Rethinking LL:
JSE & GOO

Jonathan Bachrach

MIT AI Lab

Java Syntactic Extender

- Convenient Syntactic Extension for Conventionally Syntaxed Languages
- Builds on work from Dylan and Lisp
- Joint work with Keith Playford
- OOPSLA-01
  - www.ai.mit.edu/~jrb/jse/

JSE Example

```
forEach(Task elt in tasks)
  elt.stop();
```

```
public syntax forEach {
  case #{ forEach (?:type ?elt:name in ?:expression)
            ?:statement }:
    return #{ Iterator i = ?expression.iterator();
              while (i.hasNext()) {
                ?elt = (?type)i.next();
                ?statement
             } } };
```

Scripting Languages Can Be: Fast, Clean, Safe, Powerful, and of course Fun

- Don't have to trade off fun for speed etc.
- Don't need complicated implementation

- Requires forced rethinking and reevaluation of
  - Technologies - faster, bigger, cheaper
  - Architecture - dynamic compilation
  - Assumptions - …

GOO
Art / Science / Education

- Research/Teaching vehicle
  - For rethinking language design and implementation
- Reaction to a Reaction …
- Targetted for high-performance/interactive software development for
  - Embedded systems
  - Electronic music

GOO Power

Features

- Pure object-oriented
- Multimethods
  - Slot accessors as well
- Dynamic types
  - Ext. param types
- Modules
- Macros
- Restartable exceptions

Builds on

- Proto
- Dylan
- Cecil
- CLOS
- Scheme
- Smalltalk

GOO Simplicity

- PLDI Core Wars
  - 10K Lines Implementation *
  - 10 Page Manual **
  - Hard Limit – “pressure makes pearls”

- Interpreter Semantics
- Speed through “partial evaluation”
- Implementation Serendipity

Complexity is Dangerous to Your Health

- Complexity will bite you at every turn
  ✍Minimize number of moving parts

- Complexity can be exponential in part count unless extreme vigilance is applied
- But vigilance is costly especially reactive
  ✍Apply vigilance towards minimizing part count instead

Simplified Design

Simplification

- No sealing
- Dynamic typing
- No databases
- Type-based opts only
- Prefix syntax
- No VM

Recover *x* with

- Global info / d-comp
- Type inference
- Save-image
- C (C--) backend
- Short + nesting ops
- (Obfuscated) Source

GOO: Speed and Interactivity

- Always optimized
- Always malleable

Incremental Global Optimization

- Always compiled
- During compilation dependency tracks assumptions during compilation
- Reoptimize dependents upon change
- Knob for adjusting number of dependents to make recompilation times acceptable

Managing Complexity

1. Dynamic compilation
2. Subclass? tests
3. Multimethod dispatch

Complexity Example One: Dynamic Compilation

- So you want a dynamic compiler?
  - ✍Throw away interpreter
  - ✍Allow for more global optimizations
- But what about the ensuing complexity?
  - ✍Use source instead of VM
    - Cut out the middle man
  - ✍Use C back-end and shared libraries (e.g., MathMap)
    - More realistically C--
  - ✍Trigger compiler
    - By global optimization dependencies
    - Not profile information

Complexity Example Two: Fast `Subclass?` Tests

- Crucial for the performance of languages
  - Especially languages with dynamic typing
- Used for
  - typechecks
  - downcasts
  - typecase
  - method selection
- Used in
  - Compiler - static analysis
  - Runtime

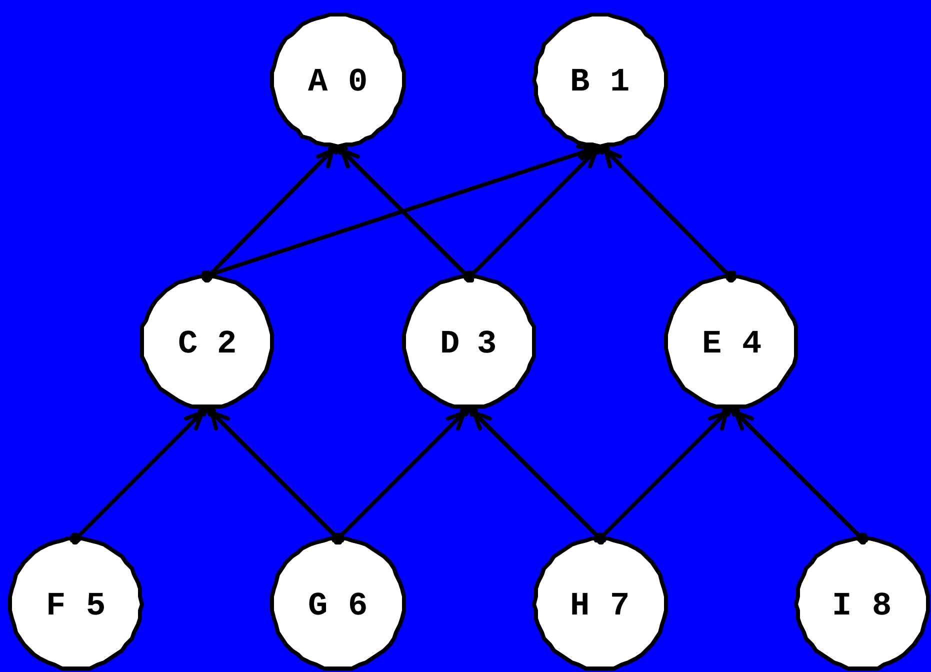
Important Subclass? Measures

- The predicate speed
- The subclass data initialization speed
- The space of subclass data
- The cost of incremental changes
  - Could be full reinitialization if fast enough

Longstanding Problem

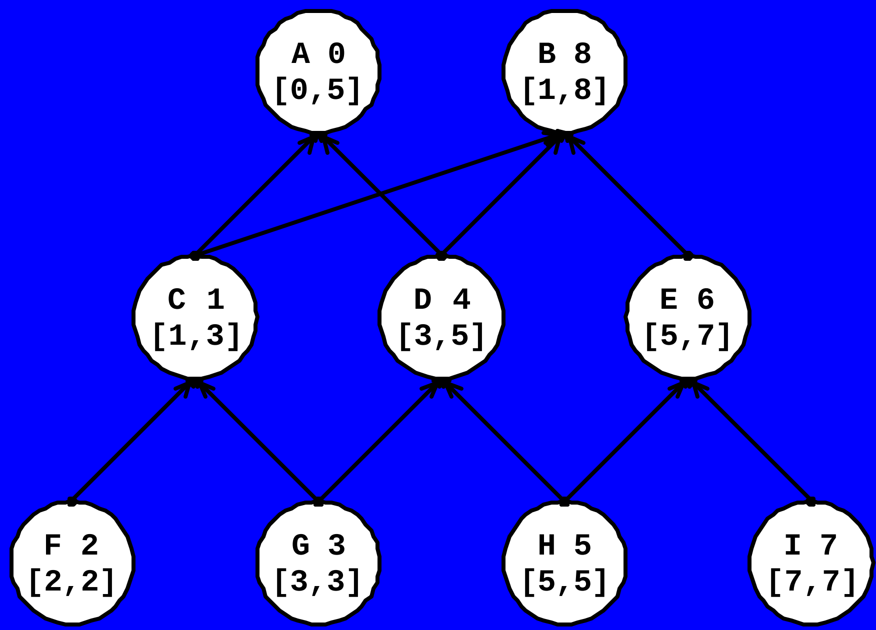
- Choose either
  - Simple algorithm with O(n^2) space or
  - Complicated slower to initialize algorithm with better space properties:
    - PE – Vitek, Horspool, and Krall OOPSLA-97
    - PQE – Zibin and Gil OOPSLA-01

Bit Matrix Algorithm



		A	B	C	D	E	F	G	H	I
		0	1	2	3	4	5	6	7	8
A	0	1	0	1	1	0	1	1	1	0
B	1	0	1	1	1	1	1	1	1	1
C	2	0	0	1	0	0	1	1	0	0
D	3	0	0	0	1	0	0	1	1	0
E	4	0	0	0	0	1	0	0	1	1
F	5	0	0	0	0	0	1	0	0	0
G	6	0	0	0	0	0	0	1	0	0
H	7	0	0	0	0	0	0	0	1	0
I	8	0	0	0	0	0	0	0	0	1

Clumping Ones



		A	C	F	G	D	H	B	E	I
		0	1	2	3	4	5	6	7	8
A	0	1	1	1	1	1	1	0	0	0
C	1	0	1	1	1	0	0	0	0	0
F	2	0	0	1	0	0	0	0	0	0
G	3	0	0	0	1	0	0	0	0	0
D	4	0	0	0	1	1	1	0	0	0
H	5	0	0	0	0	1	0	0	0	0
B	6	0	1	1	1	1	1	1	1	1
E	7	0	0	0	0	0	1	0	1	1
I	8	0	0	0	0	0	0	0	0	1

Packing SubClass Vector

		A	C	F	G	D	H	B	E	I
		0	1	2	3	4	5	6	7	8
A	0	1	1	1	1	1	1	0	0	0
C	1	0	1	1	1	0	0	0	0	0
F	2	0	0	1	0	0	0	0	0	0
G	3	0	0	0	1	0	0	0	0	0
D	4	0	0	0	1	1	1	0	0	0
H	5	0	0	0	0	1	0	0	0	0
B	6	0	1	1	1	1	1	1	1	1
E	7	0	0	0	0	0	1	0	1	1
I	8	0	0	0	0	0	0	0	0	1

```
define function isa-0? (x, y)
  x.id >= y.min-id & x.id <= y.max-id
    & scv[y.base – y.min-id + x.id] = 1
end function
```

A						C			F	G	D			H	B								E				I
A	C	F	G	D	H	C	F	G	F	G	G	D	H	H	C	F	G	D	H	B	E	I	D	H	B	E	I
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1

Eliding Range Checks

```
define function isa-2? (x, y)
  scv[y.base – y.min-id + x.id] = y.id
end function;
```

		A	C	F	G	D	H	B	E	I
		0	1	2	3	4	5	6	7	8
A	0	0	0	0	0	0	0	-	-	-
C	1	-	1	1	1	-	-	-	-	-
F	2	-	-	2	-	-	-	-	-	-
G	3	-	-	-	3	-	-	-	-	-
D	4	-	-	-	4	4	4	-	-	-
H	5	-	-	-	-	5	-	-	-	-
B	6	-	6	6	6	6	6	6	6	6
E	7	-	-	-	-	-	7	-	7	7
I	8	-	-	-	-	-	-	-	-	8

A						C			F	G	D			H	B								E				I
A	C	F	G	D	H	C	F	G	F	G	G	D	H	H	C	F	G	D	H	B	E	I	D	H	B	E	I
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0	0	0	0	0	1	1	1	2	3	4	4	4	5	6	6	6	6	6	6	6	6	7	-	7	7	8

Constant Folding

```
define function isa-2? (x, y)
  scv[y.base – y.min-id + x.id] = y.id
end function;
```

==>

```
define function isa? (x, y)
  scv[y.offset + x.id] = y.id
end function;
```

Multiple Inheritance

- Mixins get assigned randomly in simple preorder walk
- Modified walk to aggressively walk parents

Construction Algorithm

1. Number classes and calculate min/max's
2. Assign offsets and calculate PSCV size
3. Populate PSCV with positives

Preliminary Results

- Blindingly fast to construct
  - Fast enough for incremental changes
- One page of code to implement
- Comparable to PE on wide range of real-world hierarchies
  - E.g. 95% compression on 5500 class flavors hierarchy (4MB bit matrix)
- Exhibits approximately n log n space

Subclass? Miscellaneous

- Can use 8 bit keys for even better compression
- Available from:
  - `www.ai.mit.edu/~jrb/pve/`
- Thanks
  - Eric Kidd
    - Related range compression in undergrad thesis
  - Craig Chambers, James Knight, Greg Sullivan, and Jan Vitek

Complexity Example 3: Dispatch

- For a given generic function and arguments choose the most applicable method
- Example:
  - Gen: `(+ x y)`
  - Mets: `num+ int+ flo+`
  - Args: `1 2`
  - Met: `int+`

Subtype? Based Dispatch Methodology

Steps

- Dynamic subtype? based decision tree
  - Choose more specific specializers first
  - Choose unrelated specializers with stats
- Inline small methods
- Inline decision trees into call-sites

Examples

-
```
(fun (x y)
  (if (isa? x &lt;int&gt;)
      ...)))
```
  - Discriminate `int+` and `flo+` before `num+`
  - Discriminate `int+` before `flo+`
- `int+` (and slot accessors)
- `(+ x 1)` (allowing partial evaluation at call-site)

Subtype? Based Dispatch Happy Synergies

- Few moving parts
- “tag-checked” arithmetic for free
- Static dispatch for free
- One arg case comparable to vtable speed
  - Fewer indirect jumps
- Dynamic type-check insensitive to class numbering

GOO Status

Working

- Fully bootstrapped
- Linux and Win32 Ports
- Runtime system tuned
- C based dynamic compiler

In Progress

- Fast subclass?
- Dependency tracking
- Type inference
- Decision tree generation
- Parameterized types

GOO Credits Etc

- Thanks to
  - Craig Chambers
  - Eric Kidd, James Knight, and Andrew Sutherland
  - Greg Sullivan
  - Howie Shrobe (and DARPA) for funding

- Available under GPL:
  - `www.googoogaga.org`