Revision: 1.6

1st European Lisp and Scheme Workshop

Two local optima : Lisp and Scheme

C.Queinnec[a]

[a] mailto:Christian.Queinnec@lip6.fr

Content

- Lisp and Scheme
- and the others ...
- and now ?
- in quest of the next breakthrough !

Some milestones

1958	Lisp	J.McCarthy
1975	Scheme	Steele and Sussmann
1984	Common Lisp	ANSI X3J13
1997	IS Lisp	ISO-IEC/JTC1/SC22/WG13

and a fractal map of variants!

Sizes of standard

- Scheme : 50 pages (IEEE standard)
- Common Lisp : 1000 pages (ANSI standard)
- IS Lisp : 200 pages (ISO standard)

Lisp vs Scheme

- sex of `()`
- number of namespaces
- binding discipline
- libraries
- cultural usages

NIL

In Scheme : `#f` $\not\equiv$ `()` $\not\equiv$ `nil`

`()` is not even a legal program and `nil` is not defined.

LISP$_1$, LISP$_2$

How is evaluated `(f a b)` ?

In Scheme, the evaluator for `f`, `a` and `b` is the same.

In Lisp, the evaluator for `f` is different. Hence the need to coerce from one namespace to another as in

```
(funcall (function f) a b)
```

instead of

```
(f a b)
```

Algebraic unregularity

```
(if p (f a) (f b)) ≡ (f (if p a b))
(if p (f a) (g a)) ≡ (funcall (if p (function f) (function g)) a)
```

LISP$_n$

Numerous in COMMON LISP : argumental, functional, `block`, `type`, etc.

`catch` and `throw` do not introduce namespaces since they use values rather than names.

Binding discipline

Lexical or dynamic binding discipline within Common Lisp

Lexical only in Scheme (except for `call-with-current-output-port` and other IO-related functions)

```
(defvar a 0)
(defun (f b)
   (list (dynamic-ref a) b) )
(dynamic-let ((a 1))
   (f (+ 1 a)))          → (1 2)
(f 3)                    → (? 3)
```

Lexical scope connects a text area to where a variable is usable. This is a static property.

Dynamic scope is related to the duration of a computation (i.e., the extent of the stack). The dynamic environment is a run-time entity.

Dynamic scope was the scoping discipline of Lisp interpreters while lexical scope was the scoping discipline of Lisp compilers.

Both scopes are useful though dynamic scope is harder to master.

Deployment relies on dynamic scope (think of *web.xml*) with XML data and poor code abstraction. Possible solutions :

- single language ?
- single IDE ? (with cross checking code, descriptor, names)

Divergent stuff

According to the Scheme standard : no keyword, no default value within functions ; no structure, no class, no exception, small libraries.

`assq, assv, assoc` `(assoc key alist :eq comp)`

No continuation in Common Lisp, no hygienic macros.

Continuations

```
(when (enough-time)
```
show "Continuation" slides ...)

Hygienic macros

If x is free in a macro-expanded form then x refers to the same thing x denotes in the definition of the macro.

```
(let ((results '())
      (compose cons) )
  (let-syntax
     ((push (syntax-rules ()
              ((push e)
               (set! results (compose e results)) ) )))
       body
       results ) )
```

Cultural usages

In Scheme : a strong bias towards (special) functions instead of special forms.

```
(let/cc k
   body )
```

```
(call/cc (lambda (k)
           body ))
```

Scheme favors programming with thunks :

```
(catch 'tag (lambda () body))
```

Less keywords implies less code for (naïve) evaluation but more work for a (real, smart) compiler.

Partial conclusions

- Scheme has a simpler semantics and much less core concepts to teach (from simplest) : `define`, `if`, `let`, `quote`, `begin`, `lambda`, `letrec`, `set!`, `call/cc` and macros.
- Common Lisp has numerous, important and useful libraries
- Common Lisp is difficult to subset
- both have difficulties to exchange source code
- Death of research on environments (last was InterLisp)
- They share syntax and mostly functional style but **do Lisp and Scheme really belong to the same family ?**

The influence of Lisp

- Lists (`car`, `cdr` and `cons`)
- Data (Plist) driven programming
- Lisp machine, IDE
- GC, dynamic loading/linking
- Linguistic laboratory (Logo, Smalltalk, Planner, etc.)
- Long jumps
- Fast interpreters (French school with LeLisp, DSL)

The influence of Lisp (continued)

- Functions as regular data (closures)
- Program as data
  - macros (programs can be handled)
  - syntactic abstraction (DSL)
  - data as program as well (sub-specific languages `loop`, `format`, etc.)

No longer specific features

- garbage collection nearly everywhere
- dynamically loading code nearly everywhere
- closures in Perl, Python, Ruby, (sort of in Java with inner classes)
- long jumps (exceptions) in C, Ada, Java
- objects nearly everywhere
- macros : STL in C++, generic in Ada and Xdoclet, AOP in Java
- available run-time evaluator in sh, Perl, Dynamic Java, PHP
- packages nearly everywhere

Still characteristic features

- Continuations
- Some sort of reflexion
- dynamic substitution of code
- Generic functions *à la CLOS* to add behavior to already existing objects

More specific don't-have features

- Persistency with databases traditionally neglected but persistency tend to migrate deeper towards OS
- Security
- CPAN killer app (Perl has a single implementation = no dialect (only versions)) antinomic with "the application is a memory image syndrom".
- Foreign interface (better now because of GC, control-only language may prove interesting specially for plugins)

Autism should be finished !

Potentialities

- Many possible styles (better programmers, easier to understand XSLT or inner classes)

  Programming is like writing but on mathematical objects !

- Regular syntax (S-expressions, XML, XSLT)
- Macros (programs are data (MDA manifesto))
- Continuations (direct style vs control inversion)
- Mostly functional is provable
- Plist everywhere (Lisp2)

Future

The future will be linguistic !

- designing languages (the magic of notations)
- expressing facts, relationships, processes
- transforming (pretty-printing, evaluating, profiling, etc.)
- running
- debugging

Lispers/schemers are good at languages.

Conclusions

- Our future lies on our ability to deal efficiently with DSL
- We must target our teaching on fundamental mechanisms (with Scheme)
- Your turn now !