CS 421 Summer 2010 Midterm II

July 15, 2010

The total time for this exam is 70 minutes.

Print your name and netid below. Also write your netid at the top of each subsequent page.

Name:

Netid:

- This is a **closed-notes** exam. You are allowed only the materials in the exam packet. All other materials, besides pens, pencils, and erasers, are to be put away.
- Do not share anything with other students. Do not talk to other students. Do not look at another student's exam. Also, be careful not to expose your exam to easy viewing by other students. Violation of any of these rules may constitute cheating.
- If you believe there is an error, or an ambiguous question, you must document your assumptions about what the question means. The proctors are not allowed to answer questions about the exam other than the meaning and usage of English words and phrases.
- Including this cover sheet and scratch pages, there are 8 pages to the exam. Please verify that you have all 8 pages.

Question	Total points	Score	Grader
1	12		
2	10		
3	14		
4	10		
5	10		
6	14		
TOTAL	70		

1. **(12 points)**

   Give a type inference and a constraint set for the following expression.

   `let rec f = fun x -> if x<0 then (x*x,x) else let g = f in (0,0) in 1`

   You don't need to solve the constrains, just present them. Show all of the steps with the name of the rule that you are applying in each step. A reference sheet containing all rules is at the end of this exam.

Solution:

(1) LetRec:
$\Gamma \vdash$`let rec f = fun x -> if x<0 then (x*x,x) else let g = f in (0,0) in 1 :` α
(1-1) Fun:
$\Gamma \cup [$`f` $: \beta] \vdash$`fun x -> if x<0 then (x*x,x) else let g = f in (0,0) :` β
(1-2) Int:
$\Gamma \cup [$`f` $: \beta] \vdash$`1 :` α (α,`int`)
(1-1-1) If:
$\Gamma \cup [$`f` $: \beta,$ `x` $: \gamma] \vdash$`if x<0 then (x*x,x) else let g = f in (0,0) :` δ (β,$\gamma \rightarrow \delta$)
(1-1-1-1) App: (here we know that < is a function with integer parameters)
$\Gamma \cup [$`f` $: \beta,$ `x` $: \gamma] \vdash$`x<0 : bool` (γ,`int`)
(1-1-1-2) Pair:
$\Gamma \cup [$`f` $: \beta,$ `x` $: \gamma] \vdash$`(x*x,x) :` (τ_1, τ_2) (δ,(τ_1,τ_2))
(1-1-1-2-1) Arith:
$\Gamma \cup [$`f` $: \beta,$ `x` $: \gamma] \vdash$`x*x :` τ_1 (γ,`int`), (τ_1,`int`)
(1-1-1-2-2) :
$\Gamma \cup [$`f` $: \beta,$ `x` $: \gamma] \vdash$`x :` τ_2 (τ_2,γ)
(1-1-1-3) Let:
$\Gamma \cup [$`f` $: \beta,$ `x` $: \gamma] \vdash$`let g = f in (0,0) :` δ
(1-1-1-3-1) :
$\Gamma \cup [$`f` $: \beta,$ `x` $: \gamma] \vdash$`f :` η (η, β)
(1-1-1-3-2) Pair :
$\Gamma \cup [$`f` $: \beta,$ `x` $: \gamma,$ `f` $: \eta] \vdash$`(0,0) :` (μ_1, μ_2) (δ, (μ_1,μ_2))
(1-1-1-3-2-1) Int:
$\Gamma \cup [$`f` $: \beta,$ `x` $: \gamma,$ `f` $: \eta] \vdash$`0 :` μ_1 (μ_1,`int`)
(1-1-1-3-2-2) Int:
$\Gamma \cup [$`f` $: \beta,$ `x` $: \gamma,$ `f` $: \eta] \vdash$`0 :` μ_2 (μ_2,`int`)

2. **(10 points)**

   Find the most general substitution that satisfies the following set of constrains. Show each step of unification algorithm as you perform it.

$$\Big\{\big(s(s(z)), s(w)\big), \big(f(x, g(x)), f(y, z)\big), \big(g(y, x), g(s(v), y)\big)\Big\}$$

Solution:

decomposition: $\Big\{\big(s(z), w\big), \big(f(x, g(x)), f(y, z)\big), \big(g(y, x), g(s(v), y)\big)\Big\}$

orientation: $\Big\{\big(w, s(z)\big), \big(f(x, g(x)), f(y, z)\big), \big(g(y, x), g(s(v), y)\big)\Big\}$

elimination: $\Big\{\big(f(x, g(x)), f(y, z)\big), \big(g(y, x), g(s(v), y)\big)\Big\}$ with $w = s(z)$

decomposition: $\Big\{(x, y), \big(g(x), z\big), \big(g(y, x), g(s(v), y)\big)\Big\}$ with $w = s(z)$

elimination: $\Big\{\big(g(y), z\big), \big(g(y, y), g(s(v), y)\big)\Big\}$ with $w = s(z)\ o\ x = y$

orientation: $\Big\{\big(z, g(y)\big), \big(g(y, y), g(s(v), y)\big)\Big\}$ with $w = s(z)\ o\ x = y$

elimination: $\Big\{\big(g(y, y), g(s(v), y)\big)\Big\}$ with $w = s(z)\ o\ x = y\ o\ z = g(y)$

decomposition: $\Big\{\big(y, s(v)\big), (y, y)\Big\}$ with $w = s(z)\ o\ x = y\ o\ z = g(y)$

elimination: $\Big\{\big(s(v), s(v)\big)\Big\}$ with $w = s(z)\ o\ x = y\ o\ z = g(y)\ o\ y = s(v)$

decomposition: $\Big\{(v, v)\Big\}$ with $w = s(z)\ o\ x = y\ o\ z = g(y)\ o\ y = s(v)$

deletion: $\Big\{\Big\}$ with $w = s(z)\ o\ x = y\ o\ z = g(y)\ o\ y = s(v)$

deletion: $\Big\{\Big\}$ with $w = s(z)\ o\ x = s(v)\ o\ z = g(s(v))\ ,\ y = s(v)$

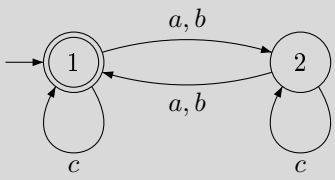
deletion: $\Big\{\Big\}$ with $w = s(g(s(v)))\ o\ x = s(v)\ ,\ z = g(s(v))\ ,\ y = s(v)$

deletion: $\Big\{\Big\}$ with $w = s(g(s(v)))\ ,\ x = s(v)\ ,\ z = g(s(v))\ ,\ y = s(v)$

3. **(6+2+6 points)**

   (a) Design a DFA that accepts all strings w over the alphabet $\{a, b, c\}$ such that the number of a's in w has the same parity as the number of b's in w (both even or both odd).
   Hint: Your DFA could contain a state for when both number of a's and b's are odd, a state for when a's are even and b's are odd, and so on (four possibilities).

Solution:



(b) i. Write a regular expression that generates all strings w over the alphabet $\{a, b, c\}$ such that w has a substring ab.

Solution:

$$(a + b + c)^* ab (a + b + c)^*$$

ii. Write a regular expression that generates all strings w over the alphabet $\{a, b, c\}$ such that w does **not** have a substring ab.

Solution:

$$(b + a^* c)^* a^*$$

4. **(10 points)**

Write an ocamllex specification for tokens of this type:

```
type mytoken = PLUS | ID of string | INT of int
```

where `PLUS` is operator `+`, `ID` is any string of *odd length* made out of letters of english alphabet and digits that starts with a letter, and `INT` is a string of digits. Have your specification to return the next token in the input. You may want to use this function `int_of_string: string -> int`.

As a hint, we have provided the following template for you to complete. Feel free to add as many cases as you want to the `parse` section.

```
rule main = parse
_______________________________  { ___________________________________ }

| _______________________________  { ___________________________________ }

| _______________________________  { ___________________________________ }
```

```
{
 let newlexbuf = (Lexing.from_channel stdin) in
 print_string "starting ...";;
 main newlexbuf
}
```

Solution:

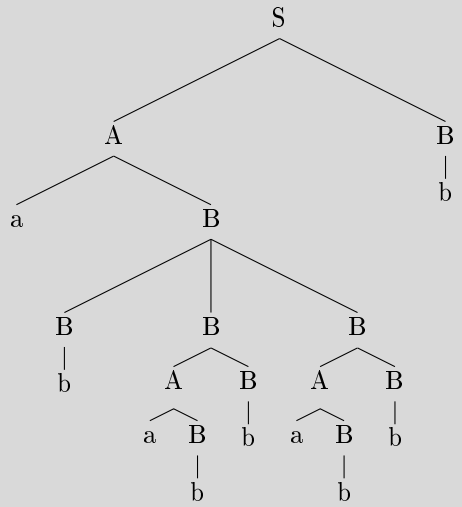
```
rule main = parse
'+'  { PLUS }
|['0'-'9']+ as s { INT int_of_string s }
|(['a'-z','A'-'Z']['a'-z','A'-'Z','0'-'9'] ['a'-z','A'-'Z','0'-'9'])*) as s {ID s}
```

5. **(5+5 points)**
   Consider the following grammar

$$
\begin{aligned}
S &\implies AB \\
A &\implies BAB \mid \texttt{a}B \\
B &\implies AB \mid \texttt{b} \mid BBB
\end{aligned}
$$

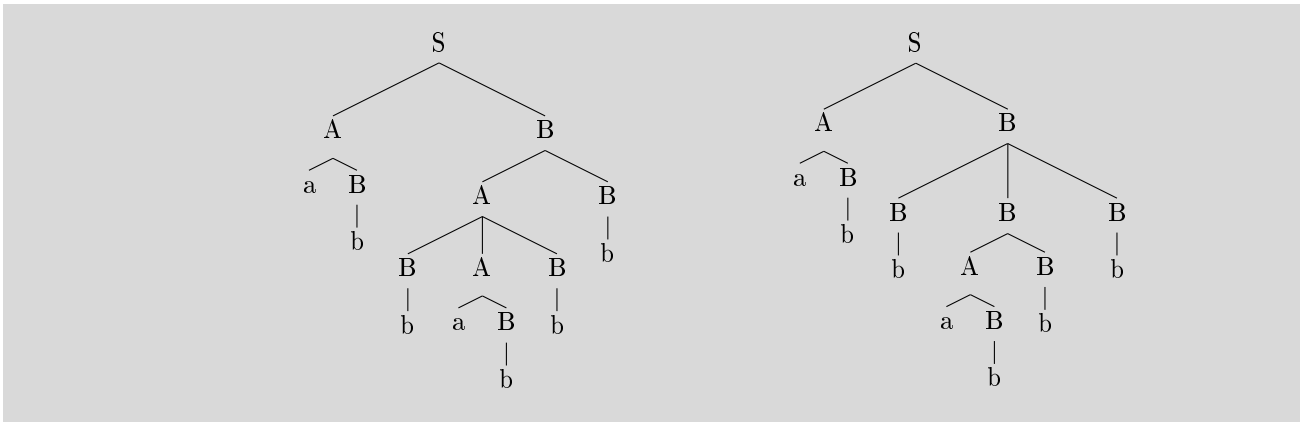
(a) Give a parse tree for *ababbabbb*.

Solution:



(b) Show that this grammar is ambiguous by presenting two parse trees that produce the same string.

Solution:


S
A
B
a
B
A
B
b
B
A
B
b
b
a
B
b
b
S
A
B
a
B
B
B
B
b
b
A
B
b
a
B
b

Rules for type derivations:

(Unit) $\dfrac{}{\Gamma \vdash () \ : \ \mathsf{unit}}$ $\dfrac{}{\Gamma \vdash [\,] \ : \ \tau \ \mathsf{list}}$ (Nil)

(Bool) $\dfrac{}{\Gamma \vdash b \ : \ \mathsf{bool}}$ where $b = \mathsf{true}$ or false

(Int) $\dfrac{}{\Gamma \vdash n \ : \ \mathsf{int}}$ where n is an integer

(Float) $\dfrac{}{\Gamma \vdash n \ : \ \mathsf{float}}$ where n is a floating point number

(Var) $\dfrac{}{\Gamma \vdash x \ : \ \tau}$ where $\Gamma(x) = \tau$

(Pair) $\dfrac{\Gamma \vdash e_1 \ : \ \tau_1 \qquad \Gamma \vdash e_2 \ : \ \tau_2}{\Gamma \vdash (e_1, e_2) \ : \ (\tau_1, \tau_2)}$

(List) $\dfrac{\Gamma \vdash e_h \ : \ \tau \qquad \Gamma \vdash e_t \ : \ \tau \ \mathsf{list}}{\Gamma \vdash e_h :: e_t \ : \ \tau \ \mathsf{list}}$

(Fun) $\dfrac{\Gamma \cup [x \ : \ \tau_x] \vdash e \ : \ \tau_e}{\Gamma \vdash \mathsf{fun}\ x \ \texttt{->}\ e \ : \ \tau_x \rightarrow \tau_e}$

(App) $\dfrac{\Gamma \vdash e_f \ : \ \tau_a \rightarrow \tau_r \qquad \Gamma \vdash e_a \ : \ \tau_a}{\Gamma \vdash e_f \ e_a \ : \ \tau_r}$

(Let) $\dfrac{\Gamma \vdash e_x \ : \ \tau_x \qquad \Gamma \cup [x \ : \ \tau_x] \vdash e \ : \ \tau}{\Gamma \vdash \mathsf{let}\ x = e_x\ \mathsf{in}\ e \ : \ \tau}$

(LetRec) $\dfrac{\Gamma \cup [x \ : \ \tau_x] \vdash e_x \ : \ \tau_x \qquad \Gamma \cup [x \ : \ \tau_x] \vdash e \ : \ \tau}{\Gamma \vdash \mathsf{let\ rec}\ x = e_x\ \mathsf{in}\ e \ : \ \tau}$

(Arithmetic Operators) $\dfrac{\Gamma \vdash e_2 : \texttt{int} \qquad \Gamma \vdash e_1 : \texttt{int}}{\Gamma \vdash e_1 \oplus e_2 : \texttt{int}}\ (\oplus \in \{+, -, *, /, \cdots\})$

(If) $\dfrac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}$

scratch paper

scratch paper