CSC 548: Lecture 1

Alan Jeffrey

Overview

Course summary

Administrivia

Topics in advanced compilers

(Re)introduction to the Hobbes compiler

Aspect-oriented programming

Course summary

Implementation of modern object-oriented languages.

Topics include static analysis, register allocation, source-to-source transformation, single static assignment, and garbage collection.

Class structure (approximately!)

1. *Overview*: getting our bearings.
2. *Static analysis*: type-checking, liveness analysis (Appel, Ch. 5).
3. *Register allocation 1*: simple register allocation (Ch. 6, 11).
4. *Register allocation 2*: not-so simple register allocation (Ch. 10, 11).
5. *Object orientation*: subtyping, subclassing, vtables (Ch. 14).
6. *Midterm exam*.
7. *Functional SSA form*: single static assignment (Ch. 19).
8. *Optimizing functional SSA*: more SSA (Ch. 15, 19).
9. *Garbage collection 1*: algorithms for gc (Ch. 13).
10. *Garbage collection 2*: implementation of gc (Ch. 13).
11. *Final exam*

Administrivia: contact details

Lecturer: Alan Jeffrey

Email: ajeffrey@cs.depaul.edu

Office: CST 840

Phone: (312) 362 8322

Office hours: 3.30-5.00pm Tuesdays and Thursdays.

Administrivia: reading materials

Course home page: http://fpl.cs.depaul.edu/ajeffrey/csc548/, contains lectures, homeworks, pointers to tools, sample source code...

Textbook: *Modern Compiler Implementation in Java* by Andrew W. Appel, Cambridge University Press, 1998.

Administrivia: prerequisites

CSC 448: Compiler Design

Administrivia: required software

A Java 1.4 compiler.

The Ant build tool.

The JUnit testing tool.

The MinGW software tools.

The JavaCC parser generator.

AspectJ (from Eclipse).

Pointers to software are on the course home page.

Administrivia: assessment

Mid-term exam (6 May 2004): 25%

Final exam (10 June 2004): 25%

Weekly homeworks (submitted using Courses OnLine, best 7 out of 8): 50%

All students must attend the mid-term and final exams.

Late assignments will not be accepted without medical evidence.

Plagiarism or collusion is unacceptable, and will earn an F in the course.

Advanced compiler design

Why learn to write a compiler?

Why learn to write even more compilers?

Recap of compilers

Phases of a compiler: lexing/parsing, static analysis, optimization, code generation. What are these?

Coming up: static analysis

What is the input of an analyzer? The output?

What is the point of static analysis?

Two kinds of static analysis: type-checking, liveness analysis. What are these?

Coming up: register allocation

What is a register allocator?

A simple register allocator: put every variable in the same register. What are the features of this allocator?

Another simple register allocator: give every variable a different slot in the stack. What are the features of this allocator?

What would an optimal register allocation be?

What is the time complexity of optimal register allocation?

Coming up: object orientation

What are the features of OO languages?

What is the difference between subclassing and subtyping?

What is a vtable? Why do we need vtables?

Coming up: functional SSA

What is a pure functional language?

Why are pure functional languages easier to optimize than imperative languages?

How can we translate an imperative language into a pure functional language? Why?

Coming up: garbage collection

What is a garbage collector? Why are gcs useful?

How does a gc work?

Why are good gcs difficult to implement?

The Hobbes language

Hobbes is a stripped-down OO language.

Features of Hobbes:

- 'Pure' OO: everything is an object
- Top-level declarations of classes, objects and threads
- Encourages immutable programming

Features we will add:

- Code generation down to IA32 assembler
- OO features: inheritance and dynamic dispatch
- Mutable variables, while loops etc.
- GC

The Hobbes compiler

To run the compiler on the test suite:

```
cd compiler
ant test
```

This requries the Ant build manager, the JUnit test tool, the JavaCC parser generator, and AspectJ. Assembling will require the MinGW tools.

Homeworks will involve rewriting parts of the compiler!

Aspect-oriented programming

What is aspect-oriented programming?

Why is AOP useful in large software systems?

Why is AOP useful in a compiler?

Aspect-oriented programming

An AspectJ program:

```
public interface Foo { ... }
class FooImpl implements Foo { ... }
aspect MyAspect {
  declare parents: Foo extends Runnable;
  public void Foo.run () { System.out.println ("Hello world"); }
}
```

What does the MyAspect aspect do?

Aspect-oriented programming

To get the tools needed to run the compiler...

1. Download Ant and JUnit (you should already have this!)

2. Download AspectJ 1.0.6, including the tools, documentation and Ant tasks. (Note: AspectJ 1.1 will not work with my code!)

3. Install AspectJ by running `java -jar aspectj-tools-1.0.6`

4. Using Winzip, extract `aspectj-docs-1.0.6.tgz` and `aspectj-antTasks-1.0.6.tgz` into the directory you chose for 3.

5. Copy `aspectj-ant.jar`, `aspectjrt.jar` and `aspectjtools.jar` from `aspectj1.0\lib` to `apache-ant-1.6.1\lib` (make sure `junit.jar` is there too!).

6. Test it with `ant test`.

Aspect-oriented programming

ant test
Buildfile: build.xml

prepare:

compile:
    [ajc] Compiling 24 source and 0 arg files to /home/alan/teaching/csc548/spring2003-2004/co

test:
    [junit] TEST hobbes.pretty.PrettyTest FAILED

BUILD SUCCESSFUL
Total time: 11 seconds

The homework is to get this test to pass!

Pretty printing

Homework 1 is about the pretty printer: it shouldn't be rocket science!

This homework is to get you used to the code base and AspectJ.

What is a pretty printer? How can we implement one?

A test program for a pretty-printer is in test/hobbes/pretty/PrettyTest.java.

Summary

This course will cover static analysis, register allocation, object-orientation, functional SSA and garbage collection.

The homeworks will involve developing features of the Hobbes compiler using AOP.

Reading: Appel, Ch. 5