Lecture 12

Ideas for the Final Project

DSLs in real world; language extensions

Ras Bodik
Shaon Barman
Thibaud Hottelier

Hack Your Language!

CS164: Introduction to Programming Languages and Compilers, Spring 2012

UC Berkeley

Final project

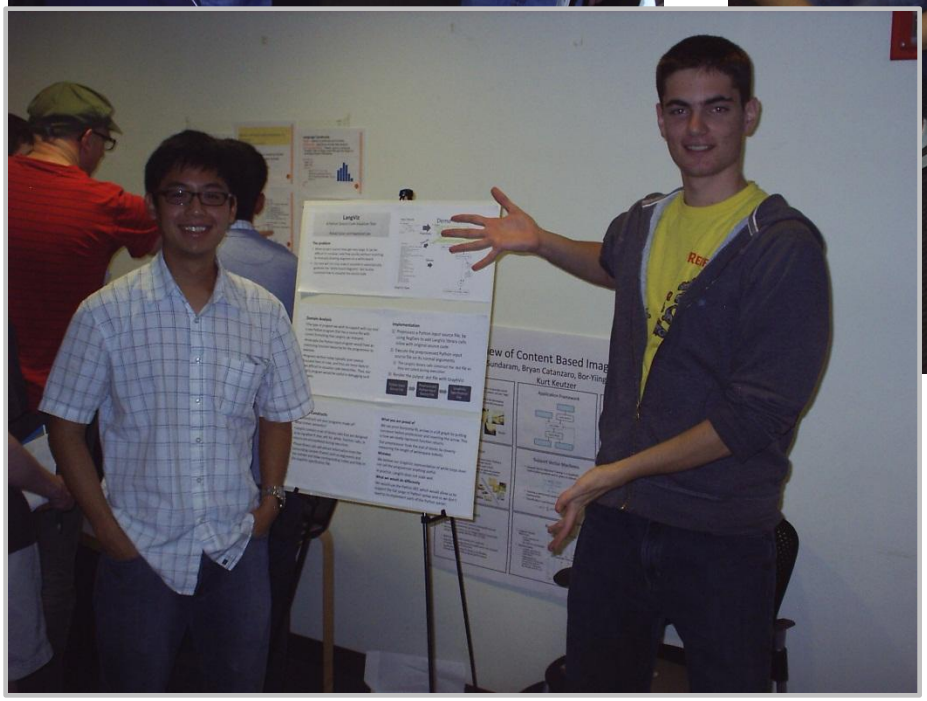
Think of it as self-designed PA10

The goal: convince yourself you can use CS164 skills to solve a real problem.

Typical final project: design and implement a small language.

Instead of final exam, you'll have free pizza and can demo your work!

CS164 Fall 2009 Demo session



Also will announce winners of contests

Best Parser Contest.
Best PA9 Browser Demo Contest.

Fall 2010

Project timeline

	13	28-Feb	Tu	natural language queries	midterm prep	submit **final project ideas**
		29-Feb	We			
	14	1-Mar	Th	data abstraction, OO		
		2-Mar	Fr			
8		5-Mar	Mo			
	15	6-Mar	Tu	**midterm**	PA5: Add **syntax-directed translation** to your parser. It can now work as a compiler and language translator. Add constructs for grammar disambiguation and write a few parsers. With	
		7-Mar	We			
	16	8-Mar	Th	types 1		
		9-Mar	Fr			
9		12-Mar	Mo			
	17	13-Mar	Tu	types 2	PA6: **Translate** a simple natural language SQL-like query to Prolog and Unit calculator. (In this assignment, you will have a chance to further debug and integrate the pieces you developed so	receive feedback on final projects
		14-Mar	We			
	18	15-Mar	Th	types 3		
		16-Mar	Fr			
10		19-Mar	Mo			
	19	20-Mar	Tu	types 4	PA7: Use your coroutine-based tree iterators to implement a **browser layout engine**. Connect it with your HTML-like parser and obtain your cs164 web browser.	
		21-Mar	We			
	20	22-Mar	Th	dataflow 1		
		23-Mar	Fr			
		26-Mar	Mo			
		27-Mar	Tu	*spring break*		**project proposal**: prepare implementation plan, submit slides for final presentations
		28-Mar	We			
		29-Mar	Th	*spring break*		
		30-Mar	Fr			
11		2-Apr	Mo			
	21	3-Apr	Tu	dataflow 2	PA8: (released before S/B) Add your scripting language to your 164 browser. Embed a little **jQuery**-like language.	
		4-Apr	We			
	22	5-Apr	Th	dataflow 3		
		6-Apr	Fr			
12		9-Apr	Mo			
	23	10-Apr	Tu	fun: advanced topics	PA9: **Reactivity.** Replace callback programming in your 164 browser with streams in the spirit of Rx.	
		11-Apr	We			
	24	12-Apr	Th	fun: advanced topics		
		13-Apr	Fr			
13		16-Apr	Mo			
	25	17-Apr	Tu	[garbage collection]		
		18-Apr	We			work on final project
	26	19-Apr	Th	class presentations 1		
		20-Apr	Fr			
14		23-Apr	Mo		midterm prep	
	27	24-Apr	Tu	class presentations 2		
		25-Apr	We			
	28	26-Apr	Th	**second midterm**		
		27-Apr	Fr			
		30-Apr	Mo			
		1-May	Tu			
		2-May	We			
		3-May	Th			
		4-May	Fr			
		7-May	Mo			
		8-May	Tu			
		9-May	We			
		10-May	Th			
		11-May	Fr	final exam (project demos)		

Final project proposal

Find a problem solvable with CS164 skills

- Your customers: programmers, end-users, web designers, …

Document how the problem is solved today

- Give example of typical code (illustrate today's problems)

Show how your small language would solve it (design)

- Rewrite typical code in your language

Outline the implementation

- Internal/external/hybrid? Compiled/interpreted?

One page of text. Due Sun Mar 3.

- Work in pairs or triples.

Finding the right problem is half the solution

A problem well stated is a problem half solved.

Inventor Charles Franklin Kettering (1876–1958)

We're all fairly good at problem solving. That's the skill we were taught and endlessly drilled on at school. Once we have a problem, we know how to turn the crank and get a solution. Ah, but finding a problem—there's the rub.

Engineering education is based on the presumption that there exists a predefined problem worthy of a solution. If only it were so!

From When the Problem is the Problem, Robert Lucky

Today

What you'll have built after PA9

- Your final project can build on PA1-9

Examples of cs164 projects

- animation
- browser extensions
- debugger for 164
- distributed continuations

Examples of influential DSLs

- protovis
- memoize
- mapReduce family

Programming Assignments

PA4-6: parser and compilers

- You'll know: write a grammar,
- translate programs to other language
- interpret programs, limited natural language processing

PA7-9: browser w/ modern scripting

- Parse and layout a subset of HTML;
- a subset of jQuery;
- reactive programming

Reactive Programming with events

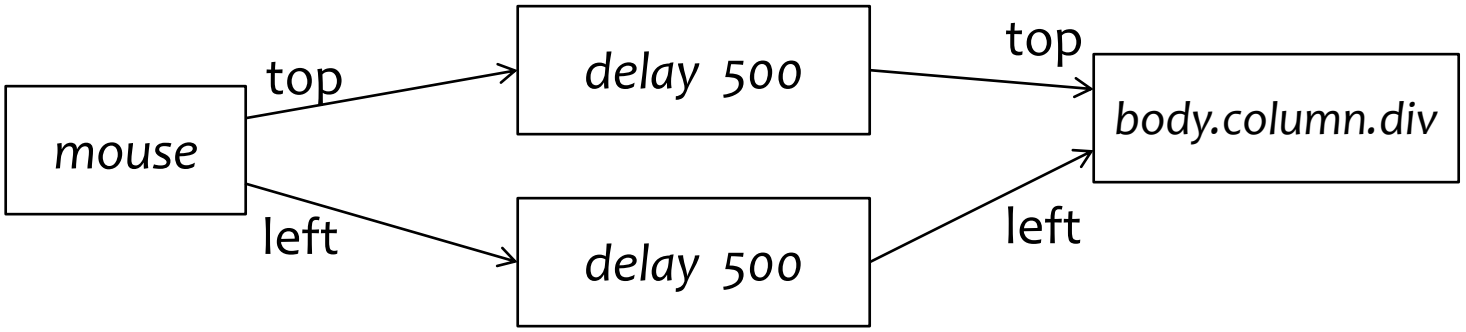
```
<div id="box" style="position:absolute; background: yellow;”>
  My box
</div>

<script>
document.addEventListener (
    'mousemove',
    function (e) {
        var left = e.pageX;
        var top = e.pageY;
        setTimeout(function() {
                document.getElementById("box").style.top = top;
                document.getElementById("box").style.left = left;
        } , 500);
    }, false);
</script>
```

PA9-like language

Program structure is clearer when data and control is explicit

- in dataflow version: **changing mouse coordinates are streams**
- coordinate streams adjust box position after they are delayed
- **structured names** of document element allow analysis



memoize

memoize

Memoize: a replacement for make.

Author: Bill McCloskey, Berkeley

Allows writing build scripts in "common" languages

- eg in Python or the shell
- rather than forcing you to rely on make's hopelessly recondite makefile language.

http://benno.id.au/memoize.py

Example 1: a shell script calling memoize

```
#!/bin/sh
memoize.py gcc -c file1.c
memoize.py gcc -c file2.c
memoize.py gcc -o program file1.o file2.o
```

Example 2: a python script calling memoize

```
#!/usr/bin/env python
import sys
from memoize import memoize
def run(cmd):
   status = memoize(cmd)
   if status: sys.exit(status)
run('ocamllex x86lex.mll')
run('ocamlyacc x86parse.mly')
run('ocamlc -c x86parse.mli')
run('ocamlc -c x86parse.ml')
run('ocamlc -c x86lex.ml')
run('ocamlc -c main.ml')
run('ocamlc -o program x86parse.cmi x86parse.cmo
   x86lex.cmo main.cmo')
```

How would you make it work?

Let's try to design it.

Goal: determine if a command needs to be rerun.

How memoize works

Key idea: determine if a command needs to run

Assumptions: a command is a pure function

- its output depends only on its input files
- common for compilers and other build tools

Computing Dependences (what cmd depends on):

- uses strace to intercept system calls, like open
- r = os.system('strace -f -o %s -e trace=%s /bin/sh -c "%s"' % (outfile, calls, ecmd))

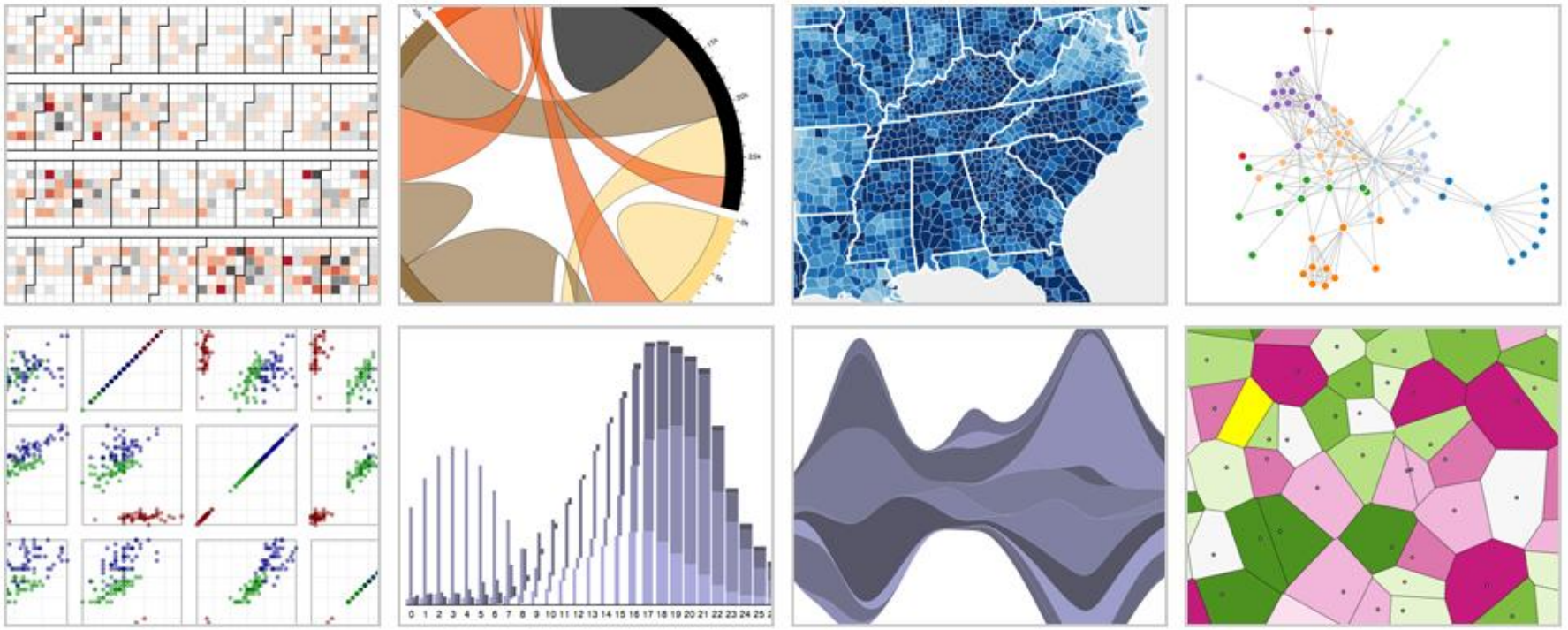
Computing file modification times:

- Alternative 1: use system file modification time
- Alternative 2: compute MD5 hash value for a value

Keep dependences and times in a file

D3

D3: a JS DSL for manipulating data documents



D3 example

Problems solvable by DSLs

Where did we already use DSLs in cs164 project:

- grammars
- graph visualization

Additional DSLs that would come really handy

- tree rewriting
- grammar debuggers
- environment visualizers

Problems solvable by DSLs

- scripting of games, build processes, etc
- templating of web pages and other documents
- graph layout (GraphViz)
- tree rewriting (GrGen)

MediaWiki Template DSL

The Template:Weather page has the text:

The Weather in {{{1}}} is always {{{2}}}.

An editor can then add the template {{Weather}} on several other wikipages. On the State of Maine page:

{{Weather|Maine|cold}}

Displays:

The Weather in Maine is always cold.

On the State of Florida page:

{{Weather|Florida|hot}}

Displays:

The Weather in Florida is always hot.

MapReduce family

- MapReduce
- Sawsall
- PlumeJava

Example projects from past cs164

Grainline: constraint language for tailors

TablUI:

brainstorming

Customers of your DSLs

Discussion continued

Low-risk final projects

- These projects are safe in that you don't need to come up with a problem to be solved by the language. You will still need to do some good thinking before you can implement these languages.

- Can implement in 164 or in another langauge (eg Lua, Python, JavaScript, etc)

Extend an existing DSL

- write a plugin for jQuery: example of a plugin that adds if/else to jQuery (this is a nice example bit it is too small for a final project)

- make jQuery animation a bit richer, for example allow some of these composable animations

- add a new kind of "mark" to protovis

Rethink an existing DSL

- Do jQuery or protovis better

Grow the 164 language

- extend 164 with some cool features, such as meta-programming (to support sugar directly)

Bug finding

- write a tool that finds a class of bugs (a program analyzer).
  - memory leaks
- This tool could instrument the program and identify potential bugs
- generate interface for C programs from a config file: examples: swig

Compilation and source-to-source translation

1) Compile the 164 language into more efficient code

ex: turn hashtables into structs (tuples) when possible

May involve adding static types or program analysis

2) Translate 164 to a language with a fast interpreter

eg Lua (Python and JS don't have full coroutines)

Motivation: remove interpretation overhead, thus enable more exciting final projects

Debugger for 164 language

- especially one that can be easily given to students in a starter kit
- breakpoints, pretty-printing of data
- exploit existing Python debuggers?

Pros

- Domain-specific languages allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify, and often even develop domain-specific language programs.
- Self-documenting code.
- Domain-specific languages **enhance quality**, productivity, reliability, maintainability, portability and reusability.
- Domain-specific languages allow **validation** at the domain level. As long as the language constructs are safe any sentence written with them can be considered safe.

Cons of DSLs

- Cost of learning a new language vs. its limited applicability
- Cost of designing, implementing, and maintaining a domain-specific language as well as the tools required to develop with it (IDE)
- Finding, setting, and maintaining proper scope.
- Difficulty of balancing trade-offs between domain-specificity and general-purpose programming language constructs.
- Potential loss of processor efficiency compared with hand-coded software.
- Proliferation of similar non-standard domain specific languages, i.e. a DSL used within insurance company A versus a DSL used within insurance company B.