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.
Addison-Wesley, 1985.

Assignments and Grading

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

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
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**

- White Paper: September 28
- Reference Manual: October 21
- Final Report: December 21

---

**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.
  ```java
  class Foo {
    public int j;
    public int foo(int k) { return j + k; }
  }
  ```
- Is syntactically correct Java, but isn’t C.

**Semantics**

What a well-formed program “means.”

- The semantics of C says this computes the $n$th Fibonacci number.
  ```java
  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;
  }
  ```
- Something may be syntactically correct but semantically nonsensical.
  - The rock jumped through the hairy planet.
- Or ambiguous
  - The chickens are ready for eating.

**Specifying Syntax**

Usually done with a context-free grammar.

Typical syntax for algebraic expressions:

- $\text{expr} \rightarrow \text{expr} + \text{expr}$
- $\text{expr} \rightarrow \text{expr} - \text{expr}$
- $\text{expr} \rightarrow \text{expr} * \text{expr}$
- $\text{expr} \rightarrow \text{digit}$
- $\text{expr} \rightarrow (\text{expr})$

---

**Semantics**

Nonsensical in Java:

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

Ambiguous in Java:

```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

Before: numbers  
After: Symbols

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>gbd</td>
</tr>
<tr>
<td>89E5</td>
<td>movl</td>
</tr>
<tr>
<td>884D</td>
<td>je</td>
</tr>
<tr>
<td>3900</td>
<td>cmpq</td>
</tr>
<tr>
<td>7E08</td>
<td>jle</td>
</tr>
<tr>
<td>2900</td>
<td>subq</td>
</tr>
<tr>
<td>1556</td>
<td>le</td>
</tr>
<tr>
<td>C9</td>
<td>leave</td>
</tr>
<tr>
<td>C3</td>
<td>ret</td>
</tr>
<tr>
<td>29C2</td>
<td>l.t.</td>
</tr>
<tr>
<td>EBPF</td>
<td>jmp</td>
</tr>
</tbody>
</table>

FORTRAN

**Before**  
**After:** Expressions, control-flow

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

**COBOL**

**Before**  
**After:** Expressions, control-flow

```
01 employee-file-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).

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-point (point))
                 (c-comment-point (c-isvoid)
                                  (c-doinsert))
                 (search-backward "\*"))
                (forward-line 1))
            (search-backward ":\*"
                            (looking-at ":\*"))
            (looking-at "\*\*"))
            (looking-at "\*\*"))
            (forward-line 1))
```

APL

**Powerful operators, interactive language**

```
{D-GUARDIANW}=NIP(FORK)
{D-AMB]B}R=RA(PSIONS HAVING A GAUSSIAN NORMAL DISTRIBUTION
{D-AMB]=R(A(A)AND) R(AND) HAVING A GAUSSIAN NORMAL DISTRIBUTION
{D-AMBR}A P(A(A)AND) R(AND) HAVING A GAUSSIAN NORMAL DISTRIBUTION
{D-GO}A  
{D-LONG}=LARGEST_INTEGER
{D-LIP}=LIP/C
{D-LIP}E=HÖW MANY WORDS WE NEED
{D-UNISO1}O=QUICKER THAN WE NEED
{D-UNISO2}O=QUICKER THAN WE NEED
{D-UNISO3}O=QUICKER THAN WE NEED
{D-UNISO4}O=QUICKER THAN WE NEED
{D-UNISO5}O=QUICKER THAN WE NEED
{D-UNISO6}O=QUICKER THAN WE NEED
{D-UNISO7}O=QUICKER THAN WE NEED
{D-UNISO8}O=QUICKER THAN WE NEED
{D-UNISO9}O=QUICKER THAN WE NEED
{D-UNISO10}O=QUICKER THAN WE NEED
{D-UNISO11}O=QUICKER THAN WE NEED
{D-UNISO12}O=QUICKER THAN WE NEED
{D-UNISO13}O=QUICKER THAN WE NEED
{D-UNISO14}O=QUICKER THAN WE NEED
{D-UNISO15}O=QUICKER THAN WE NEED
{D-UNISO16}O=QUICKER THAN WE NEED
{D-UNISO17}O=QUICKER THAN WE NEED
{D-UNISO18}O=QUICKER THAN WE NEED
{D-UNISO19}O=QUICKER THAN WE NEED
{D-UNISO20}O=QUICKER THAN WE NEED
```

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:
    # traverse( t, continue, alternate) #
    print(e OF t);
    defer(switch, continue, alt);
  FI;
```

SNOBOL, Icon

**String-processing languages**

```
LETTER = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ$#@'
SP.CH = "+-,=.*()'/& 
SCOTA = SP.CH
SCOTA &' =
Q = "'
QLIT = Q FENCE BREAK(Q) Q
ELSE = ELIP = "" | "" | ANY(SCOTA) | BREAK(SCOTA) | REM
B = (BREAK("")) | BREAK(|)
F1 = BREAK("")) | REM
F2 = FI
CASE = ("LCL" | "SEP") ANY("ABC")
ARGP = ANY("(E"
DREM = ANY("(F"
F3C = BREM(ELEM FENCE)
```

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 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

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

VisiCalc, Lotus 1-2-3, Excel

The spreadsheet style of programming

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>Hours</td>
</tr>
<tr>
<td>2</td>
<td>Wage per hour</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Total Pay</td>
</tr>
</tbody>
</table>

SQL

Database queries

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(b, e). edge(d, f).
path(X, X).
polya(X, Y) :-
    edge(X, Z), path(Z, Y).