

## Sample Questions for Midterm 2 (CS 421 Spring 2023)

Some of these questions may be reused for the exam.

1. Give a (most general) unifier for the following unification instance. Capital letters denote variables of unification. Show your work by listing the operation performed in each step of the unification and the result of that step.

$$\{X = f(g(x),W); h(y) = Y; f(Z,x) = f(Y,W)\}$$

2. Write a function

**unify\_eliminate** : **typeVar** -> **monoTy** -> (**monoTy** \* **monoTy**) list ->  
((**monoTy** \* **monoTy**) list -> (**typeVar** \* **monoTy**) list option) ->  
(**typeVar** \* **monoTy**) list option

where **unify\_eliminate** given the arguments **ns** is a **typeVar**, **t** is a **monoTy**, **rem\_constraints** is a list of **monoTy** pairs describing a set of equational constraints on **monoTys**, and **unify\_rem** is a function capable of returning a substitution capable of solving **rem\_constraints** if a solution exists, but is not necessarily capable of solving any other set of constraints. When fully applied, **unify\_eliminate** then returns a solution to the larger set of constraints: (**TyVar ns, t**) :: **rem\_constraints**

3. For each of the following descriptions, give a regular expression over the alphabet {a,b,c}, and a right regular grammar that generates the language described.
  - a. The set of all strings over {a, b, c}, where each string has at most one a
  - b. The set of all strings over {a, b, c}, where, in each string, every b is immediately followed by at least one c.
  - c. The set of all strings over {a, b, c}, where every string has length a multiple of four.
4. This problem is too big to be an exam question, but it can be cut down to a few different problems that would be possible. For example, just having strings is a plausible exam problem. I include the whole problem because it poses several difficulties to challenge you, and because the end result is actually useful.

Write an ocamllex file that translate a (simplified) comma separated values (.csv) file into a reversed list of reversed lists of entries, where an entry is an integer, a float, or a string. Entries are represented using the following type:

**type csv\_entry = INT of int | FLOAT of float | STRING of string**

The file will contain newline ended rows of comma separated values, where a value is representation of an integer, float or string. An integer is represented as a nonempty sequence of digits, possibly preceded by a minus sign, where the leading digit is a 0 only if the integer is a 0 and 0 is not preceded by a minus sign. Floats are similar, but with a decimal point. So a float may start with a minus sign but then must have a nonempty sequence of digits, starting with 0 only if the integer part is 0, followed by a period, follow by a nonempty sequence of digits that does not end in a 0. (So, 1. , 1.0, .9 and -.5 are some examples of things that are not legal.) There are two representations of strings. Strings only use printable characters, as described in MP8, but including \ as an ordinary character. One type of string representation is

any (possibly empty) sequence of printable characters not including a double quotes or comma. Its value should be the string that contains that sequence of characters. If the same sequence also represents an integer or a float, it should be translated as the integer or float, not a string. The second type of string representation must begin with and end with a single double quotes. Inside commas may freely be used. Since double quotes end the string, they can't appear inside the string without special support. This time, a double quote inside a string is represents by a pair of double quotes. So "" represents a string that has one character which is a double quotes. On an exam, you would be given some of the contents described here as starter code. A sample contents for a csv file is

```
""",Does,"""",this ,""",?,"""work""",
\n,4,-0.4,33,0.1,0.2
```

Be sure to put only ASCII characters into your test file, or use test\_csv.csv.

5. Consider the following grammar:

```
<S> ::= <A> | <A> <S>
<A> ::= <Id> | ( <B>
<B> ::= <Id> ] | <Id><B> | ( <B>
<Id> ::= 0 | 1
```

For each of the following strings, give a parse tree for the following expression as an <S>, if one exists, or write “No parse” otherwise:

- a. ( 0 1 ( 1 ] ( ( 1 0 ] 1
- b. 0 ( 1 0 ( 1 ]
- c. ( 0 ( 1 0 1 ] 0 ]

6. You are given the following grammar over nonterminal <s>, <b> and <e>, and terminals c, x, l, and eof, with start symbol <s>:

```
P0: <s> ::= <e>eof
P1: <e> ::= c<b><e><e> P2: <e> ::= x
P3: <b> ::= <e>l
```

and Action and Goto tables generated by YACC for the above grammar:

	Action				Goto		
State	x	c	l	[eof]	<b>	<e>	<s>
st1	S3	S4	Err	Err		st2	
st2	Err	Err	Err	A			
st3	R2	R2	R2	R2			
st4	S3	S4	Err	Err	st6	st5	

