Vrije Universiteit Brussel

DLD

Dynamic Languages Day

February 13th 2006

Vrije Universiteit Brussel

Vrije Universiteit Brussel

Agenda

10:00 - 10:30 Scheme as an introductory language

Viviane Jonckers (SSEL)

10:30 - 11:20 Smalltalk

Johan Brichau (PROG/LIFL), Roel Wuyts (deComp/ULB)

11:20 - 11:40 Coffee Break

11:40 - 12:30 Self

Ellen Van Paesschen (PROG)

12:30 - 13:30 Lunch

13:30 - 14:50 Generic Functions in Common Lisp / CLOS

Pascal Costanza (PROG)

14:50 - 15:10 Coffee Break

15:10 - 16:30 The CLOS Metaobject Protocol

Pascal Costanza (PROG)

Vrije Universiteit Brussel

Why Computers Scientists should know

- Every Computer Scientist should GO META at least once in his/her life (Dave Thomas in JOT)
  - Those who have experienced the ”engine room” via a Scheme meta-circular interpreter, or Smalltalk or CLOS meta-class programming have a fundamentally deeper perspective on computation
  - Providing and understanding the meta view is the way to toss off the syntactic baggage of a new programming language and identify and focus on its unique features and anomalies
  - Building IDE, i.e. inspectors, interpreters, debuggers, browsers, serialisation, etc. requires the meta view

Metaprogramming

- **Metaprogramming** is the writing of programs that write or manipulate other programs (or themselves) as their data, e.g. interpreters, compilers, debuggers, etc.
- The language in which the metaprogram is written is called the **meta-language**. The language of the programs that are manipulated is called the **object-language**.
- The capacity of a programming language to be its own meta-language is called **reflexivity**.

Reflection

- In computer science, reflection is the ability of a program to observe and possibly modify itself
- A language supporting reflection provides a number of features available at runtime such as:
    - Discover and modify source code constructions as first-class objects at runtime
    - Convert a string matching the symbolic name of a class or function into a reference to or invocation of that class or function
    - Evaluate a string as if it were a source code statement at runtime
- Reflection is a valuable language feature for facilitating metaprogramming. Having the programming language itself as a first class data type (as in Lisp) is also very useful.

Vrije Universiteit Brussel

Meta Object Protocol

- A *Meta Object Protocol* (MOP) is a method for accessing the guts of an object system through a *Meta Class*. A *Meta Class* is the class for a class.
- *Meta Classes* are responsible for the overall behaviour of an object system. They handle delegation, access, etc.
- Generally there are 2 major aspects to a meta object protocol
  - **Introspection** The ability to read the attributes of an object or class such as: what is this classes subclass? What methods are available to this class?
  - **Intercession** The ability to modify the behaviour of an object or class such as: change this objects parent, bind this method to this class or object.)
- (The JAVA core reflection API provides introspection only)

DLD

Scheme as a first Programming Language for Computer Scientists

VIVIANE JONCKERS

Vrije Universiteit Brussel

Practicalities

- New to all students: less risk of mind polution
- Manageable: very little syntax, minimalist approach
- Interpreted environment: ease of getting started

Vrije Universiteit Brussel

Expressiveness in style

- Allows within one language the introduction of:
  - Functional programming
  - Imperative programming
  - Data-directed programming
  - Object-oriented programming
  - Stream programming
  - Constraint programming
  - Logic programming

Main features

- Functional language
- Dynamically typed
- Lexically scoped
- Garbage collector

- Functions are first-class objects
- Higher order functions
- Closures

- Symbols as primitive data type
- Cons-cells as universal data structuring mechanism
- Natural recursion in operation and data
- Source code is just a list

Functional, dynamic typing

```
>>> (define (average x y)
         (/ (+ x y) 2))
average
>>> (average 6 8)
7

>>> (define (fac n)
        (if (= 0 n)
            1
            (* n (fac (- n 1)))))
fac
>>> (fac 5)
120
```

Lexical scope

```
>>> (define (sqrt x)
         (define (good-enough?  guess)
           (< (abs (- (square guess) x)) 0.001))
         (define (improve guess)
           (average guess (/ x guess)))
         (define (sqrt-iter guess)
           (if (good-enough? guess)
             guess
             (sqrt-iter (improve guess))))
        (sqrt-iter 1))
sqrt

>>> (sqrt 5)
2.2360688956433634
```

Functions as parameters

```
>>> (define (search func a b)
        (define (close-enough? x y) (< (abs (- x y)) 0.001))
          (let ((mid (average a b)))
            (if (close-enough? a b) mid
                (let ((value (func mid)))
                  (cond
                    ((positive value) (search func a mid))
                    ((negative value) (search func mid b))
                    (else mid))))))
search

>>> (define (map func somelist)
        (cond
          ((null? somelist) '())
          (else (cons (func (car somelist)) (map func (cdr somelist))))))
map
```

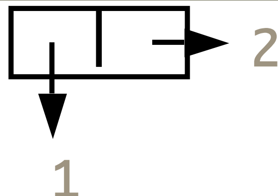
Functions as return value
Closures

```
>>> (define (make-account balance)
      (define (withdraw amount)
        (if (>= balance amount)
            (begin
              (set! balance (- balance amount))
              balance)
            "insufficient funds"))
     (define (deposit amount)
       (set! balance (+ balance amount))
       balance)
     (define (dispatch m)
       (cond ((eq? m 'withdraw) withdraw)
             ((eq? m 'deposit) deposit)
             (else (error "unknown request --MAKE-ACCOUNT" m))))
     dispatch)
make-account
```

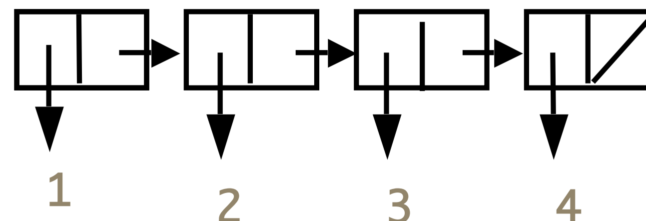
Vrije Universiteit Brussel

Cons as universal building block

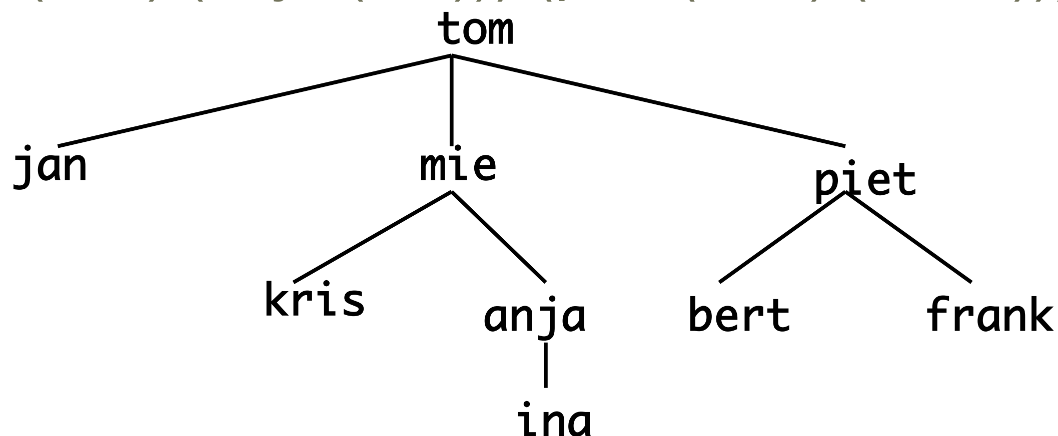
```
>>> (cons 1 2)
(1.2)
```




```
>>> (cons 1 (cons 2 (cons 3 (cons 4 '()))))
(1 2 3 4)
```




```
>>> (cons …
(tom (jan) (mie (kris) (anja (ina))) (piet (bert) (frank))))
```



Programs are data

- Cons has the closure property
- Linked Lists are one of Lisp languages' major data structures
- In Lisp all program code is written as s-expressions, i.e. parenthesized lists
- As a result Lisp programs can manipulate source code as a data structure which allows
  - To easily write a meta-circular interpreter
  - Through the macro system to create new syntax for new "little languages" embedded in Lisp

Vrije Universiteit Brussel

Meta-circular interpreter Eval

```
(define (eval exp env)_
  (cond
    ((self-evaluating? exp) exp)_
    ((variable? exp) (lookup-variable-value exp env))_
    ((quoted? exp) (text-of-quotation exp))_
    ((assignment? exp) (eval-assignment exp env))_
    ((definition? exp) (eval-definition exp env))_
    ((if? exp) (eval-if exp env))_
    ((lambda? exp) (make-procedure (lambda-parameters exp)_
                                   (lambda-body exp) env))_
   ((begin? exp)(eval-sequence (begin-actions exp) env))_
   ((cond? exp) (eval (cond->if exp) env))_
   ((application? exp) (apply (eval (operator exp) env)_
                              (list-of-values (operands exp) env)))_
   (else_(error "Unknown expression type -- EVAL" exp))))
```

Meta-circular interpreter Apply

```
(define (apply procedure arguments)_
  (cond
   ((primitive-procedure? procedure)_
     (apply-primitive-procedure procedure arguments))_
   ((compound-procedure? procedure)_
    (eval-sequence_ (procedure-body procedure)_
        (extend-environment_
           (procedure-parameters procedure)_
           arguments_
           (procedure-environment procedure))))_
    (else (error_"Unknown procedure type -- APPLY" procedure))))
```

Adding new syntax through macros

```
>>> (define-macro while
        (lambda (condition . todo)
          `(do ()
               ((not ,condition) #t)
               ,@todo)))
while

>>> (define-macro repeat
        (lambda (todountilcond)
          `(do ()
             (,(last todountilcond) #t)
             ,@(butlast
                   (butlast
                      todountilcond)))))
repeat
```

Agenda

10:00 - 10:30 Scheme as an introductory language

Viviane Jonckers (SSEL)

10:30 - 11:20 Smalltalk

Johan Brichau (PROG/LIFL), Roel Wuyts (deComp/ULB)

11:20 - 11:40 Coffee Break

11:40 - 12:30 Self

Ellen Van Paesschen (PROG)

12:30 - 13:30 Lunch

13:30 - 14:50 Generic Functions in Common Lisp / CLOS

Pascal Costanza (PROG)

14:50 - 15:10 Coffee Break

15:10 - 16:30 The CLOS Metaobject Protocol

Pascal Costanza (PROG)

Vrije Universiteit Brussel

JAVA Core Reflection API

- Accomodates two categories of applications:
  - One category is comprised of applications that need to discover and use all of the public members of a target object based on its run-time class. These applications require run-time access to all the public fields, methods, and constructors of an object. Examples in this category are services such as *Java*(TM) Beans, and lightweight tools, such as object inspectors. These applications use the instances of the classes Field, Method, and Constructor obtained through the methods getField, getMethod, getConstructor, getFields, getMethods, and getConstructors of class Class.
  - The second category consists of sophisticated applications that need to discover and use the members declared by a given class. These applications need run-time access to the implementation of a class at the level provided by a class file. Examples in this category are development tools, such as interpreters, inspectors, and class browsers, and run-time services, such as *Java*(TM) Object Serialization. These applications use instances of the classes Field, Method, and Constructor obtained through the methods getDeclaredField, getDeclaredMethod, getDeclaredConstructor, getDeclaredFields, getDeclaredMethods, and getDeclaredConstructors of class Class.

Levels of meta

- 1 Level System All objects can be viewed as classes and all classes can be viewed as objects (as in Self), there is no need for meta-classes as objects describe themselves
- 2 Level System All Objects are instances of a Class but Classes are not accessible to programs
- 3 Level System All objects are instances of a class and all classes are instances of Meta-Class. The Meta-Class is a class and is therefore an instance of itself (really making this a 3 1/2 Level System). This allows classes to be first class objects and therefore classes are available to programs
- 5 Level System What Smalltalk provides. Like a 3 Level System, but there is an extra level of specialized Meta-Classes for classes. There is still a Meta-Class as in a 3 Level System, but as a class it also has a specialized Meta-Class, the "Meta-Class class" and this results in a 5 Level System: object, class, class class (Smalltalk's Meta-Classes), Meta-Class, Meta-Class class