Meta-programming & you

Robin Message

Cambridge Programming Research Group

10^{th} May 2010

What's meta-programming about?

```
result=somedb.customers.select
        {first_name+" "+last_name}
    where name LIKE search_query+"%"
```

What's meta-programming about?

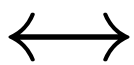
```
result=somedb.customers.select
        {first_name+" "+last_name}
    where name LIKE search_query+"%"
```

```
result=somedb.runQuery("SELECT first_name,last_name FROM customers
    WHERE name LIKE ?",search_query+"%")
    .map{|first_name,last_name|
        first_name+" "+last_name
    }
```

What's meta-programming about?

```
result=somedb.customers.select
        {first_name+" "+last_name}
    where name LIKE search_query+"%"
```

```
result=somedb.runQuery("SELECT first_name,last_name FROM customers
    WHERE name LIKE ?",search_query+"%")
    .map{|first_name,last_name|
        first_name+" "+last_name
    }
```



AWESOME!

What is meta-programming?

- Metaprogramming is the writing of computer programs that write or manipulate other programs (or themselves) as their data, or that do part of the work at compile time that would otherwise be done at runtime.
- A metaprogram is a program that manipulates other programs (or itself) as its data. The canonical example is a compiler.
- Meta-programming, by which is meant that one constructs an interpreter for a language close to the problem and then writes problem solutions using the primitives of this language.

"Code that creates, manipulates or influences other code."

Where are we going?

1. What is meta-programming?
2. Why do meta-programming?
   - Optimisation
   - Abstraction
   - Expressiveness
3. What is a Domain Specific Language?
4. How can we implement a DSL in...
   - Java
   - Ruby
   - Lisp

Why?

Q: Why do meta-programming?

Why?

Q: Why do meta-programming?

Optimisation Specialise the interpretation of the language

Abstraction Improve the language

Expressiveness Create a new language

Examples

- C++ templates
- FFTW
- Compiler plugins
- Partial evaluation
- Static typing

Abstraction

Whatever abstraction mechanism your language provides, there will always be parts of your program that can't be abstracted[citation needed].

Abstraction

Whatever abstraction mechanism your language provides, there will always be parts of your program that can't be abstracted[citation needed].

Examples

Aspects Expressing cross-cutting concerns

Abstraction

Whatever abstraction mechanism your language provides, there will always be parts of your program that can't be abstracted[citation needed].

Examples

Aspects Expressing cross-cutting concerns

Java 1.5 iterators

```
1 for(Iterator i=collection.iterator();i.hasNext();) {
2     Element e=i.next();
```

$\longleftrightarrow$

```
1 for(Element e:collection) {
```

Abstraction

Whatever abstraction mechanism your language provides, there will always be parts of your program that can't be abstracted[citation needed].

Examples

Aspects Expressing cross-cutting concerns

Java 1.5 iterators

```
1 for(Iterator i=collection.iterator();i.hasNext();) {
2     Element e=i.next();
```

$\longleftrightarrow$

```
1 for(Element e:collection) {
```

Why can't we describe new syntax and control structures in some kind of meta-Java?

Expressiveness

Express programs that cannot be represented in your language.

Examples

A Prolog interpreter Semantics unlike most other languages

Domain Specific Languages Semantics of a particular domain

What is a Domain Specific Language?

Q: Give me some examples

What is a Domain Specific Language?

Q: Give me some examples

We will discus shallow embedded DSLs.

embedded within the syntax of an existing language

shallow embedding using the data types of an existing language

domain area of interest

specific specialised to, designed for

Where are we now?

1. What is meta-programming?
2. Why do meta-programming?
   - Optimisation
   - Abstraction
   - Expressiveness
3. What is a Domain Specific Language?
4. How can we implement a DSL in...
   - Java
   - Ruby
   - Lisp

How to implement a DSL

Or, let's get on with the code!

Running example

Expressions made up of numbers, operators and variables

$area = (width + 8) * (height + 4)$

```
1 abstract class Expression {
2     abstract double calculate(Map<String,double> env);
3     Expression add(Expression b) {
4         return new Add(this,b);
5     }
6     Expression mul(Expression b) {
7         return new Mul(this,b);
8     }
9 }
```

Method Chaining

Add and Multiply classes

```
1  class Add extends Expression {
2      private Expression a,b;
3      Add(Expression _a,Expression _b) {
4          a=_a;b=_b;
5      }
6      double calculate(Map<String,double> env) {
7          return a.calculate(env)+b.calculate(env);
8      }
9  }
10 class Mul extends Expression {
11     private Expression a,b;
12     Mul(Expression _a,Expression _b) {
13         a=_a;b=_b;
14     }
15     double calculate(Map<String,double> env) {
16         return a.calculate(env)*b.calculate(env);
17     }
18 }
```

Variables and numbers

```
1  class Var extends Expression {
2      private String name;
3      Var(String _n) {
4          name=_n;
5      }
6      double calculate(Map<String,double> env) {
7          return env.get(name);
8      }
9  }
10 class Num extends Expression {
11     private double num;
12     Num(double _n) {
13         num=_n;
14     }
15     double calculate(Map<String,double> env) {
16         return num;
17     }
18 }
```

Is our new DSL pleasant to use?

```
1 static Expression add(Expression a,Expression b){return new Add(a,b
      );}
2 static Expression variable(String name){return new Var(name);}
3 static Expression number(double n){return new Num(n);}
4
5 Expression area=add(variable("width"),number(8)).mul(variable("
      height")).add(number(4)));
6
7 area.calculate(env);
```

Is Java any good for DSLs?

Q: Well, what do you think?

Is Java any good for DSLs?

Q: Well, what do you think?

- Method chaining
- Static binary methods are repetitive
- Wrapping names and numbers
- Expression problem
- Tree structure

Not really

```ruby
class Expression
  def initialize &b
    @block=b
  end
  def calculate environment
    @env=environment
    instance_eval &@block
  end
  def method_missing name,*args
    if args.length==0
      @env[name]
    else
      super name,*args
    end
  end
end
```

Is Ruby good for DSLs?

```
1 area=Expression.new {
2   (width+8) * (height+4)
3 }
4
5 area.calculate :width => 100, :height => 5
```

Looks pretty good to me

Why is Ruby good for DSLs?

- Blocks
- Symbols
- method_missing and other hooks
- Always have eval (MAD)

Why is Ruby good for DSLs?

- Blocks
- Symbols
- method_missing and other hooks
- Always have eval (MAD)

Q: Any problems with this approach?

Dynamic evaluation and namespace pollution

```
1 whoops=Expression.new {
2     true + false
3 }
4 whoops.calculate :true => 1, :false => 0
```

Trade-off in shallow embedding of DSLs – it is easy to confuse DSL features with language features.

Introduction to Lisp

Scheme in one slide...

Datatypes Literals (`"foo"`, `42`), symbols (`name`, `*`, `some-long-thing!`), lists `(a b c)`, pairs `(a . b)`

Evaluation A literal is a literal, a symbol has a value and a list is a function call

Functions `(function-name arguments...)` evaluates the arguments, then evaluate the function-name and invoke what it returns with the evaluated arguments

Macros Look like functions, but might evaluate differently

List operations `list` returns its arguments as a list, `cons`≡cons, `car`≡head, `cdr`≡tail, `cadr`≡head of tail, and so on

Quoting `'x`≡`(quote x)`, and quote is a macro that returns its argument without evaluating it.
`` `(a ,b c) `` means `(list 'a b 'c)`

Built-ins `define if cond equal? list? map eval`

Expressions in Lisp

```
1  (define (deinfix exp)
2    (cond
3      ((and (list? exp) (equal? 3 (length exp))) (list (cadr exp) (
           deinfix (car exp)) (deinfix (caddr exp))))
4      (else exp)
5  ) )
6
7  (define (calculate exp env)
8    (map (lambda (v) (eval `(define ,(car v) ,(cdr v)))) env)
9    (eval exp)
10 )
11
12 (define area '((width + 8) * (height + 4)))
13 (calculate area '((width . 100) (height . 50)))
```

"deinfix" turns $(a * b)$ into `(* a b)`.

"calculate" evaluates a Lisp expression in a specific environment

Bit ugly, only fully bracketed expressions

Remember how we wanted meta for abstraction?

```
1  (define (prec op)
2    (case op
3      ('+ 1) ('- 1) ('* 2) ('/ 2) ('^ 3) (else 0)
4  ) )
5  (define (rassoc op) (eq? op '^))
6  (define (munge exp)
7      ...
8  )
9
10 (define-syntax expression
11   (syntax-rules ()
12     ((_ exp ...)
13      (deinfix (munge '(exp ...)))
14 ) ) )
15
16 (calculate (expression (width + 8) * (height + 4)) '((width . 100)
       (height . 50)))
```

Competition: A working version of "munge" in Lisp.

Differentiation

```
(define (diff exp var)
  (cond
    ((symbol? exp) (if (eq? exp var) 1 0))
    ((number? exp) 0)
    ((equal? 3 (length exp)) (diff-binop (car exp) (cadr exp) (
        caddr exp) var))
    (else (raise (string-append "Not an expression" exp)))
) )
(define (diff-binop op a b var)
  (cond ((equal? '+ op) `(+ ,(diff a var) ,(diff b var)))
        ((equal? '- op) `(- ,(diff a var) ,(diff b var)))
        ((equal? '* op) `(+ (* ,(diff a var) ,b) (* ,a ,(diff b var
            )))))
        ((equal? '^ op) (if (number? b)
                            (if (eq? b 0) 0 (diff-binop '* a `(^ ,a
                                ,(- b 1)) var))
                            (raise "^ not a non-number")
                        ))
        (else (raise (string-append ``Can't differentiate '' (
            symbol->string op))))
)  )
```

Simplification

```
(define (simplify exp)
  (cond ((and (list? exp) (equal? 3 (length exp))) (simplify-binop
      (car exp) (cadr exp) (caddr exp)))
        (else exp)
) )
(define (simplify-binop op ina inb)
  (define sima (simplify ina))
  (define simb (simplify inb))
  (match (list op sima simb)
    ((list '+ a 0) a)
    ((list '- a 0) a)
    ((list '+ 0 b) b)
    ((list '+ a a) `(* 2 ,a))
    ((list '- a a) 0)
    ((list '* a 0) 0)
    ((list '* 0 b) 0)
    ((list '* a 1) a)
    ((list '* 1 b) b)
    ((list '^ a 0) 1)
    ((list '^ a 1) a)
    (else (list op sima simb))
) )
(simplify (diff (expression x ^ 2 - 2 * x + 1) 'x))
```

Is Lisp good for DSLs?

Is Lisp good for DSLs?

HELL YEAH!

- Lightweight syntax
- Symbols
- Code as data
- Macros
- define-syntax
- eval

Conclusions

- Meta-programming lets us optimise, improve abstraction and increase expressiveness
- DSLs are a powerful and useful tool
- Lisp is an awesome tool for meta-programming, because code is data

Conclusions

- Meta-programming lets us optimise, improve abstraction and increase expressiveness
- DSLs are a powerful and useful tool
- Lisp is an awesome tool for meta-programming, because code is data

Thank you! Any questions?