Programming Languages and Translators

COMS W4115

Pieter Bruegel, *The Tower of Babel*, 1563

Prof. Stephen A. Edwards
Fall 2004
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: 2–3 PM Tuesday, Thursday
Schedule

Tuesdays and Thursdays, 11:00 AM to 12:15 PM
Room 535, Seeley W. Mudd
Lectures: September 7 to December 9
Midterm: November 9
Final: December 9
Final project report: December 21
Holidays: November 2 (Election day), November 25 (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, Ravi Sethi, and Jeffrey D. Ullman.
*Compilers: Principles, Techniques, and Tools.*
Addison-Wesley, 1985.
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

Off my home page,
http://www1.cs.columbia.edu/~sedwards/

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.

Homework is to be done by yourself.

The Project
The Project

Design and implement your own little language.

Five deliverables:

1. A white paper 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.
White Paper

Follow the style of the Java white paper (see the class website for a link), but tone down the marketing hype. 4–8 pages.

Answer the question, “why another language?” with a description of what problem your language solves and how it should be used.

Small snippets of code to show syntax is enough.
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 white paper
2. Language Tutorial
4. Project Plan
5. Architectural Design
6. Test Plan
7. Lessons Learned
8. Complete listing
Due Dates

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>White Paper</td>
<td>September 28 soon</td>
</tr>
<tr>
<td>Reference Manual</td>
<td>October 21</td>
</tr>
<tr>
<td>Final Report</td>
<td>December 21</td>
</tr>
</tbody>
</table>
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 lá Tcl)
Petri net simulation language
What’s in a 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.

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

Is syntactically correct Java, but isn’t C.
Specifying Syntax

Usually done with a context-free grammar.

Typical syntax for algebraic expressions:

\[
expr \quad \rightarrow \quad expr + expr \\
| \quad expr - expr \\
| \quad expr \times expr \\
| \quad expr/expr \\
| \quad digit \\
| \quad (expr)
\]
Components of a language: Semantics

What a well-formed program “means.”

The semantics of C says this computes the $n$th Fibonacci number.

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

Something may be syntactically correct but semantically nonsensical.

The rock jumped through the hairy planet.

Or ambiguous

The chickens are ready for eating.
Semantics

Nonsensical in Java:

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

Ambiguous in Java:

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

<table>
<thead>
<tr>
<th>Before: numbers</th>
<th>After: Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td><code>gcd: pushl %ebp</code></td>
</tr>
<tr>
<td>89E5</td>
<td><code>movl %esp, %ebp</code></td>
</tr>
<tr>
<td>8B4508</td>
<td><code>movl 8(%ebp), %eax</code></td>
</tr>
<tr>
<td>8B550C</td>
<td><code>movl 12(%ebp), %edx</code></td>
</tr>
<tr>
<td>39D0</td>
<td><code>cmpl %edx, %eax</code></td>
</tr>
<tr>
<td>740D</td>
<td><code>je .L9</code></td>
</tr>
<tr>
<td>39D0</td>
<td><code>.L7: cmpl %edx, %eax</code></td>
</tr>
<tr>
<td>7E08</td>
<td><code>jle .L5</code></td>
</tr>
<tr>
<td>29D0</td>
<td><code>subl %edx, %eax</code></td>
</tr>
<tr>
<td>39D0</td>
<td><code>.L2: cmpl %edx, %eax</code></td>
</tr>
<tr>
<td>75F6</td>
<td><code>jne .L7</code></td>
</tr>
<tr>
<td>C9</td>
<td><code>.L9: leave</code></td>
</tr>
<tr>
<td>C3</td>
<td><code>ret</code></td>
</tr>
<tr>
<td>29C2</td>
<td><code>.L5: subl %eax, %edx</code></td>
</tr>
<tr>
<td>EBF6</td>
<td><code>jmp .L2</code></td>
</tr>
</tbody>
</table>
FORTRAN

Before
gcd:  pushl  %ebp
     movl  %esp,  %ebp
     movl  8(%ebp),  %eax
     movl  12(%ebp),  %edx
     cmpl  %edx,  %eax
     je    .L9
     .L7:  cmpl  %edx,  %eax
           jle   .L5
           subl  %edx,  %eax
     .L2:  cmpl  %edx,  %eax
           jne  .L7
     .L9:  leave
           ret
     .L5:  subl  %eax,  %edx
           jmp   .L2

After: Expressions, control-flow

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

20   end
COBOL

Added type declarations, record types, file manipulation

data division.
file section.
*    describe the input file
fd employee-file-in
    label records standard
    block contains 5 records
    record contains 31 characters
    data record is employee-record-in.

01    employee-record-in.
02    employee-name-in    pic x(20).
02    employee-rate-in    pic 9(3)v99.
02    employee-hours-in   pic 9(3)v99.
02    line-feed-in        pic x(1).
LISP, Scheme, Common LISP

Functional, high-level languages

(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 "\
                 (looking-at "\#")
      (forward-line 1))

)
APL

Powerful operators, interactive language

[0] Z←GAUSSRAND N;B;F;M;P;Q;R
[1] ♠ Returns ω random numbers having a Gaussian normal distribution
[2] ♠ (with mean 0 and variance 1) Uses the Box-Muller method.
[4] ♠
[5] Z←0
[7] L1:Q←N−PZ ♠ how many more we need
[8] →(Q≤0)/L2 ♠ quit if none
[9] Q←1.3×Q÷2 ♠ approx num points needed
[10] P←−1+(2÷M−1)×−1+?(Q,2)P ♠ random points in −1 to 1 square
[11] R+/+P÷P ♠ distance from origin squared
[12] B+(R≠0)∧R<1
[13] R+/+B/R ♠ points within unit circle
[14] F+((−2×(ΦR)+R)×.5
[16] →L1
[17] L2:Z←N+Z
[18] ♠ ArchDate: 12/16/1997 16:20:23.170

Algol, Pascal, Clu, Modula, Ada

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

PROC insert = (INT e, REF TREE t)VOID:
    # NB inserts in t as a side effect #
    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;

PROC trav = (INT switch, TREE t, SCANNER continue, alternative)VOID:
    # traverse the root node and right sub-tree of t only. #
    IF t IS NIL THEN continue(switch, alternative)
    ELIF e OF t <= switch THEN
        print(e OF t);
        traverse(switch, r OF t, continue, alternative)
    ELSE # e OF t > switch #
        PROC defer = (INT sw, SCANNER alt)VOID:
            trav(sw, t, continue, alt);
            alternative(e OF t, defer)
    FI;

Algol-68, source http://www.csse.monash.edu.au/~lloyd/tildeProgLang/Algol68/treemerge.a68
SNOBOL, Icon

String-processing languages

LETTER = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ$#@'
SP.CH = "+-=.*()'/&"
SCOTA = SP.CH
SCOTA ' & ' =
Q = ""'
QLIT = Q FENCE BREAK(Q) Q
ELEM = QLIT | 'L' Q | ANY(SCOTA) | BREAK(SCOTA) | REM
F3 = ARBNO(ELEM FENCE)
B = (SPAN(' ') | RPOS(0)) FENCE
F1 = BREAK(' ') | REM
F2 = F1
CAOP = ('LCL' | 'SET') ANY('ABC') |
+ 'AIF' | 'AGO' | 'ACTR' | 'ANOP'
ATTR = ANY('TLSIKN')
ELEM C = '(' FENCE *F3C ') ' | ATTR Q | ELEM
F3C = ARBNO(ELEMC FENCE)
ASM360 = F1 . NAME B
+ ( CAOP . OPERATION B F3C . OPERAND |
+ F2 . OPERATION B F3 . OPERAND)
+ B REM . COMMENT

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
The object-oriented philosophy

class Shape(x, y); integer x; integer y;
virtual: procedure draw;
begin
    comment -- get the x & y components for the object --;
    integer procedure getX;
    getX := x;
    integer procedure getY;
    getY := y;

    comment -- set the x & y coordinates for the object --;
    integer procedure setX(newx); integer newx;
    x := newx;
    integer procedure setY(newy); integer newy;
    y := newy;
end Shape;
C

Efficiency for systems programming

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

```bash
class() {
    classname=`echo "$1" | sed -n '1 s/ *:.*$//p'`
    parent=`echo "$1" | sed -n '1 s/Ã.*: */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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hours</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>Wage per hour</td>
<td>$ 5.36</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Total Pay</td>
<td>= B1 * B2</td>
</tr>
</tbody>
</table>
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

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