Little Languages

and other programming paradigms

What is a language?

- "...a set of conventions for communicating an algorithm." - Horowitz
- But why just algorithms?
- HTML = hypertext markup language
- Tells browser what to do, but not exactly an algorithm
- In fact, browser has considerable smarts & retains considerable freedom.
- HTML is more like specifying input data
  - to a generic webpage layout algorithm
  - to validators, style checkers, reformatters ...
  - to search engines and machine translation systems

LilyPond (www.lilypond.org)

c4
<<c4 d4 e4>>
{ f4 <<c4 d4 e4>> }
<< g2 \ { f4 <<c4 d4 e4>> } >>
c C F C
twin kle twin kle little star

LilyPond (www.lilypond.org)

Screech and boink
Random complex notation
Han-Wen Nienhuys

dot (www.graphviz.org)

- Implemented in combo of C++, Scheme, LaTeX
- So it is built on top of another little language ...
- Which is itself built on top of TeX
  - (an extensible little language: you can define new commands)
- Which is itself built in "literate Pascal" ...
- Lilypond reminds me of MS BASIC:
  - play "c1.d8"
  - Much better than TRS-80: beep 278,12; beep 295, 4
  - More generally, play a$ where a$ is any string var – your program could build a$ at runtime!
- Other great thing about MS BASIC: draw "u183d1013"
dot (www.graphviz.org)

```plaintext
digraph g {
  graph [rankdir = "LR"];
  node [fontsize = "16" shape = "ellipse"];
  edge [];
  "node0" [shape = "record" label = "<f0> 0x10ba8 | <f1> | <f2> | -1"];
  "node1" [shape = "record" label = "<f0> 0xf7fc4380 | <f1> | <f2> | -1"];
  "node2" [shape = "record" label = "<f0> 0xf7fc4380 | <f1> | <f2> | -1"];
  "node3" [shape = "record" label = "<f0> 0xf7fc4380 | <f1> | <f2> | -1"];
  "node0":f0 -> "node1":f0 [id = 0];
  "node0":f1 -> "node2":f0 [id = 1];
  "node1":f0 -> "node3":f0 [id = 2];
}
```

What’s the hard part? Making a nice layout! Actually, it’s NP-hard…

nodes
edges

dot (www.graphviz.org)

```
digraph G {Hello->World}
```

Proof that it’s really a language:

```
digraph G {Hello->World}
```

Running the compiler from the Unix shell (another language!)

```
echo "digraph G {Hello->World}" | dot -Tpng >hello.png
```

A little language for fractal cube graphics (embedded into Haskell)

((greenC .|. redC) .-. blueC) ./. whiteC

A little language for fractal cube graphics (embedded into Haskell)

```
rcube 0 = Cache "rcube0" $ shape white Box{}
rcube n = Cache ("rcube"++(show n)) $ (s1 ./. s2) ./. (s2 ./. s1)
  where
  s2 = (s12 .-. invisible) .-. (invisible .-. s11)
  s1 = (s12 .-. s11) .-. (s11 .-. s12)
  s11 = (white .|. invisible) .|. (invisible .|. white)
  s12 = (white .|. white) .|. (white .|. white)
  white = rcube (n-1)
```

Needs recursion

slide thanks to Claus Reinke (modified)

Compiles into VRML (Virtual Reality Modeling Language)
Logo: A little(?) language for little people

- Created by Seymour Papert in 1968
  - Papert was first to see how computers could change learning
  - Had worked with the great Jean Piaget, studying children's minds
  - (Also, with Marvin Minsky, founded the MIT AI Lab and invented the first neural networks)

- Logo – a dialect of LISP
  - Fewer parentheses
  - Focus on graphics
  - Physical metaphor – robot turtles; kids could pretend to be turtles
  - Easy for kids to get started programming

Turtle talk (controlling a cursor with position, orientation, and drawing pen):
- forward d, backward d
- turnright a, turnleft a

Control structures:
- repeat n cmds, ifelse c cmds cmds
- to proctype params cmds, proctype

Little languages: More examples (quick survey)

More little (or not so little) languages
(Do these describe algorithms or data?)

- The "units" program
  - You have: (1e-14 lightyears + 100 feet) / s
  - You want: furlongs per half fortnight
  - Answer: 376067.02 (other calculators are similar …)

- Regular expressions: pattern matching
  - b(c|de)*f – does it match bdedecf? overlap with (bd)*ef?

- Makefiles: running commands under certain conditions
  - Automatically determines order to run them (with parallelization)

- Lex and yacc: specify the format of another language!
  - Compiles into code for tokenizing and parsing that language

- Awk: process each line of a structured file
  - $2==$3 { sum += $0; print $0, sum }

Protocols

- Programming languages are mainly used to deliver monologues
- But sometimes you talk to an application …
  - … and it talks back! Also in a structured language.
  - Compiler error messages? Not a great example.

- There are a lot of text-based protocols
- HTTP is one (and FTP before it)
  - You say to cs.jhu.edu: GET /holygrail HTTP/1.0
  - cs.jhu.edu replies: 404 Not Found
Conversing with the sendmail daemon

```
220 blaze.cs.jhu.edu ESMTP Sendmail 8.12.9/8.12.9; Tue, 31 Jan 2006 11:06:02 -0500 (EST)

helo emu.cs.jhu.edu
250 blaze.cs.jhu.edu Hello emu.cs.jhu.edu [128.220.13.179], pleased to meet you
expn cs325-staff
250-2.1.5 Jason Eisner <jason@...>
250 2.1.5 Jason Smith <jrs026@...>
qup
221 2.0.0 blaze.cs.jhu.edu closing connection
Connection closed by foreign host.
```
What users are like (even techies!)

“Some people find it hard to understand why you can't simply add more and more graphical notation to a visual language. For example, there have been many cases of people proposing (in private communication) all kinds and extensions to the language of statecharts. These people could not understand why you can't just add a new kind of arrow that "means synchronization", or a new kind of box that "means separate-thread concurrency". ... It seemed to them that if you have boxes and lines and they mean things, you can add more and just say in a few words what they are intended to mean.

A good example of how difficult such additions can really be is the idea of having overlapping states in statecharts... [It] took a lot of hard work to figure out a consistent syntax and semantics for such an extension. In fact, the result turned out to be too complex to justify implementation.

Nevertheless, people often ask why we don't allow overlapping... It is very hard to convince them that it is not at all simple. One person kept asking this: "Why don't you just tell your system not to give me an error message when I draw these overlapping boxes?", as though the only thing that needs to be done is to remove the error message and you are in business!

Some general-purpose languages also support extension well enough

Don't add to your language – keep it simple

Some general-purpose languages also support extension well enough

Implementing your language (if not embedded)

How will the machine understand your language?

Interpreter

Examples

Interactive command-and-control languages:

Linux shell, scripting languages,

Query languages: SQL, Prolog, ...

Client-server protocols: HTTP, Dynagraph ("incfaced", ...)
Implementing your language (if not embedded)

How will the machine understand your language?

- **Interpreter**
  - Translates your entire program into a lower-level language
  - Can look at the whole program to understand line 8
  - Can rearrange or combine multiple lines for efficiency
  - Only has to translate it once
- **Compiler**
  - Translates your entire program into a lower-level language
  - Can look at the whole program to understand line 8
  - Can rearrange or combine multiple lines for efficiency
  - Only has to translate it once

  - Examples:
    - `g++` compiles C++ into machine code, which is then interpreted by the chip
    - `javac` compiles Java into Java bytecode, which is then interpreted by the Java Virtual Machine
    - `dynac` compiles Dyna into C++, which is then compiled by `g++`

Pieces of a compiler:

- **front-end**
  - program as text
  - scanner
  - token stream

- **back-end**
  - parse tree
  - symbol table
  - intermediate code generator
  - optimiser
  - optimised code
Languages to help you build languages

A typical compiler pipeline:
- scanning (lexical analysis)
- parsing (syntax analysis)
- static analysis (types, scopes, ...)
- optimisation
- code generation
- followed by a runtime system
- code execution
- memory management, ...

Other ways to classify languages
- Declarative vs. procedural
- High-level vs. low-level (sort of the same thing)
- Domain-specific vs. general purpose

Imperative vs. object-oriented vs. functional vs. logic
- Ask me, or take 600.426 Programming Languages

First-generation through sixth-generation
- Browse web to learn history of programming languages

<table>
<thead>
<tr>
<th>Era</th>
<th>Machine languages</th>
<th>4th</th>
<th>Application languages (4GLs)</th>
<th>5th</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td></td>
<td>3rd</td>
<td>AI techniques, inference</td>
<td>6th</td>
</tr>
<tr>
<td>2nd</td>
<td>Assembly languages</td>
<td></td>
<td>Neural networks (?), others</td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td>Procedural languages</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Oh yeah ...
- So, Prof. Eisner, what are declarative methods??
  - A declarative program states only what is to be achieved
  - A procedural program describes explicitly how to achieve it

Sorting in a declarative language
- “Given array X, find an array Y such that
  - (1) Y is a permutation of X
  - (2) Y’s elements are in increasing order”
- Compiler is free to invent any sorting algorithm! (Hard?)
- You should be aware of when compiler will be efficient/inefficient

Sorting in a procedural language
- “Given array X, run through it from start to finish, swapping adjacent elements that are out of order…”
- Longer and probably buggier
- Never mentions conditions (1) and (2), except in comments