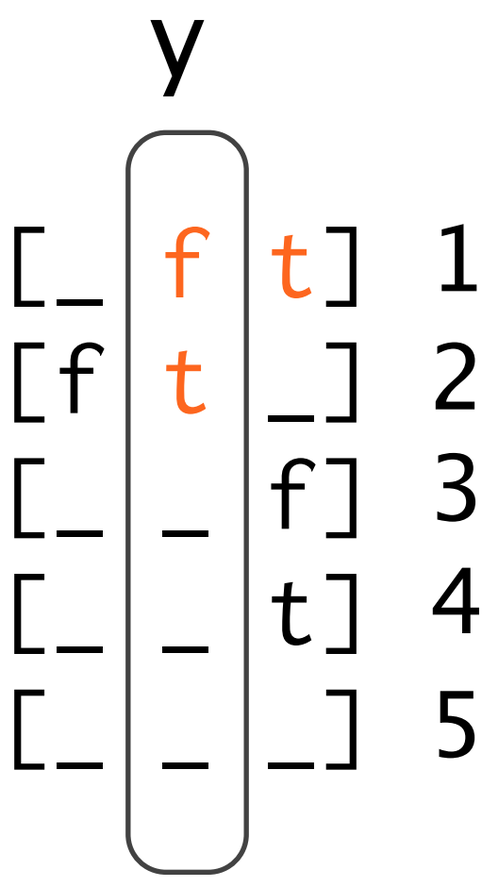
Lets build an optimizing pattern match compiler

Necessity

```
(match [x y z]
  [_ false true] 1
  [false true _] 2
  [_ _ false] 3
  [_ _ true] 4
  :else 5)
```

```
x y z
[_ f t] 1
[f t _] 2
[_ _ f] 3
[_ _ t] 4
[_ _ _] 5
```

```
x y z
[_ f t] 1
[f t _] 2
[_ _ f] 3
[_ _ t] 4
[_ _ _] 5
```


y
[_ f t] 1
[f t _] 2
[_ _ f] 3
[_ _ t] 4
[_ _ _] 5

```
 y x z
[f _ t] 1
[t f _] 2
[_ _ f] 3
[_ _ t] 4
[_ _ _] 5
```

```
(cond
 (= y false) (cond
              (= z false) (let [] 3)
              (= z true) (let [] 1)
              :else 5)
 (= y true) (cond
             (= x false) (let [] 2)
             :else (cond
                    (= z false) 3
                    (= z true) 4
                    :else 5))
 :else (cond
        (= z false) (let [] 3)
        (= z true) (let [] 4)
        :else 5))
```

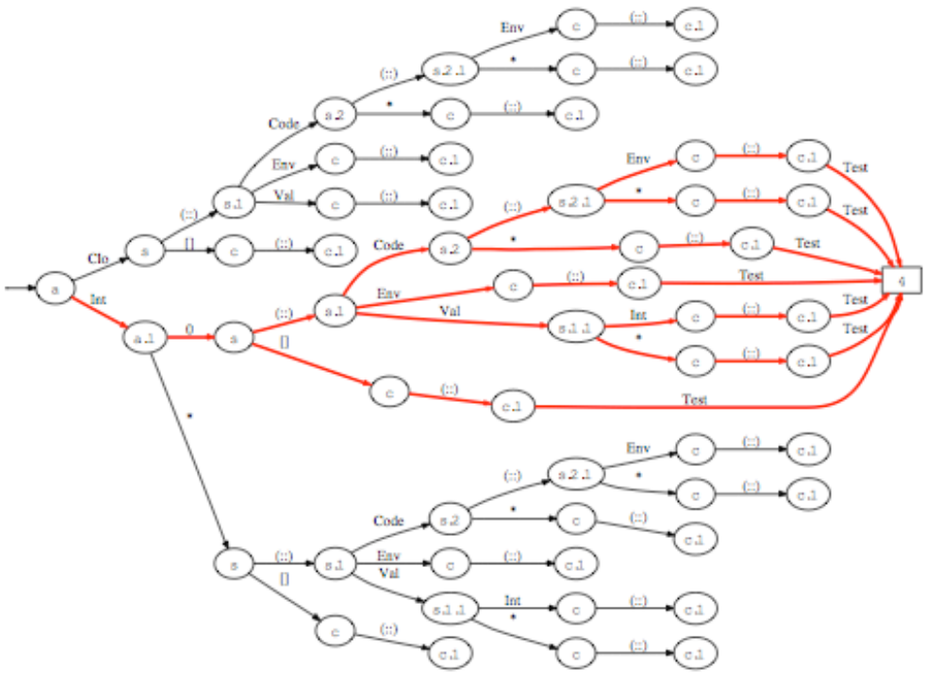


Figure 6. Naive decision tree for example 3

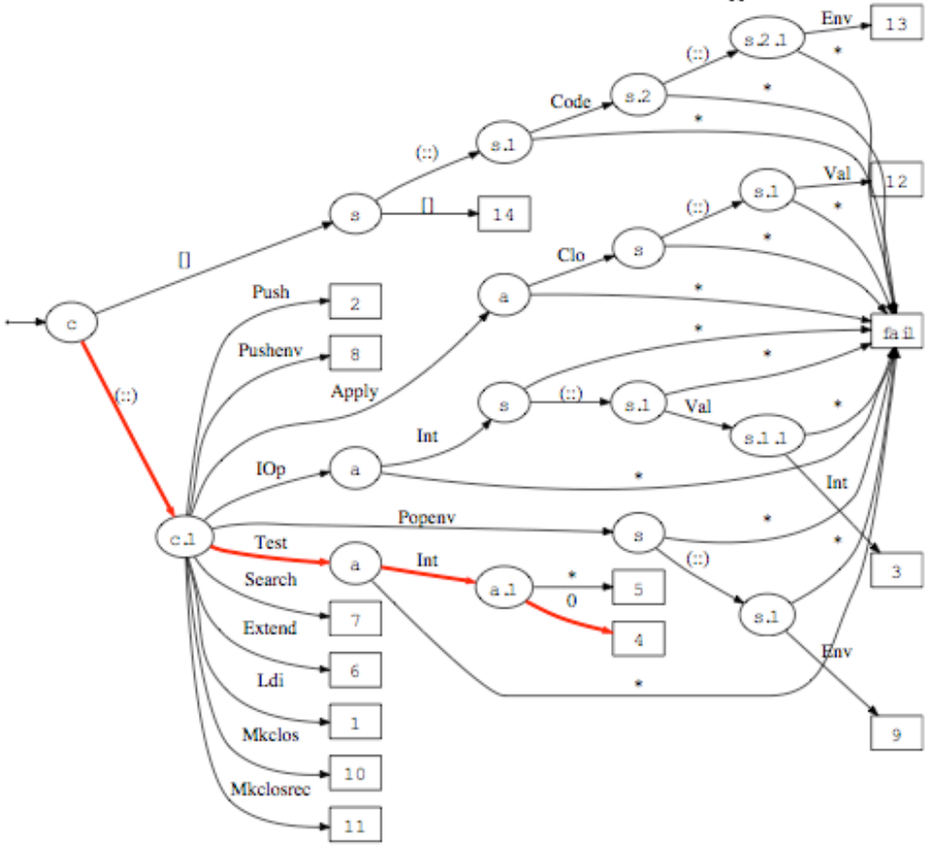
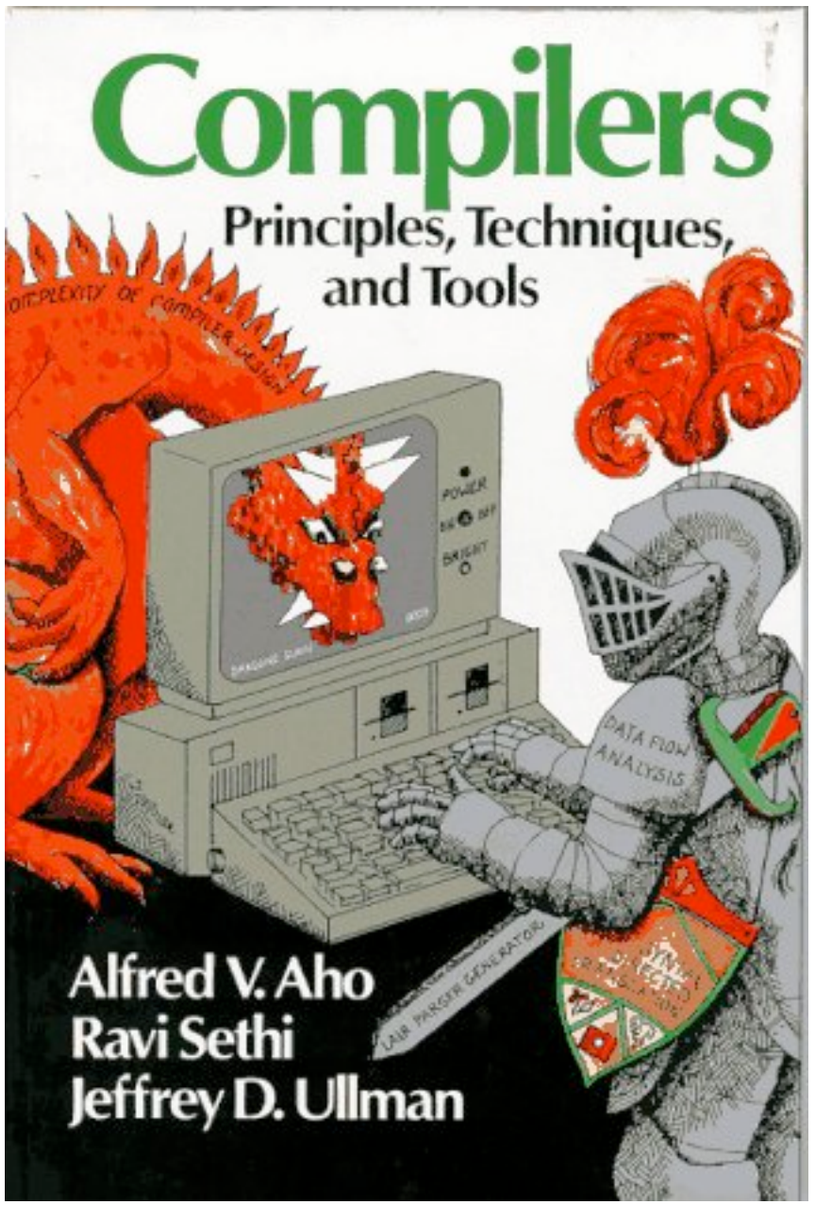


Figure 7. Minimal decision tree for example 3

Implementation


Compilers
Principles, Techniques,
and Tools
COMPLEXITY OF COMPILER DESIGN
POWER
BRIGHT
DATA FLOW
ANALYSIS
LALR PARSER GENERATOR
Alfred V. Aho
Ravi Sethi
Jeffrey D. Ullman

No lexing

No parsing

Almost all of our code is just implementing the algorithm in the paper - no distractions

in fact most of the extra code is just ensuring that the whole process is extensible

Radical simplicity

Features?

- seq patterns

- seq patterns
- map patterns

- seq patterns
- map patterns
- or patterns

- seq patterns
- map patterns
- or patterns
- guards

- seq patterns
- map patterns
- or patterns
- guards
- as

- seq patterns
- map patterns
- or patterns
- guards
- as
- locals matching

- Fully extensible

- Fully extensible
- Pattern match Java objects

- Fully extensible
- Pattern match Java objects
- Example – pattern match with regexes, how much code to implement?

looks pretty darn object oriented to me

```
        tch.regex
      (:use [match.core :only [emit-pattern to-source pattern-equals
                               pattern-compare]]))

;; define our type
(defrecord RegexPattern [regex])

;; pattern equality and sorting
(defmethod pattern-compare [RegexPattern RegexPattern]
  [a b] (if (and (= (.pattern (:regex a)) (.pattern (:regex b)))
                 (= (.flags (:regex a)) (.flags (:regex b))))
          0 -1))

;; emit our type when we parse the match expression
(defmethod emit-pattern java.util.regex.Pattern
  [regex]
  (RegexPattern. regex))

;; what does our pattern test?
(defmethod to-source RegexPattern
  [pat ocr]
  `(re-matches ~(:regex pat) ~ocr))
```

```
(let [s "hello, world!"]
  (match [s]
[#"hello.+"] 0
[#"goodbye.+"] 1
:else 2))
```

```
(if (re-matches #"hello.+" s)
  0
  (if (re-matches #"goodbye.+" s)
  1
  (if :else
    2
    nil)))
```

is a byte the most important data structure of all?

vector patterns

is a byte the most important data structure of all?

vector patterns

- originally only to match persistent vectors

is a byte the most important data structure of all?

vector patterns

- originally only to match persistent vectors
- we can fully leverage Maranget's work since vectors support random access

is a byte the most important data structure of all?

vector patterns

- originally only to match persistent vectors
- we can fully leverage Maranget's work since vectors support random access
- but wait, why can't we apply this to lower level data structures? primitive arrays? bytes?

for map pattern matching we have protocol that you can implement or extend a type too.

But this is too heavyweight or not applicable for primitive arrays and bytes

Cost of Abstraction

everything I've shown so far would be a pain to implement w/o macros. you could do it of course, but the performance would be nothing to write

everything I've shown so far would be a pain to implement w/o macros. you could do it of course, but the performance would be nothing to write

- runtime abstractions are too costly

everything I've shown so far would be a pain to implement w/o macros. you could do it of course, but the performance would be nothing to write

- runtime abstractions are too costly
- impossible to guess what is desired

everything I've shown so far would be a pain to implement w/o macros. you could do it of course, but the performance would be nothing to write

- runtime abstractions are too costly
- impossible to guess what is desired
  - "sufficiently smart compiler" ... snort

everything I've shown so far would be a pain to implement w/o macros. you could do it of course, but the performance would be nothing to write

- runtime abstractions are too costly
- impossible to guess what is desired
- "sufficiently smart compiler" ... snort
- we can't solve your problem - lets give you an easy interface to solve it yourself

- You can match primitive arrays

- You can match primitive arrays
- You can match bits in a byte

- You can match primitive arrays
- You can match bits in a byte
- Zero overhead

- You can match primitive arrays
- You can match bits in a byte
- Zero overhead
- JVM can optimize your match expression in incredible ways

Cost of Worse is not Better?

1 month of work

in my spare time

with spare time help from

1 other dev

~1200 LOC

Experience Report

Experience Report

- Probably took as long to understand the paper as to implement

Experience Report

- Probably took as long to understand the paper as to implement
- Match primitive arrays faster than Scala or Racket

Experience Report

- Probably took as long to understand the paper as to implement
- Match primitive arrays faster than Scala or Racket
- and probably everyone else ;)

Experience Report

- Probably took as long to understand the paper as to implement
- Match primitive arrays faster than Scala or Racket
- and probably everyone else ;)
- Many more optimizations still possible (shrink code size with backtracking, constant time dispatch on literals, etc.)

A **Big** Mapping Dilemma

Jeremy Ashkenas and I were having a conversation about MVC frameworks and I've looked at a lot of them and I think they're terrible.

They're trying to bring traditional UI development to browser without deep consideration for the mapping dilemmas around the asynchrony the DOM and event handling

Client Side Web Programming

ClojureScript

Solving the mapping dilemma

Solving the mapping dilemma

- How well do our languages let us delay critical decisions?

Solving the mapping dilemma

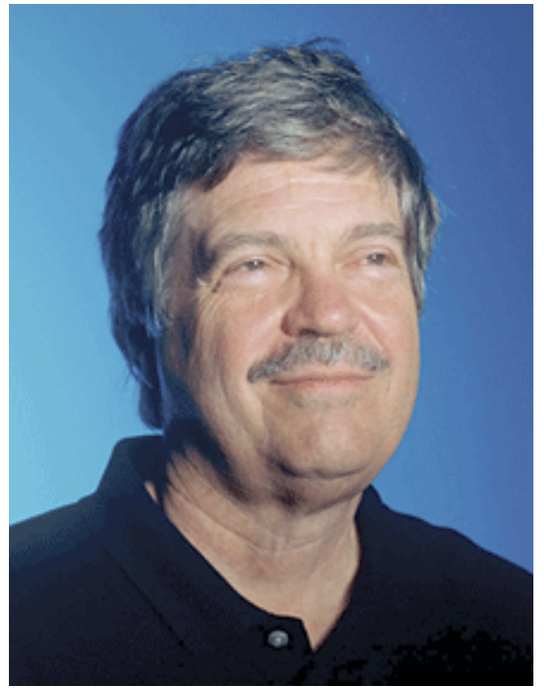
- How well do our languages let us delay critical decisions?
- How well do our languages let us pick strategies - including code generation.

Solving the mapping dilemma

- How well do our languages let us delay critical decisions?
- How well do our languages let us pick strategies – including code generation.
- How few assumptions do our tools have about the problems we want to solve?

Solving the mapping dilemma

- How well do our languages let us delay critical decisions?
- How well do our languages let us pick strategies – including code generation.
- How few assumptions do our tools have about the problems we want to solve?
- How much of Computer Science is at our finger tips when we sit down to work?




Peter
Van Roy

Final thoughts

- Throw traditional Object Orientation out the door, get rid of all of your assumptions

- Throw traditional Object Orientation out the door, get rid of all of your assumptions
- Be humbled, it’s worth it

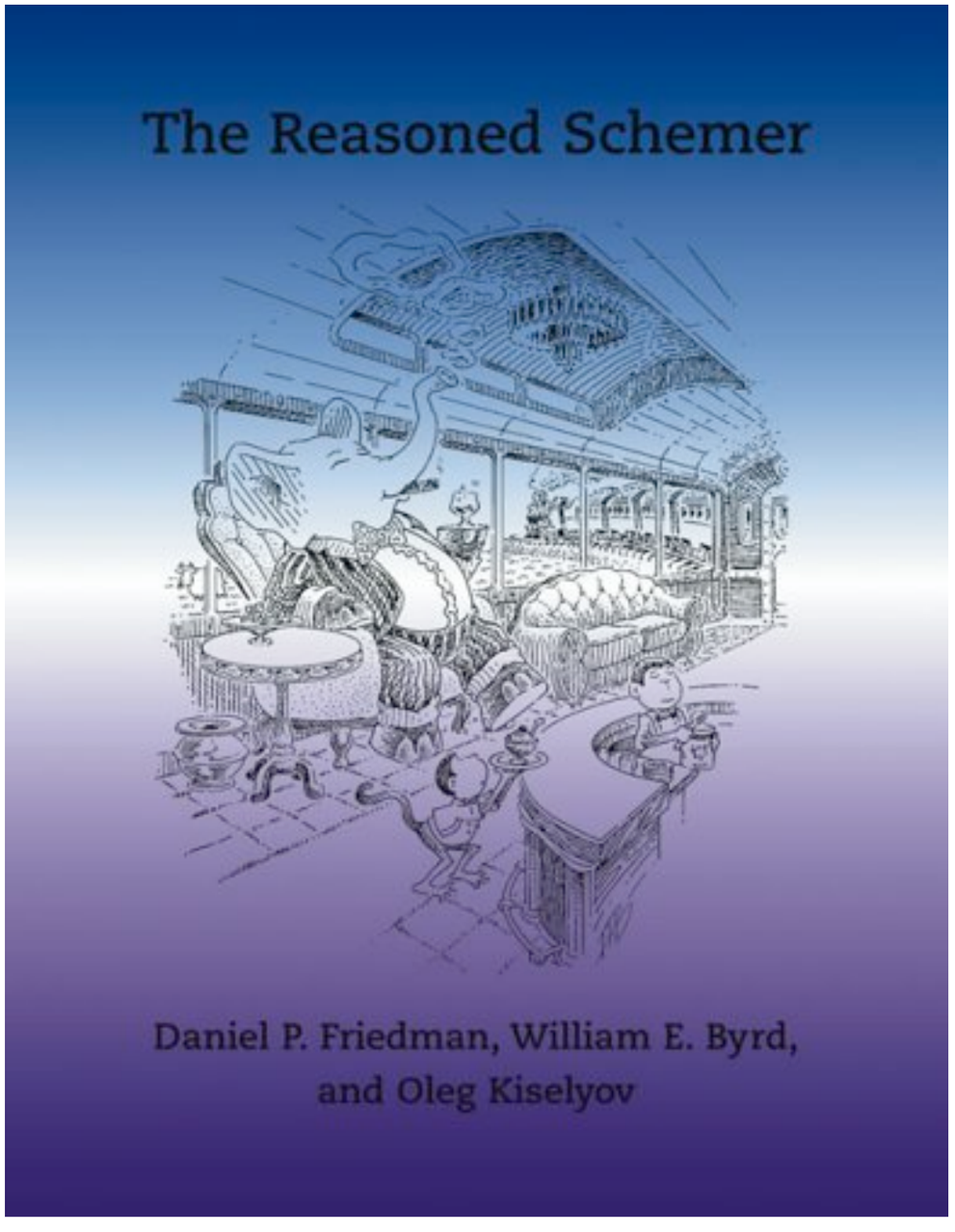
- Throw traditional Object Orientation out the door, get rid of all of your assumptions
- Be humbled, it's worth it
- Object Oriented The Good Parts will become a lot clearer

- Throw traditional Object Orientation out the door, get rid of all of your assumptions
- Be humbled, it's worth it
- Object Oriented The Good Parts will become a lot clearer
- Collectively, we need to learn how to grow languages - no more waiting around, life is too damn short

my entrance down the rabbit hole.

beautiful mixing of execution and compilation. The source code is a elegant 200 lines of Scheme yet they can express all of Prolog's power, it's purely functional, it is efficient

for an experienced Object Oriented and even a Functional programmer I think this will be a very, very humbling book.



“Let's not make what we don't know into a religion, for God's sake. What we need to do is to constantly think and think and think about what's important. We have to have our systems let us get to the next levels of abstraction as we come to them.”

Special Thanks To

Daniel P. Friedman
William Byrd
Oleg Kiselyov
Luc Maranget
Craig Chambers
Ambrose Bonnaire-Sargeant
Pepijn de Vos
Rich Hickey
Clojure/core

Questions?