Implementing Programming Languages for Fun and Profit with OMeta

Alessandro Warth
Viewpoints Research Institute & UCLA
Who am I?
NOT David Simmons
NOT Billy Idol
programming languages
STEPS

... toward the reinvention of programming
The STEPS Project

**Goal:** To create a highly useful end-user system including...

- operating system
- programming environment
- “applications”
- graphics, sound

\[ \text{personal computing} \]
... in under 20,000 LOC!
Windows XP
~40 million LOC
Squeak

~200 thousand LOC
Why?

• To put people in charge of their own SW destinies
  • No single person can understand 40,000,000 LOC (~library)
  • You can “own” 20,000 LOC (~book)
Why? (cont’d)

- Didactic value
  - a curriculum for university students to learn about powerful ideas
  - may even be good for AP C.S.
Programming Languages

- STEPS: code size, understandability
- Choice of programming language has a big impact on both
- What’s the right one for STEPS?
Long Lines...

- It takes lots of time and effort to implement a programming language
- limits how much experimenting we can do
Big and Bad

- Traditional PL implementations are **BIG**
- only have 20k LOC for the whole thing!
OMeta

an OO language for pattern matching
JavaScript (OMeta/Squeak)

```javascript
polygon = function(side) {
    for (var i = 0; i < side; i++) {
        forward(100);
        turn(360 / side);
    }
}
polygon(5);
```
Sun’s Lively Kernel (OMeta/COLA)

```
P = new Pen();
P.setPenColor(Color.red);
for(var i=1; i<=50; i++)
    { P.go(2*i); P.turn(60); };
P.drawLines();
```
Toylog

- Front-end to Prolog for children
- Runs on Squeak
- ~70 LOC

Homer is Bart’s father.
Marge is Bart’s mother.
x is y’s parent if x is y’s father or
  or x is y’s mother.
x is Bart’s parent?
Toylog Demo
What can OMeta do for you?

- Make your life easier with DSLs
  - good DSLs come from people who need them
  - not PL people
- Make your apps scriptable by end-users
  - ... without making them learn Smalltalk
Roadmap

- A brief overview of OMeta
- pattern matching
- OO features
- ...
- OMeta/JS
Traditional PL Implementation
Traditional PL Implementation

visitors, for AST transformations and code generation

lex, for lexical analysis

yacc, for parsing
Pattern Matching: A Unifying Idea!

- **lexical analysis**: characters → tokens
- **parsing**: tokens → parse trees
- **constant folding and other optimizations**: parse trees → parse trees
- **code generation**: parse trees → code
Why use PM for everything?

- Simplicity
- Less stuff to learn (lowers learning curve)
- Great for extensibility
  - trad. impls hard to extend
  - OMeta: every part of PL impl. (e.g., parsing, tree traversals, codegen) can be extended using same mechanism
Pattern Matching

• Other langs have PM, do we need OMeta?
• ML-style pattern matching
  • great for tree transformations
  • not good for lexing/parsing
  • “That’s what ML-lex and ML-yacc are for!”
• OMeta is based on PEGs
Parsing Expression Grammars (PEGs) [Ford, '04]

- Recognition-based foundation for describing syntax
- Only prioritized choice
  - no ambiguities
- easy to understand
- Backtracking, unlimited lookahead
- Semantic predicates, e.g., \(?[x == y]\)
PEGs, OMeta style

dig ::= $0 \mid \ldots \mid $9

num ::= <num> <dig>
    \mid <dig>

expr ::= <expr> $+ <num>
    \mid <num>
PEGs, OMeta style

dig ::= ($0 | ... | $9):d

num ::= <num>:n <dig>:d
    | <dig>

expr ::= <expr>:e $+ <num>:n
    | <num>
PEGs, OMeta style

dig ::= ($0 | ... | $9):d => [d digitValue]

num ::= <num>:n <dig>:d => [n * 10 + d]
   | <dig>

expr ::= <expr>:e $+ <num>:n => [{#plus. e. n}]
   | <num>


Increasing Generality

- PEGs operate on streams of characters
- OMeta operates on streams of objects
  - anything matches any one object
  - strings, e.g., ‘hello’
  - symbols, e.g., #ans
  - numbers, e.g., 42
  - “listy” objects, e.g., {'hello' #ans 42 {}}
Example: evaluating parse trees

eval ::= \{#plus <eval>:x <eval>:y\} \Rightarrow [x + y]

| <anything>:n ?[n isNumber] \Rightarrow [n]

\{#plus. \{#plus. 1. 2\}. 3\} \rightarrow 6
dig ::= ($0 | ... | $9):d  => \[ d \text{ digitValue} \]

num ::= <num>:n <dig>:d  => \[ n * 10 + d \]
| <dig>

expr ::= <expr>:e $+ <num>:n  => \{ \texttt{#plus. e. n} \}
| <num>
OMeta is Object-Oriented

dig ::= ($0 | ... | $9):d => [d digitValue]
num ::= <num>:n <dig>:d => [n * 10 + d]
    | <dig>
expr ::= <expr>:e $+ <num>:n => [{#plus. e. n}]
    | <num>
OMeta is Object-Oriented

MyLang

dig ::= ($0 | ... | $9):d  =>  [d digitValue]
num ::= <num>:n <dig>:d  =>  [n * 10 + d]
    | <dig>
expr ::= <expr>:e $+ <num>:n  =>  [#plus. e. n]}
    | <num>
OMeta is Object-Oriented

OMeta

anything ::= ...
...

dig ::= ($0 | ... | $9):d => [d digitValue]
num ::= <num>:n <dig>:d => [n * 10 + d]
| <dig>
expr ::= <expr>:e $+ <num>:n => [#plus. e. n]
| <num>

MyLang
OMeta is Object-Oriented

OMeta

anything ::= ...

...

dig ::= ($0 | ... | $9):d  => [d digitValue]

MyLang

num ::= <num>:n <dig>:d  => [n * 10 + d]
  | <dig>

expr ::= <expr>:e $+ <num>:n => [{#plus. e. n}]
  | <num>

MyLang++

expr ::= <expr>:e $- <num>:n => [{#minus. e. n}]
  | <super #expr>
Parameterized rules

digit ::= $0 | $1 | $2 | $3 | $4
| $5 | $6 | $7 | $8 | $9
Parameterized rules

digit ::= $0 \mid $1 \mid $2 \mid $3 \mid $4
| $5 \mid $6 \mid $7 \mid $8 \mid $9

range :a :b ::= <anything>:x ?[x >= a]
?^[x <= b] => [x]
Parameterized rules

digit ::= $0 | $1 | $2 | $3 | $4
      | $5 | $6 | $7 | $8 | $9

range :a :b ::= <anything>:x ?[x >= a]
               ?[x <= b] => [x]

digit ::= <range $0 $9>
Higher-order rules

formals ::= <name> ($, <name>)*
args ::= <expr> ($, <expr>)*
Higher-order rules

formals ::= <name> ($, <name>)*
args ::= <expr> ($, <expr>)*

listOf :p ::= <apply p> ($, <apply p>)*
Higher-order rules

formals ::= <name> ($, <name>)*
args    ::= <expr> ($, <expr>)*

listOf :p ::= <apply p> ($, <apply p>)*

formals ::= <listOf #name>
args     ::= <listOf #expr>
OMetaJS = OMeta + JavaScript

OMeta Parser

JS Parser
OMetaJS = OMeta + JavaScript

OMeta Parser

JS Parser

OMetaJS Parser
OMetaJS = OMeta + JavaScript
OMetaJS = OMeta + JavaScript
OMetaJS = OMeta + JavaScript

- duplicated effort
- versioning problem
Foreign rule invocation

- Lend input stream to another grammar

```plaintext
ometajsp {  
ometakjs ::= <foreign OMetaParser #grammar>  
             | <foreign JSParser    #stmt>
}
```

- Compose multiple grammars w/o worrying about name clashes
This OMeta, That OMeta

• Several versions of OMeta:
  • OMeta/Squeak
  • OMeta/COLA
  • OMeta/JS
  • ...

• Slightly different syntaxes

• Use different languages for semantic actions and predicates
OMeta/JS
JavaScript Workspace

- Takashi Yamamiya’s handy work
- Inspiration for OMeta/JS
- Workspace-style interface for JavaScript
- Runs inside the web browser, works like a Wiki
JavaScript

- Dynamic language
- First-class functions (closures)
- Late-bound
- `eval()`
- Huge performance improvements lately
  - new webkit runs at “.5 Squeaks”
Plus...

IT’S EVERYWHERE!
World's BEST DAD

ASSEMBLY LANGUAGE
Switch to web browser
Forget Guitar Hero... I could be the next Dan Ingalls!
For more info...

- **DLS’07 Paper**
- **OMeta Mailing List**
  [http://vpri.org/mailman/listinfo/ometa](http://vpri.org/mailman/listinfo/ometa)
- **OMeta/JS wiki**
  [http://jarrett.cs.ucla.edu/ometa-js](http://jarrett.cs.ucla.edu/ometa-js)
- **Ask questions now**
THE END
Selected Related Work

- Parsing Expression Grammars [Ford, '04]
- LISP70 Pattern Matcher [Tesler et al., '73]
- Parser combinator libraries [Hutton, '92], [Bracha’07]
- “Modular Syntax” [Grimm, ’06]