

CS421 Summer 2006 Midterm Study Guide

CS 421 — Introduction to Programming Languages
Summer Term 2006

1. OCaml and Higher-Order Functions

- (a) Using just `fold`, write a function `max` that finds the largest element in a list of numbers. Consider this to be 0 for an empty list.

- (b) Say you are given the following definition of lists:

```
type 'a mylist = Empty | Cons of 'a * 'a mylist
```

Write a function `myMap` that takes a function and maps it over the elements of a `mylist`.

- (c) Say you are given the following definition of trees, with an arbitrary number of children:

```
type 'a mytree = Leaf of 'a | Node of 'a mytree list
```

Write a function `fold_mytree` that takes a function for the node case, a function for the leaf case, an identity, and a `mytree`, and returns the result of folding the functions over the tree. Note, you may want to think about mutually recursive functions here.

- (d) Using `fold_mytree`, write a function `sum_mytree` that will sum the values of the leaves.

2. Type Derivations

Note, type derivation rules are on the last page.

(a) Show the complete type derivation for the following term:

```
(fun x -> if x = 0 then true else false) 3
```

(b) Show the complete type derivation for the following term:

`let x = 5 in g x`

where `g` is assigned type `g : int → string`

3. Unification

- (a) Give a most general unifier for the following unification problem. Lower case letters (f, g, h, d) are constants or term constructors: specifically, f and g are term constructors with arity 1, h is a term constructor with arity 2, and d is a constant term with arity 0. Letters α, β , and γ are variables. Show your work by listing the operation performed in each step of unification – decompose, orient, delete, eliminate – and the result of the step. If unification is not possible, work as far as possible and show where unification fails. If unification does not fail, show the final substitution, which should be a set of variable to term mappings.

$$\{\mathbf{f}(\alpha) = \mathbf{f}(\mathbf{g}(\gamma)); \gamma = \mathbf{d}; \alpha = \mathbf{g}(\beta); \mathbf{h}(\alpha, \gamma) = \mathbf{h}(\alpha, \beta)\}$$

- (b) Using the unifier discovered above, apply the substitution and show the final terms, which should be equalities and which should have no variables. If unification got stuck above, just write “unification failed” below.

4. λ -Calculus

- (a) Write the λ -calculus version of `or`. You will need the following definitions:

$$\begin{aligned}\text{True} &\equiv \lambda x.\lambda y.x \\ \text{False} &\equiv \lambda x.\lambda y.y \\ \text{if_then_else} &\equiv \lambda b\ c\ d.b\ c\ d\end{aligned}$$

- (b) Write a function that, given a pair `p`, will return pair `q` such that the first element of `p` is the second of `q` and the second of `p` is the first of `q` (i.e. that flips the two elements of the pair). You will need the following definitions:

$$\begin{aligned}\text{pair } a, b &\equiv \lambda a\ b\ x.x\ a\ b \\ \text{fst } p &\equiv \lambda p.p(\lambda x\ y.x) \\ \text{snd } p &\equiv \lambda p.p(\lambda x\ y.y)\end{aligned}$$

- (c) Using the definition of plus for Church numerals, show the result of adding the Church numeral for 1 to itself – show all β and α reduction steps needed, and use a lazy evaluation order.

$$\begin{aligned} 1 &\equiv \lambda f x.f x \\ \text{plus} &\equiv \lambda n m f x.n f(m f x) \end{aligned}$$

- (d) Using the function to flip the elements of pairs defined above, illustrate the use of this function on the pair $\lambda x.x c d$ (here c and d could be any two terms, but we don't need to look inside them so we can just abstract them away). You may use any evaluation order.

5. Regular Expressions

(a) Given the alphabet $\{x, y, z\}$, provide a regular expression that defines all strings ending in z that have at least one y

(b) Given the alphabet $\{a, b\}$, provide a regular expression that defines all strings of a 's and b 's that start and end with the same letter.

6. DFAs and NFAs

(a) Given the alphabet $\{x, y, z\}$, define an NFA that recognizes all strings ending in z that have at least one y

(b) Given the alphabet $\{a, b\}$, define a DFA that recognizes all strings of a 's and b 's that start and end with the same letter.

Rules for type derivations:**Constants**

$$\frac{}{\vdash n : \text{int}} \text{(assuming } n \text{ is an int)}$$

$$\frac{}{\vdash \text{true} : \text{bool}}$$

$$\frac{}{\vdash \text{false} : \text{bool}}$$

Variables

$$\frac{}{\Gamma \vdash x : \tau} \text{if } (x : \tau) \in \Gamma$$

Arithmetic Operators

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}} \quad (\oplus \in \{+, -, *, /, \dots\})$$

Relational Operators

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \sim e_2 : \text{bool}} \quad (\sim \in \{<, >, \leq, \geq, =, \neq, \dots\})$$

Booleans

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \ \&\& \ e_2 : \text{bool}}$$

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \ || \ e_2 : \text{bool}}$$

$$\frac{\Gamma \vdash e_1 : \text{bool}}{\Gamma \vdash ! e_1 : \text{bool}}$$

If

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \text{if } e_1 \ \text{then } e_2 \ \text{else } e_3 : \tau}$$

Function Abstraction

$$\frac{\Gamma \cup [x : \tau_1] \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2}$$

Function Application

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 \ e_2 : \tau_2}$$

Let

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \cup [x : \tau] \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \ \text{in } e_2 : \tau'}$$

Letrec

$$\frac{\Gamma \cup [x : \tau] \vdash e_1 : \tau \quad \Gamma \cup [x : \tau] \vdash e_2 : \tau'}{\Gamma \vdash \text{let rec } x = e_1 \ \text{in } e_2 : \tau'}$$