Programming Languages and Translators

COMS W4115

Pieter Bruegel, The Tower of Babel, 1563

Prof. Stephen A. Edwards
Fall 2007
Columbia University
Department of Computer Science

Instructor

Prof. Stephen A. Edwards
sedwards@cs.columbia.edu
http://www1.cs.columbia.edu/~sedwards/
462 Computer Science Building
Office Hours: 4–5 PM Tuesday, 4–5 PM Wednesday

Schedule

Tuesdays and Thursdays, 2:40 PM to 3:55 PM
535 Mudd
Lectures: September 4 to December 6
Midterm: October 25
Final: December 6 (in-class)
Final project report: December 18
Holidays: November 6 (Election day), November 22 (Thanksgiving)

Objectives

Theory of language design
- Finer points of languages
- Different languages and paradigms
Practice of Compiler Construction
- Overall structure of a compiler
- Automated tools and their use
- Lexical analysis to assembly generation

Required Text

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman.

Assignments and Grading

40% Programming Project
20% Midterm (near middle of term)
30% Final (at end of term)
10% Individual homework

Project is most important, but most students do well on it. Grades for tests often vary more.

Prerequisite: Java Fluency

You and your group will write perhaps 5000 lines of Java; you will not have time to learn it.
We will be using a tool that generates fairly complicated Java and it will be necessary to understand the output.

Prerequisite: COMS W3157 Advanced Programming

Teams will build a large software system
Makefiles, version control, test suites
Testing will be as important as development

Prerequisite: COMS W3261 Computability and Models of Computation

You need to understand grammars
We will be working with regular and context-free languages
Class Website
Contains syllabus, lecture notes, and assignments.
Schedule will be continually updated during the semester.

Collaboration
Collaborate with your team on the project.
Exception: CVN students do the project by themselves.
Do your homework by yourself.
Tests: Will be closed book with a one-page “cheat sheet”
of your own devising.
Don’t cheat on assignments: If you’re dumb enough to
cheat, I’m smart enough to catch you.

The Project
Design and implement your own little language.
Five deliverables:
1. A proposal describing and motivating your language
2. A language reference manual defining it formally
3. A compiler or interpreter for your language running on
some sample programs
4. A final project report
5. A final project presentation

Teams
Immediately start forming four-person teams to work on
this project.
Each team will develop its own language.
Suggested division of labor: Front-end, back-end, testing,
documentation.
All members of the team should be familiar with the whole
project.
Exception: CVN students do the project by themselves.

First Three Tasks
1. Decide who you will work with
   You’ll be stuck with them for the term; choose wisely.
2. Elect a team leader
   Languages come out better from dictatorships, not
democracies. Besides, you’ll have someone to blame.
3. Select a weekly meeting time
   Harder than you might think. Might want to discuss
   with a TA you’d like to have so it is convenient for
   him/her as well.

Project Proposal
Describe the language that you plan to implement.
Explain what problem your language can solve and how it
should be used. Describe an interesting, representative
program in your language.
Give some examples of its syntax and an explanation of
what it does.
2–4 pages

Language Reference Manual
A careful definition of the syntax and semantics of your
language.
Follow the style of the C language reference manual
(Appendix A of Kernighan and Ritchie, The C
Programming Language; see the class website).

Final Report Sections
1. Introduction: the proposal
2. Language Tutorial
4. Project Plan
5. Architectural Design
6. Test Plan
7. Lessons Learned
8. Complete listing
Due Dates
Proposal September 25
Reference Manual October 18
Final Report December 18

Design a language?
A small, domain-specific language.
Think of awk or php, not Java or C++.
Examples from earlier terms:
Quantum computing language
Geometric figure drawing language
Projectile motion simulation language
Matlab-like array manipulation language
Screenplay animation language

Other language ideas
Simple animation language
Model train simulation language
Escher-like pattern generator
Music manipulation language (harmony)
Web surfing language
Mathematical function manipulator
Simple scripting language (à la Tcl)
Petri net simulation language

Components of a language: Syntax
How characters combine to form words, sentences, paragraphs.

```
The quick brown fox jumps over the lazy dog.
```

is syntactically correct English, but isn’t a Java program.

```
class Foo {
   public int j;
   public int foo(int k) { return j + k; }
}
```

Is syntactically correct Java, but isn’t C.

Components of a language: Semantics
What a well-formed program “means.”
The semantics of C says this computes the \( n \)th Fibonacci number.

```
int fib(int n)
{
    int a = 0, b = 1;
    for (i = 1; i < n; i++)
    {
        int c = a + b;
        a = b;
        b = c;
    }
    return b;
}
```

Specifying Syntax
Usually done with a context-free grammar.
Typical syntax for algebraic expressions:

```
expr  →  expr + expr
   |  expr - expr
   |  expr * expr
   |  expr/ expr
   |  digit
   |  ( expr )
```

Semantics
Something may be syntactically correct but semantically nonsensical.

```
int fib(int n)
{
    int a = 0, b = 1;
    for (i = 1; i < n; i++)
    {
        int c = a + b;
        a = b;
        b = c;
    }
    return b;
}
```

The rock jumped through the hairy planet.

Or ambiguous

```
int fib(int n)
{
    int a = 0, b = 1;
    for (i = 1; i < n; i++)
    {
        int c = a + b;
        a = b;
        b = c;
    }
    return b;
}
```

The chickens are ready for eating.

```
class Foo {
   int bar(int x) { return Foo; }
}
```

Nonsensical in Java:

```
class Foo {
   public int foo(int) { return j + k; }
}
```

```
class Bar {
   public float foo() { return 0; }
   public int foo() { return 0; }
}
```

Ambiguous in Java:
Specifying Semantics

Doing it formally beyond the scope of this class, but basically two ways:

- **Operational semantics**
  Define a virtual machine and how executing the program evolves the state of the virtual machine

- **Denotational semantics**
  Shows how to build the function representing the behavior of the program (i.e., a transformation of inputs to outputs) from statements in the language.

Most language definitions use an informal operational semantics written in English.

Great Moments in Programming Language Evolution

<table>
<thead>
<tr>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before: numbers</td>
</tr>
<tr>
<td>55</td>
</tr>
<tr>
<td>8985</td>
</tr>
<tr>
<td>884508</td>
</tr>
<tr>
<td>8850C</td>
</tr>
<tr>
<td>390D</td>
</tr>
<tr>
<td>740D</td>
</tr>
<tr>
<td>390D</td>
</tr>
<tr>
<td>780B</td>
</tr>
<tr>
<td>290D</td>
</tr>
<tr>
<td>390D</td>
</tr>
<tr>
<td>75F6</td>
</tr>
<tr>
<td>C9</td>
</tr>
<tr>
<td>C3</td>
</tr>
<tr>
<td>29C2</td>
</tr>
<tr>
<td>EBF6</td>
</tr>
</tbody>
</table>

**FORTRAN**

Before: numbers

```fortran
gcd: pushl %ebp
movl %esp, %ebp
movl %ebp, %eax
cmpl %edx, %eax
je .L9
jle .L7
subl %edx, %eax
.L2: cmpl %edx, %eax
jne .L7
.L9: leave
ret
```

After: Symbols

```fortran
10 if (a .EQ. b) goto 20
if (a .LT. b) then
  a = a - b
else
  b = b - a
endif
goto 10
```

**COBOL**

**LISP, Scheme, Common LISP**

**APL**

Powerful operators, interactive language

**(defun gnome-doc-insert ()
  "Add a documentation header to the current function.
  Only C/C++ function types are properly supported currently.
  (interactive)
  (let (c-insert-here (point))
    (save-excursion
      (beginning-of-defun)
      (let (c-arglist
        c-funcname
c-point (point))
        c-comment-point
c-isvoid
c-doinsert)
      (search-backward "°"
        (forward-line -2)
        (while (or (looking-at "°")
          (looking-at "° * ")
          (looking-at "°#"))
        (forward-line 1))
```}

**Algol, Pascal, Clu, Modula, Ada**

Imperative, block-structured language, formal syntax
definition, structured programming

```algol
PROC insert = (INT e, REF TREE t)VOID:
  IF TREE(t) IS NIL THEN t := HEAP NODE := (e, TREE(NIL), TREE(NIL))
  ELIF e < e OF t THEN insert(e, l OF t)
  ELIF e > e OF t THEN insert(e, r OF t)
  FI;
```

**SNOBOL, Icon**

String-processing languages
**BASIC**

Programming for the masses

```
10 PRINT "GUESS A NUMBER BETWEEN ONE AND TEN"
20 INPUT A$
30 IF A$ = "5" THEN PRINT "GOOD JOB, YOU GUessed IT"
40 IF A$ = "5" GOTO 100
50 PRINT "YOU ARE WRONG. TRY AGAIN"
60 GOTO 10
100 END
```

**Simula, Smalltalk, C++, Java, C#**

The object-oriented philosophy

```
class Shape(x, y); integer x; integer y;
virtual: procedure draw;
begin
  comment -- get the x & y coordinates --;
  integer procedure getX;
  getX := x;
  comment -- set the x & y coordinates --;
  integer procedure setX(newx);
  x := newx;
end Shape;
```

**C**

Efficiency for systems programming

```
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

**ML, Miranda, Haskell**

Purer functional language

```
structure NewStack = struct
  type 'a stack = 'a list
  exception Empty
  val empty = []
  fun isEmpty (s:'a stack):bool =
    (case s
      of [] => true
      | _ => false)
  fun top (s:'a stack): =
    (case s
      of [] => raise Empty
      | x::xs => x)
  fun pop (s:'a stack):'a stack =
    (case s
      of [] => raise Empty
      | x::xs => xs)
  fun push (s:'a stack, x: 'a):'a stack = x::s
  fun rev (s:'a stack):'a stack = rev (s)
end
```

**sh, awk, perl, tcl, python**

Scripting languages: glue for binding the universe together

```
class() {
  classname='echo "$1" | sed -n '1 s/ * :. * $//p''
pARENT='echo "$1" | sed -n '1 s/ * : * //p''
hppbody='echo "$1" | sed -n '2,$p''
forwarddefs="$forwarddefs
class $classname;"
if (echo $hppbody | grep -q "$classname()"); then
  defaultconstructor=
else
  defaultconstructor="$classname() {}"
fi
}
```

**VisiCalc, Lotus 1-2-3, Excel**

The spreadsheet style of programming

```
create table shirt
  id smallint unsigned not null auto_increment,
  style enum('t-shirt', 'polo', 'dress') not null,
  color enum('red', 'blue', 'white', 'black') not null,
  owner smallint unsigned not null
  references person(id),
  primary key (id)
); 
insert into shirt values
  (null, 'polo', 'blue', last_insert_id()),
  (null, 'dress', 'white', last_insert_id()),
  (null, 't-shirt', 'blue', last_insert_id());
```

**Prolog**

Logic Language

```
edge(a, b), edge(b, c).
edge(c, d), edge(d, e), edge(e, f).
path(X, X).
pth(X, Y) :-
  edge(X, Z), path(Z, Y).
```