Lisp 50 år
Lysator 2009-09-08 / Anders Haraldsson, IDA

- Lisp historik
  - Egenskaper, konstruktioner
  - Utvecklingsmiljöer
  - Lisp maskiner
  - Standardisering
- Egen relation till Lisp
  - Utveckling av Lispsystem
  - Egen forskning
  - Undervisning
- Lisp som första programmeringsspråk
- Påverkan på andra språk, vad är fortfarande Lisp-specifikt?
- Varför blev Lisp aldrig ett riktigt programspråk?
- Infortmation från 2 Lispkonferenser under våren 2009.
- Vad händer nästa 50 år.

Egen tidig relation till Lisp

- 1968. Gick Lispkurs för Erik Sandewall i Uppsala.
- 1969. Gjorde 3-betygsarbete (C-uppsats). Ett program om släktskapsrelationer i Lisp.
- 1970. Höll min första Lispkurs, och har ännu inte hållit min sista.
- 1974. Skrev min första Lisp-kompendium / bok.
- 1970-1974, Implementering av Lisp-system
- 1971-1977, Forskning och doktorsexamen. A program manipulation sysem based on partial evaluation. (Första i datalogi och nr 14 på LiTH)

Egen tidig relation till Lisp

- 1969. Forskningsgruppen Datalogilaboratortiet skapades vid Uppsala universitet

  Lisp-utveckling och AI-forskning

  - CD 3600 (Jan Kent)
  - Siemens (Jaak Urmi, Haraldsson mfl)
  - IBM 360/370 (Jaak Urmi, Mats Andersson, Jim Goodwin mfl)
  - Fortran Lisp (Mats Nordström)
  - PL 360 / Lisp (Tore Risch)
  - QLISP (Rene Reboh, Haraldsson)
- 1976 Större delen av gruppen flyttade till Linköping där Erik Sandewall blev professor.
- Sandewall, Haraldsson, Olle Willen, Sture Hägglund, Pär Emanuelsson, Östen Oskarsson, Jerker Wilander, Mats Andersson, Jaak Urmi, Erik Tengvald, Jim Goodwin, senare Tore Risch m fl?

A historical view Lisp

- McCarthy 1959

Två huvudsakliga utvecklingslinjer:

Lisp1.5 -> BBN-Lisp -> Interlisp -> Medley (Lisp maskiner) -> Cvommon Lisp

Lisp 1.5 -> MacLisp > Scheme -> Symbolics/LMI (Lisp maskiner) -> Common Lisp

- Sandewalls Lisp experience paper, Computing Suvey 1977
  Förklarar Lisp-communitin för den mer ”konventionella” språk-värden.

John McCarthy – design considerations for Lisp 1.5 (CACM April 1960)

- *The system was designed to facilitate experiments with a proposed system called Advice Taker, whereby a machine could instructed to handle declarative as well as imperative sentences and could exhibit ”common sense” in carrying out its instructions.*
- *... be based on a scheme for representing partial recursive functions of a certain class of symbolic expressions.* (S-expressions)
- *… this formalism has advantages both as a programming language and as a vehicle for developing a theory of computation.*
- *… the universal S-function apply which plays the theoretical role of a universal Turing machine and the practical role of an interpreter.*
- *… representation of S-expressions in the memory … by list structures … and the representation of S-functions as programs*

Lisp - concepts

- partial functions – programs may not terminate
- conditional expression (p_1 -> e_1, … , p_n -> e_n)
- recursive functions

  n! = (n = 0 -> 0, T -> n * (n – 1)!)
- notation for functions and forms – lambda-expressions (Church)

  $\lambda((x, y), y^2+x)$
- S-expressions ordered pair (A . (B . C))
- a list can be represented as an S-expression

  the list (A B) is represented by (A . (B . NIL))
- functions of S-expressions in meta notation

  cons[(A . B); x]
- functions represented as S-expressions (program-data equivalence)
- the universal S-function *apply*, takes an S-function f as an S-expression and an argument list of S-expressions, and the eval-function taking a form and an association list with variables and values (expressions?).

Lisp concepts

- functions with functions as arguments (higher order functions)
- Implementation on IBM 704. Lists represents with computer word and addresses,
- … the left box of a rectangle represents the address field of the word … and the right box …. the decrement field of the word. Gave CAR (Contents of Address Register) and CDR (Contents of Decrement Register)!
- free storage list (15.000 cons-cells in IBM 704), automatic garbage collection
- Top lopp with EVALQUOTE (more or less APPLY)

```
APPEND ((A B C) (X Y)) same as
(APPEND (QUOTE (A B C)) (QUOTE (X Y)))
```

Lisp development –> BBN -> Xerox-> INTERLISP -> Medley

- Warren Teitelman, Ph D thesis 1966 introduces programming tools:

  spelling corrections

  breakpoints, advice (change interface to function, trace)

  DWIM (DO What I Mean), corrected errors, unbound atoms,
  undefined functions etc

  structure editing on memory resident list structure

  file package med “cleanup”

  CLISP (Conversational Lisp), could write

  (IF N=0 THEN 1 ELSE N*(FAC N-1))

  used DWIM (as errors, N=0 was unbound atom!)

  corrected: (IFFN=0 THENN 1 ELSE N*8FACN-1)

  Masterscope (program analysis – function calls, functions used destructive operations, etc , editing)

Lisp development –> BBN -> Xerox-> INTERLISP -> Medley (cont.)

- Implemented PDP10 (first QZ Sthlm computer center), DEC20 (used in Lkp both education and research), Xerox Lisp machines/Danderlion/Medley
- Datalogilaboratoriet implemented

  a FORTRAN-versiron of LISP (Mats Nordström), parts of INTERLISP on a small Siemens 305 (Urmi, Haraldsson)

  full INTERLISP on IBM 360 / Siemens (Jaak Urmi, Mats Andersson among others)

Lisp development –> BBN -> Xerox-> INTERLISP -> Medley (cont.)

- Language features:

  LAMBDA as now, NLAMBDA did not evaluated its argument. (Did not used macros). Use *eval* for those arguments which should be evaluated.

  An implementation of (FOR I = 1 TO N (PRINT I))

```
(NLAMBDA (VAR X START-EXPR Y END-EXPR FOR-BODY)
      (SET VAR (EVAL START-EXPR))
      … go on looping …)
```

Lisp development –> BBN -> Xerox-> INTERLISP -> Medley (cont.)

- Lexical closures, FUNARG-expressions produced by using FUNCTION. Introduces spaghetti stack. (Bobrow)
- Environment, stacks, was reachable from the language (STKPOS, RETFROM)

- Dynamic binding with “shallow binding techniques”.
- Common Lisp / Medley came with static binding. Was implemented on top of INTERLISP

- Lisp machines introduced graphical environments.
- Xerox -> Envos -> Venue (out of business)

Lisp development –> MacLisp – Lisp machines, new Lisp dialect: Scheme

- Macros for introducing special forms (mid 60)
- Errorhandling (ERRSET), CATCH och THROW (1972)
- Big numbers, arbitrary precision (1971)
- Read tables –programmable syntax
- MIT Lisp Machine project (1974) (Grenblatt, Stallman, Moon) –LMI (Lisp Machine Inc), Symbolics

Lisp Machines

- Von Neuman architecture not good for Lisp
- First ideas Deutsch (1973)

    "a singel-user minicomputer-calss machine that would be specially microcoded to run lisp and support a Lisp development environment."

  cdr-coding
- Micro coded, incremental and generational (ephemeral) garbage collection

Scheme

- MIT Sussman, Steele, 1975
- Actually implemented to understand Hewitt’s actor model. They needed continuations and introduced static binding.

  Actors and lexical closures were the same concept.
- Lambda: The ultimate Imperative (Steele 1976) demonstrated how a number of control structures could be implemented in Scheme
- Tail recursion
- Scheme -> T; compilation through source-to-source transformation to continuation-passing style (CPS)

Common Lisp

- Two directions INTERLISP and MacLisp.
- 1981 an attempt to get them together.
- 1984 Common Lisp the language.
- Development of object orientation

  flavors

  CLOS (Common Lisp Object Oriented System)

  multiple inheritance, mixins, generic functions,
  method combination, metaclasses, meta-objects,
  user customization of instance creation,
  change class etc

Main features of Lisp

- Programs as data structures
- Programs are abstract syntax trees, no parsing
- Procedures as first order objects, closures
- Extendible language, the macro facility
- Interpreted and incremental
- Style of programming

  Sandewalls (1977) *The Lisp Experience* in Computing Survey
  the first paper explaining Lisp outside the Lisp/AI- community

Concepts in programming languages – a historical view AI-languages

- Planner, QLISP
- Pattern matching
- Backtracking

- Logikprogrammering / Prolog

My own research: Concepts and methods (1971-1974) Non-determinism, continuations

- PCDB Predicate Calculus Data Base (IFIP 1974)
  - Coompilation of predicate formulas (Horn clauses) to Lisp-programs (mor or less compilatoion of logic programs (Prolog) to Lisp
- remainder procedure (today continuations) with funarg-expressions (closures)

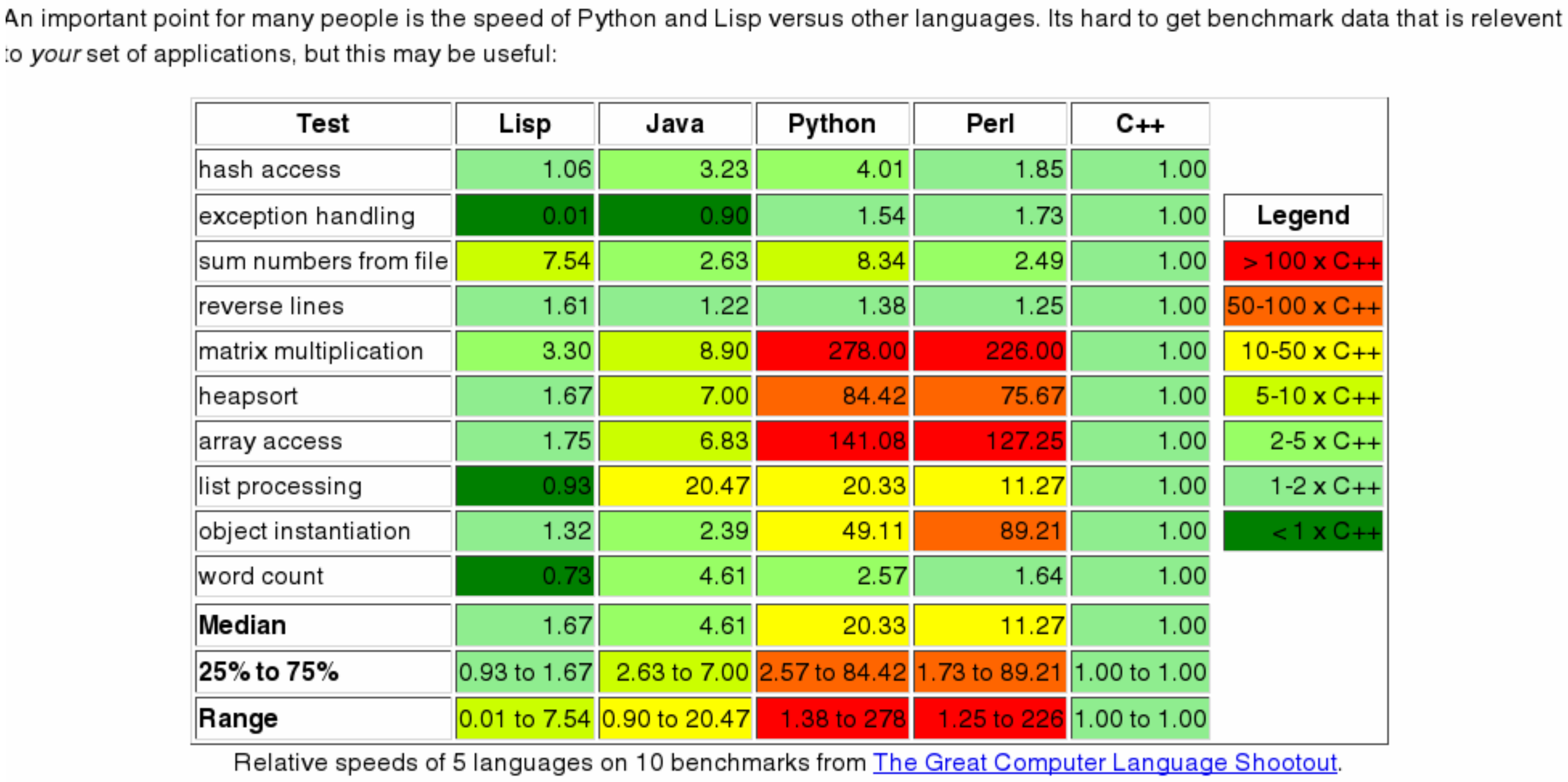
My own research: Concepts and methods Closures, partial evaluation

- Partial evaluering
- REDFUN
- Relation interpretation and compilation
- Self application (Futamuras projections)
  - Generating compiler from interpreter
  - Compiler-compilers
- State of the art today

Python

- “Basically, Python can be seen as a dialect of Lisp with "traditional" syntax (what Lisp people call "infix" or "m-lisp" syntax). Python supports all of Lisp's essential features ***except macros***, and you don't miss macros all that much because it does have eval, and operator overloading, and regular expression parsing, so you can create custom languages that way.” (http://norvig.com/python-lisp.html 2009-09-07)

- Interaktiv REPL (Read Eval Print Loop)
- Objekten är typade, ej variablerna
- Har symboler
- Inbyggda listor
- Dictionaries (nyckel-värdepar)
- Funktioner – lambdauttryck
- Closure
- Iteratorer, generatorer
- Anrop till interpretator med eval/apply/exec



An important point for many people is the speed of Python and Lisp versus other languages. Its hard to get benchmark data that is relevent to *your* set of applications, but this may be useful:

Test	Lisp	Java	Python	Perl	C++
hash access	1.06	3.23	4.01	1.85	1.00
exception handling	0.01	0.90	1.54	1.73	1.00
sum numbers from file	7.54	2.63	8.34	2.49	1.00
reverse lines	1.61	1.22	1.38	1.25	1.00
matrix multiplication	3.30	8.90	278.00	226.00	1.00
heapsort	1.67	7.00	84.42	75.67	1.00
array access	1.75	6.83	141.08	127.25	1.00
list processing	0.93	20.47	20.33	11.27	1.00
object instantiation	1.32	2.39	49.11	89.21	1.00
word count	0.73	4.61	2.57	1.64	1.00
Median	1.67	4.61	20.33	11.27	1.00
25% to 75%	0.93 to 1.67	2.63 to 7.00	2.57 to 84.42	1.73 to 89.21	1.00 to 1.00
Range	0.01 to 7.54	0.90 to 20.47	1.38 to 278	1.25 to 226	1.00 to 1.00

Legend
> 100 x C++
50-100 x C++
10-50 x C++
5-10 x C++
2-5 x C++
1-2 x C++
< 1 x C++

Relative speeds of 5 languages on 10 benchmarks from The Great Computer Language Shootout.

Ruby

- I stort som Python
- Block/kod – kan kan överföras som parametrar

C++

- Ny standard C++0x (se wikipedia)
- C++ kommer innehålla lambda-uttryck
  []\(int x, int y) {return x+y}
- Någon form av closure

Egna reflektioner

- Does syntax matter?

Kan man inte programmera/editera direkt i det abstrakta syntaxträdet, dvs strukturen.

Programstrukturen kan visualiseras på andra sätt. Textuell syntax kan vara bra både som inmatning och för läsbarhet till resp från syntaxträdet.

Medley / Lispmaskinerna hade struktureditor. Textfil skapades för lagring av program och för snygg utskrift (jfr publiceringsnotation Algol)

Egna reflektioner

- Det enda som Lisp verkar ha kvar är att programmen representeras i manipulerbar datastruktur, vilket enkel möjliggör makros, advicing och programtransformationer.
- Finns det inga andra språk med samma modell?

The macro debate

- Det pågår en debatt om användningen av makro

- Kunna skapa ett mer domänspecifikt språk
- Komplicerar förståelsen
- Hygieniska makros

Lisp som första programmeringsspråk i undervisningen

- D och C har haft Lisp sedan mitten av 80-talet.
- Y har haft Scheme sedan mitten av 90-talet.
- Modell: Börja med funktionell programmering.
- Har det vara bra? Är det fortfarande bra? Skall det vara så i framtiden?

Varför blev Lisp aldrig ett ”riktigt” programmeringsspråk?

- Före sin tid?
- Annorlunda syntaxen, parenteserna?
- Man skriver rekursiva program?
- Mappningen mot hårdvara?
- Inkrementella, dynamiska, snabbhet?

Många system innehåller någon typ av gömd Lisp, men man vågar inte riktigt tala om det.

Lisp nästa 50 år

- Det är svårt att skapa en community för ett gammalt språk. Inget problem med “nya” Lisp-dialekter (om man får kalla dessa detta) som Python och Ruby.
- Några starka Lisp-utvecklare av produkter: Allegro, LispWorks
- Common Lisp, som standard, är för klumpig. Skall man dra igång en ny standardisering?
- Det enda “nya” språket man talar om, som skulle kunna ta över är Clojure (Rick Hickey)

Lisp nästa 50 år – Clojure?

- Clojure is a dynamic programming language that targets the Java Virtual Machine. It is designed to be a general-purpose language, combining the approachability and interactive development of a scripting language with an efficient and robust infrastructure for multithreaded programming. Clojure is a compiled language - it compiles directly to JVM bytecode, yet remains completely dynamic. Every feature supported by Clojure is supported at runtime. Clojure provides easy access to the Java frameworks, with optional type hints and type inference, to ensure that calls to Java can avoid reflection.

Lisp nästa 50 år – Clojure?

- Clojure is a dialect of Lisp, and shares with Lisp the code-as-data philosophy and a powerful macro system. Clojure is predominantly a functional programming language, and features a rich set of immutable, persistent data structures. When mutable state is needed, Clojure offers a software transactional memory system and reactive Agent system that ensure clean, correct, multithreaded designs.