Topics in Computational Linguistics — Parsing and Generation —

Stephan Oepen
Dan Flickinger

Center for the Study of Language and Information
Stanford University — Spring Quarter 2003

{oe|dan}@csli.stanford.edu

http://lingo.stanford.edu/courses/03/pg/
A Few Central Concepts

Algorithm

Date: circa 1894
... a procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation; broadly: a step-by-step procedure for solving a problem or accomplishing some end especially by a computer.

(Merriam Webster)

Implement

... to give practical effect to and ensure of actual fulfillment by concrete measures. (Merriam Webster)
Common-Lisp: Syntax

- Numbers: 42, 3.1415, 1/3;
- strings: "foo", "42", "(bar)";
- symbols: pi, t, nil, foo, Fo0;
- lists: (1 2 3 4 5), (), nil,

```lisp
(defun ! (n)
  (if (= n 0)
    1
    (* n (! (- n 1))))
```

- Lisp manipulates *symbolic expressions* (known as ‘sexps’);
- sexps come in two fundamental flavours, atoms and lists;
- atoms include numbers, strings, symbols, structures, et al.;
- lists are used to represent *both* data and program code;
- rather few ‘magic’ characters: ‘(’, ‘)’, ‘:’, ‘,’ ‘;’, ‘#’, ‘|’, ‘‘’;
- all operators use *prefix* notation;
- symbol case does *not* matter.
Common-Lisp: Semantics (aka Evaluation)

- Constants (e.g. numbers and strings, `t` and `nil`) evaluate to themselves:
  
  ```lisp
  ```

- Symbols evaluate to their associated value (if any):
  
  ```lisp
  ? pi → 3.141592653589793
  ? foo → error (unless a value was assigned earlier)
  ```

- Lists are function calls; the first element is interpreted as an operator and invoked with the `values` of all remaining elements as its arguments:
  
  ```lisp
  ? (* pi (+ 2 2)) → 12.566370614359172;
  ```

- The `quote()` operator (abbreviated as `'`) suppresses evaluation:
  
  ```lisp
  ? (quote (+ 2 2)) → (+ 2 2)
  ? ’foo → foo
  ```
A Couple of List Operations

- **first()** and **rest()** destruct lists; **cons()** builds up new lists:
  
  ? (first '(1 2 3)) → 1  
  ? (rest '(1 2 3)) → (2 3)  
  ? (first (rest '(1 2 3))) → 2  
  ? (rest (rest (rest '(1 2 3)))) → nil  
  ? (cons 0 '(1 2 3))) → (0 1 2 3)  
  ? (cons 1 (cons 2 (cons 3 nil))) → (1 2 3)

- many additional list operations (derivable from the above primitives):
  
  ? (list 1 2 3) → (1 2 3)  
  ? (append '(1 2 3) '(4 5 6)) → (1 2 3 4 5 6)  
  ? (length '(1 2 3)) → 3  
  ? (reverse '(1 2 3)) → (3 2 1)
Lists: Internal Representation

\[(1 \ 2 \ 3)\]  \[(1 \ 2) \ 3\]

\[(\text{cons} \ 1 \ (\text{cons} \ 2 \ (\text{cons} \ 3 \ \text{nil})))\]  \[(\text{cons} \ (\text{cons} \ 1 \ (\text{cons} \ 2 \ \text{nil})) \ (\text{cons} \ 3 \ \text{nil}))\]

Stanford — 3-Apr-03 (oe@csli.stanford.edu)

Parsing and Generation (13)
Assigning Values — ‘Generalized Variables’

- `defparameter()` declares a ‘global variable’ and assigns a value:
  
  ? (defparameter *foo* 42) → *FOO*
  
  ? *foo* → 42

- `setf()` associates (‘assigns’) a value to a symbol (a ‘variable’):
  
  ? (setf *foo* (+ *foo* 1)) → 43
  
  ? *foo* → 43
  
  ? (setf *foo* '(1 1 3)) → (1 1 3)

- `setf()` can also alter the values associated to ‘generalized variables’:
  
  ? (setf (first (rest *foo*)) 2) → 2
  
  ? *foo* → (1 2 3)

  ? (setf (cons 0 *foo*) 2) → error
Predicates — Conditional Evaluation

- A *predicate* tests some condition and evaluates to a boolean truth value; nil indicates *false* — anything non-nil (including t) indicates *true*:

  ? (listp '(1 2 3)) → t
  ? (null (rest '(1 2 3))) → nil
  ? (or (not (numberp *foo*)) (and (>= *foo* 0) (< *foo* 42))) → t
  ? (equal (cons 1 (cons 2 (cons 3 nil))) ’(1 2 3)) → t

- conditional evaluation proceeds according to a boolean truth condition:

  ? (if (numberp *foo*)
      (+ *foo* 42)
      (first (rest *foo*)))
  → 2
More Conditional Evaluation

- if() is fairly limited: exactly one sexp in its then and else branches:
  \[(\text{if test sexp sexp})\]

- when() and unless() generalize then and else branches, respectively:
  \[(\text{when test sexp ... sexp})\]
  \[(\text{unless test sexp ... sexp})\]

- cond() allows an arbitrary number of conditions and associated sexps:
  \[(\text{cond (test}_1\text{ sexp ... sexp)}\]
  \[.\]
  \[.\]
  \[(\text{test}_n\text{ sexp ... sexp})\]
  \[(t\text{ sexp ... sexp})\]
Defining New Functions

- `defun()` associates a function definition (`'body'`) with a symbol:

  ```lisp
  (defun name (parameter₁ ... parameterₙ) body)
  ```

- `(defun ! (n)  
  (if (= n 0)  
    1  
    (* n (! (- n 1)))))))

  → !

- `? (! 0) → 1`

- `? (! 5) → 120`

- when a function is called, actual arguments (e.g. ‘0’ and ‘5’) are bound to the function parameter(s) (i.e. ‘n’) for the scope of the function body;
- functions evaluate to the value of the last sexp in the function body.
Recursion as a Control Structure

- A function is said to be *recursive* when its *body* contains a call to itself:

  (defun mlength (list)
   (if (null list)
       0
       (+ 1 (mlength (rest list)))))

- ? (mlength '(a b)))
  0: (MLENGTH (A B))
  1: (MLENGTH (B))
  2: (MLENGTH NIL)
  2: returned 0
  1: returned 1
  0: returned 2
  → 2

- *body* contains (at least) one recursive and one non-recursive branch.
Suggested Homework

• Read through Chapters One to Five of [Winston & Horn, 1989]; have a look at Chapters Six, Eleven, and Thirteen if you feel like it.

• try running the LKB in Sweet Hall (see the course site for instructions); use the *common-lisp* buffer in emacs to evaluate some Lisp examples;

• send us an email with a few pieces of information on yourself:

From: Angelika Mustermann (mustermann@stanford.edu)
To: dan@csli.stanford.edu, oe@csli.stanford.edu
Subject: SSP 239a: Parsing and Generation

hi dan and stephan,

i’m angelika (see email above) from your SSP class this quarter. i’m a Symbolic Systems major with little programming experience (only Perl and Prolog, sadly). i have had some exposure to unification grammars and, primarily, take this class to learn how formal grammars get used in practise. i expect to pursue a career in natural language processing, so more programming experience should be handy. anytime thursday afternoon after 1:00 works for me; friday and wednesday are bad.

cheers - geli