Overview of MiniJava

Lecture 4

Course Administration

- If you drop the course, please make it official
- If you are on the waiting list, see Michael-David Sasson in 379 Soda Hall
- I think that anyone enrolled can get ID card registered to get into Soda Hall after hours.
  - Go to CS reception (Soda Hall 3rd floor)

- Questions about course policies?

Andrew Appel, the textbook author is at Princeton University

- The first version of our text used a language named "Tiger" (the Princeton mascot). Designed by Appel for teaching about compilers.
- Why is designing a new language and describing it so hard?
- Why?
  - Except if it is trivial, there are many little details.
  - Describing a language: how? Axiomatically? Informally? Operationally?.

Why the change?

- Except if it is trivial, there are many little details to a language that must be nailed down.
  - Syntax
  - Semantics
  - Pragmatics
- Tiger was simple/tricky/boring.(?)
- How to be formal yet readable? Definition Axiomatically? Informally? Operationally?.

MJ Overview

- MJ is a language that is supposed to be strictly a subset of Java.
- Designed to
  - Provide a useful one-semester exercise
  - Give a taste of implementation of modern
    - Abstraction
    - Static typing
    - Memory management
    - Object Oriented implementation of functions
- How to leave out things?

Reminscent of the novel *Gadsby* by Ernest Wright

- The entire manuscript of Gadsby was written in a subset of the English language. Namely, without the letter "e"
- http://www.spinelessbooks.com/gadsby/

From page one of Gadsby …

A child's brain starts functioning at birth; and has, amongst its many infant convolutions, thousands of dormant atoms, into which God has put a mystic possibility for noticing an adult's act, and figuring out its purport.

Up to about its primary school days a child thinks, naturally, only of play. But many a form of play contains disciplinary factors. "You can't do this," or "that puts you out," shows a child that it must think, practically or fail. Now, if, throughout childhood, a brain has no opposition, it is plain that it will attain a position of "status quo," as with our ordinary animals. Man knows not why a cow, dog or lion was not born with a brain on a par with ours; why such animals cannot add, subtract, or obtain from books and schooling, that paramount position which Man holds today.

But a human brain is not in that class. Constantly throbbing and pulsating, it rapidly forms opinions; attaining an ability of its own; a fact which is startlingly shown by an occasional child "prodigy" in music or school work. And as, with our dumb animals, a child's inability convincingly to impart its thoughts to us, should not class it as ignorant.

Anyway, Lexical matters in MJ are reasonable subset

- Identifiers begin with a letter
- Integers and Booleans (true, false) are the only basic data types.
- Binary arithmetic or logical operators are && < + - *
- Additional operators are ()[]{}=.!;
- NO Strings. NO floats.
- Comments are /* */ or // to end of line. NOT nested, contrary to p. 484. *(but we will allow nests)*

Restrictions that are major simplifications in MJ vs Java

- There is a main class with additional class declarations at the top level.
- No overloading.
- No classes defined within classes.
- All methods are public.
- No initializers by calling <name of> class
- No INPUT (all data must be literal integers in the program body).

Semantics: what does an MJ program compute?

- Conditions for an alleged MJ program to be legal.

1. The program must grammatically fit in the terms of the (page 485) MiniJava grammar.
2. The program must be legal Java. In which case---
   The meaning of the MJ program is the same as the corresponding Java program.

- Why does this help?
  - There is a readily available reference implementation (javac, java)
  - There are (many) books, including some standards, that can provide informal descriptions, formal descriptions, examples.

Semantics: Is this just a cheap trick?

- Yes, but you probably think you know Java, so you might not mind.
- Even if you don't know Java
  - There is a readily available reference implementation (javac, java) for your computer.
  - There are (many) books, including some standards, that can provide informal descriptions, formal descriptions, examples of Java programs.
  - There are examples of MJ programs in the text's web site (also locally)

Java and even more so, MJ, can be written in Lisp

- Java's authors were influenced by Common Lisp (G.L. Steele, Jr. especially), for example, implicit pointers (like lisp, not C)
- Lisp, Java, and MJ all use garbage collection: We don't have to write one, which is what would have to be done if we implemented MJ in C.
- We can implement operations in Lisp that are supersets of MJ and then cut them off at the knees if we want to be strict. E.g. Lisp can read integers of any length (or operate on them). A strict implementation would truncate them to what's legal in Java.

MJ Primitive Types, Constructors

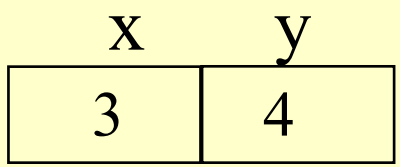
- Integer
- Boolean
- No strings, characters, floats

- Ways of combining
  - Arrays
  - Classes /methods /variables/ inheritance

MJ variables live in classes or instances

```
class Point {
    int x; int y;
    public boolean Init
              (int newx, int newy)
     x = newx; y=newy;
     return true;}
//…
Point a; boolean z;
 a = new Point; z=a.Init(3,4);
```

- The value for "a" is [a pointer to] a record with a slot for each attribute.



Methods are really functions

- Methods have names, parameter lists with types, optional return type (else VOID), bodies, and lexical scope.
- The names are distinguished by the classes of the instance being referenced, as well as the types of the arguments. Names have scope within methods.
- A simple single-inheritance hierarchy exists for classes. [maybe compare to OO in Scheme, CLOS?]
- All programs are defined in a global lexical environment which includes built-in functions like system.out.println -- and not much else.

Information Hiding in MJ

- Overloading is forbidden.
- Names are visible lexically (as in Scheme!)
- But not nesting makes most of this trivial.

Print is really very weak

- Print only prints integers, which it magically knows about, but doesn't know about strings or anything else.

MJ has rather simplistic object oriented stuff in it

- So how will we learn about cool OO stuff, beyond that in Java?
  - We will do this in ANSI CL, later, and in a neater form.

MJ Types

- All variables must be declared
  - compiler cannot infer simple types for variables from initial values
- All intermediate expressions, e.g. in 3*(x+4*y) are of obvious types.
- No "casts".

MJ Type Checking: any holes?

Instances exist before they have values.

If an implementation checks for this possibility at runtime, all other type errors should be identifiable by a static compile-time checking program. (Compare to, say Lisp, or C).

Are there any other runtime conditions that need to be checked? [Assume MJ allows input]

Other components: Expressions, Statements

- Mostly a Statement language + *,- …
- Loops:
- while (E) do S
- Conditionals if E S else S
- if E S
- Arithmetic, logical operations
- Assignment x = E
- Access x.length, this.fun(..)
- Sequences {S ; S; …}

Compare to Lisp which is

Mostly an EXPRESSION language

in Java and MJ, this is illegal: x=if z 3; else 4.
in Lisp, this IS legal: (setf x (if z 3 4))

(Java but not MJ has a conditional expression
x= (z)?3:4 though…)

MJ Memory Management

- Memory is allocated every time an instance of a class or an array is created. Calls can be recursive requiring a stack. We can allocate space (in fact my MJ implementation does…) on a heap for environments (remember CS61a?)
- Memory is deallocated automatically when an object is not reachable anymore
  - Done by the garbage collector (GC)
  - We use the Lisp GC, but it is done "automagically"

Course Project

A Lexical Analyzer

- MJ source -> token-stream

A type-checker

- MJ source ➔ token-stream ➔intermediate code = data for typechecker.

A complete code tree for an interpreter

- MJsource ➔ token-stream ➔intermediate code = data for simple interpreter to be evaluated

Add this to an interpreter and you can run MJ.

Course Project / II

- Modify interpreter so we have a complete compiler
  - MJ ➔… ➔our own assembly language for which we also have an assembler and a machine simulator.
- Split in 6 (or so) programming assignments (PAs)
  - **Lex, Parse, IC (augmented parser), typechecker, interpreter (given to you?), compiler**
- There is adequate time to complete assignments well before the end of the semester.
- Individual or team (max. 2 students)

Programming Assignment I reminder

- Due TONIGHT,Thursday, 11:59PM
  - Turn in via the SUBMIT command:
    - cd to your directory, say hw1, with files hw1.cl and README

    type submit hw1

Programming Assignment II

- Due Sept 22. 11:59PM
  - Lexical analysis, skeleton code provided.
  - Tools vs hand-coding