Final Review

COMP 524: Programming Languages
Srinivas Krishnan
April 25, 2011
Lay of the Land

Question Types

➡ Simple True/False, Fill in the Blanks
➡ Problem Solving: Read a scenario and apply what you have learnt so far
➡ Theoretical Problems: Similar to midterm, there will be one asymptotic complexity question on Generational GC ;)
➡ Code: Write Snippets of Code, Read and Evaluate Code
➡ Extra Credit: Problem Solving based on class lectures
Lay of the Land

Nuts n Bolts

- 120 Minutes, but you can take all 180 minutes if you want
- Closed Book
- Total Points: -100
- Coverage: All topics since the beginning of semester, focus on topics on the latter half of the semester
Suppose you are stranded on a desert island with a computer (with an assembler and a basic OS) and need a compiler for a high-level language such as Haskell. How do you obtain one?

By cross-compiling.

1) Start by modifying the existing Haskell compiler that can generate x86 machine code to also be able to generate PowerPC machine code.

2) Now, compile the modified compiler with itself to produce a compiler that runs on x86 but produces PowerPC machine code.

3) Use the compiler produced in step 2), which runs on x86 but produces PowerPC code, and compile itself again. This time, the result is a compiler that runs on PowerPC and that produces PowerPC machine code.

We went from a x86->x86 compiler to a PowerPC->PowerPC compiler.
Still on the desert island.
Suppose we have a working, self-hosting Haskell compiler on our rescue pod computer (an Intel x86 machine), but found a PowerPC computer in a satellite wreck, and would like to have a working, self-hosting Haskell compiler on the satellite computer.

By cross-compiling.

1) Start by modifying the existing Haskell compiler that can generate x86 machine code to also be able to generate PowerPC machine code.

2) Now, compile the modified compiler with itself to produce a compiler that runs on x86 but produces PowerPC machine code.

3) Use the compiler produced in step 2), which runs on x86 but produces PowerPC code, and compile itself again. This time, the result is a compiler that runs on PowerPC and that produces PowerPC machine code.

We went from a x86->x86 compiler to a PowerPC->PowerPC compiler.
What are the first two phases of a compiler?

Lexical analysis and syntax analysis.
How can we recognize regular grammars?

With DFAs (deterministic finite automata).
What are the four types of terms in Prolog

Atoms, variables, numeric literals, and structures
Write a Prolog clause xor_equal/4 that has the following behavior:

\[
\begin{align*}
xor\_equal&(a, a, b, c). \Rightarrow true. \\
xor\_equal&(a, b, b, b). \Rightarrow true. \\
xor\_equal&(a, a, b, b). \Rightarrow false. \\
\end{align*}
\]

\[
\begin{align*}
xor\_equal&(A, B, C, D) :&- A = B, C = D, !, fail. \\
xor\_equal&(A, B, _, _) :&- A = B, !. \\
xor\_equal&(_, _, C, D) :&- C = D. \\
\end{align*}
\]
How can you do iterative computations in Prolog?

Only with recursion, since there are no loops
What is the Cut Operator used for in Prolog?

Control backtracking.

\[
\begin{align*}
  a & :- b. \\
  b & :- c. \\
  c & :- d. \\
  b. \\
  a(X) & :- b(X), c(X), !, d(X).
\end{align*}
\]
Cut Operator

“Cut” branches from the search tree.

- Avoid finding “too many” answers.
  - E.g., answers could be symmetrical / redundant.

Superfluous answer because \( X \) unified with both \( B \) and \( C \).

```prolog
one_of(X, A, _, _) :- X = A.
one_of(X, _, B, _) :- X = B.
one_of(X, _, _, C) :- X = C.

?- one_of(unc, duke, unc, state).
true .

?- one_of(unc, duke, unc, unc).
true ;
true .
```
Cut Operator

controlling backtracking

The cut (!) predicate.

- Written as exclamation point.

- **Always succeeds**.

- **Side effect**: discard all previously-found backtracking points.
  - i.e., **commit to the current binding of variables**; don’t restore.

```prolog
one_of_cut(X, A, _, _) :- X = A, !.
one_of_cut(X, _, B, _) :- X = B, !.
one_of_cut(X, _, _, C) :- X = C.
?- one_of(unc, duke, unc, unc).
```
Negation

Prolog negation differs from logical negation.

- Otherwise not implementable.
- **Math**: \( \text{not } X \) is true if and only if \( X \) is false.
- **Prolog**: \( \text{not } X \) is true if goal \( X \) cannot be satisfied.
  - i.e., \( \text{not } X \) is true if Prolog cannot find an answer for \( X \).

**SWI Syntax**: \(+ X\) means \texttt{not X}.

Can be defined in terms of **cut**.

\[
\text{not}(X) :- \text{call}(X), !, \text{fail}.
\]

\[
\text{not}(X).
\]
State the binding time for the following

- "if"
- `#define D = 10`
- `printf("%s", "Hello World")`

```python
class Base(object):
    def a_method(self):
        print "a_method was called"
    def dangerous(self):
        print "Replacing methods can cause tricky bugs!"

Base.a_method = dangerous
```

Design Time
Compile Time
Link Time
Run Time
Describe a memory allocation policy for short and long lived objects

- `int a = 10; // Short Time on the stack`
- `int *a = (int *)malloc(sizeof(int)) // Long Lived on the Heap`
- `*a = 10`
What does a language that supports function calls need to implement?

A runtime stack.
What is kept on a runtime stack?

- Parameters
- Local Variables of the function
- Where to return once the function exits

Extra Credit: What are 2 assembly instructions on x86 for stack management?

- Push and Pop
How did Fortran support Subroutines?

Static Allocation of Memory

What is the disadvantage of Static Allocation?

Waste of Memory, Limited Recursion depth
What are the 4 steps of a Call Sequence

➡ Setup, before call to subroutine.
➡ Prologue, before subroutine body executes.
➡ Epilogue, after subroutine body completes (the return).
➡ Cleanup, right after subroutine call.
What are the 3 ways to pass parameters

• Pass By Value: C/C++
• Pass By Reference: C/C++, Java
• Pass By Name: Haskell
List all common parameter types and their purpose.

- positional: normal parameters
- optional/default: parameters that are mostly the same
- variable parameters: no predetermined number of actual parameters
- keyword parameters: can appear in any order
How can you avoid exponential recursion

Use Tail Recursion

• Pass values to the next stage instead of evaluating a function to retrieve the value
How can you optimize evaluation of a boolean expression?

Use short-circuit

```java
HashMap dict = null;
// possibly initialized by other code
if (dict != null && dict.contains("key"))
    // do something;
```
What is the referencing environment?

The set of all active bindings; all valid names
What do you need to implement a nested subroutine?

**Closest nested scope:** a binding is active in the scope in which it is declared and in each nested scope, unless it is shadowed by another binding.

Need Symbol Table support.

Push a new frame onto the stack.
What is an anonymous function?

Simply a function that is not bound to a name.

Anonymous functions are not necessarily nested, and not necessarily a function parameter.

Lambda Expressions:
print 'by last', sorted(strings, key=lambda x: x[-1])

(lambda x: x * x)(2)
In a nested subroutine, what is a free variable?

Any variable that is neither a formal parameter (passed to the function) nor a local declaration.
What is a closure?

A nested subroutine in which the free variables are bound to entities (objects) residing in the lexical scope in which the nested subroutine was defined.
Closure Example: A “Hidden” Stack

```
Python:

def make_stack():
    mystack = []
    def _push(x):
        mystack.append(x)
    def _pop():
        val = mystack[-1]
        del mystack[-1]
        return val
    return (_push, _pop)

(a, b) = make_stack()
(c, d) = make_stack()
a(1); a(2); a(3)
c(9); c(8); c(7)

print 'b:', b(), b(), b()
print 'd:', d(), d(), d()
```

Output:

```
b: 3 2 1
d: 7 8 9
```
What’s the difference between call-by-name and inlining?

Call-by-name is a function call semantics (that could be implemented in different ways). It defines how parameters are used.

Inlining is a compiler optimization (that may not change the calling semantics). It applies to how a function is called.

Inlining can apply to any function call semantic.
What is the key benefit of a statically, weakly typed language?

Execution speed: no runtime checks are required.
Which data type fundamentally requires runtime checks in a strongly typed language?

Disjoint union types; tag checks must occur at runtime.
Disjoint Union

One value, chosen from multiple (disjoint domains).

**Mathematical view.**

- Simply a union of all possible types (= sets of values).
- Each value is **tagged** to tell to which domain it belongs.
  - Tag can be used for checks at runtime.

\[
\{(1) \times S'\} \cup \{(2) \times T\} = \{(t, x) | (t = 1 \land x \in S) \lor (t = 2 \land y \in T)\}
\]

**Example:**
A pixel color can be defined using **RGB** (red, green, blue color channels) or **HSB** (hue, saturation, brightness). Both are simply **three-tuples**, but values must be distinguished at runtime in order to be rendered correctly.
Disjoint Union in Haskell

*enumeration of named tuples*

**Algebraic data type.**

→ Generalizes enumeration types and composite types.

--- Implicit fields: only types are given, no explicit names
--- These can be accessed using pattern matching
--- (de-structuring bind).

```haskell
data Coordinate = Coord2D Int Int
               | Coord3D Int Int
```

--- Enumeration type.

```haskell
data ColorName = White | Black | Green | Red | Blue | CarolinaBlue
```

--- Explicit field names.

```haskell
data Color = RGB { red :: Int, green :: Int, blue :: Int}
            | Named ColorName
            | HSB { hue :: Double, sat :: Double, bright :: Double}
```

--- Composite type of composite types.
--- Again, implicit fields.

```haskell
data Pixel = Pixel Coordinate Color
```
When is a recursive function call tail-recursive?

When it is the last expression to be evaluated before the return from the function.
When is a recursive function tail-recursive?

When every possible control flow path contains either:
1) exactly one recursive call that is tail-recursive, or
2) no recursive call at all.

In other words, a single non-tail-recursive call is sufficient to render a function non-tail-recursive.
def recsum(x):
    if x==1:
        return x
    else:
        return x+recsum(x-1)

No, the function is not tail-recursive, since the recsum needs to be evaluated before ‘+’ can occur.
Fix the previous recursive function

def tailrecsum(x, running_total=0):
    if x == 0:
        return running_total
    else:
        return tailrecsum(x - 1, running_total + x)
Explain early and late binding in the context of object-oriented languages.

With early binding, the method being invoked is determined at compile time based on the type of the reference.

With late binding, the method being invoked is determined at run time based on the type of the value (i.e., the object).
Name two advantages of delegation.

1) Avoids the fragile base class problem.
2) Facilitates the use of “fat” interfaces, i.e., interfaces that require many methods to be implemented.

Does Java support delegation?
No !!
Delegation Example

```csharp
interface Bar {
    void bar();
}

class DefaultBar implements Bar {
    void bar() {
        ...
    }
}

class MyClass implements Bar {
    private DefaultBar barImpl = new DefaultBar();

    void bar() {
        barImpl.bar();
    }
}
```

C# provides explicit delegate syntax
What’s “monkey patching?”

Modification of classes at runtime (adding / removing / replacing methods and attributes).
Give an example of a language that supports runtime modification of its objects and classes?

Python with a caveat??

Library does not have a C backend
The class concept is fundamental to all object-oriented languages. True or false? (Why?)

No, prototype-based languages do not require classes
Give three uses of a RTS

- Garbage Collection
- Error Checking
- Just-In-Time Compilation
Describe how a Mark-Sweep GC works?
What is wrong with this graph when Ref Counting is used?

Memory leak when stooges is freed
How would you fix the previous graph?

- Use tags with ref count to make sure stack and heap references are separated.
How much memory is available in a Copy Based GC?

Half the memory, as you maintain a free and used regions.
What is “Finalizer” and When do you use it?

Release all non-memory resources e.g. File Handles, Thread resources
What is the possible catch-22 in GC?

GC runs when memory is low, and GC needs memory to run. Concurrent GC’s solve this problem.
When does JIT code incur penalty and how many times?

One-time penalty, initially when the byte code is compiled for heavily used objects.

For rarely used code segments, penalty is proportional to the number of accesses.
How is Binary Translation different from JIT?

BT works on machine code to machine code transformation e.g. ESX server uses BT to implement virtualization

JIT works at the level of byte code. e.g. Java uses JIT to speedup runtime of applications
When can JIT be more aggressive?

Long lived applications, allows the JIT to learn from its mistake and better amortize the cost of a mistake. E.g. Server apps of Java.
What is the difference between MultiProgramming and MultiThreaded programming

Multiprogramming gives you the illusion of concurrency by interleaving processes on a single CPU

Multi-threaded programs on multiple processors have the ability to concurrently run at the same time
Speedup: ➔ If a task takes time ‘t’ on one processor, on ‘n’ processors should the time be t/n?

**No, Amdahl’s law:** The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program.
How many students did actually submit review questions?