CS421 Fall 2014 Midterm 2 Solution

Name:	
NetID:	

- You have **75 minutes** to complete this exam.
- This is a **closed-book** exam. All materials (e.g., calculators, cell phones and scrap paper), except writing utensils are prohibited.
- Do not share anything with other students. Do not talk to other students. Do not look at another students exam. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.
- If you believe there is an error, or an ambiguous question, you may seek clarification from myself or one of the TAs. You must use a whisper, or write your question out. Speaking out aloud is not allowed.
- Including this cover sheet and rules at the end, there are 20 pages to the exam, including two blank pages for workspace. Please verify that you have all 20 pages.
- Please write your name and NetID in the spaces above, and also in the provided space at the top of every sheet.

Question	Points	Bonus Points	Score
1	16	0	
2	16	0	
3	14	0	
4	13	0	
5	16	0	
6	25	0	
7	0	10	
Total:	100	10	

Problem 1. (16 points)

Below are a serries of partial type deriviations, some of which contain errors. For each one select whether all steps of the derivation present are correct, and if not, circle at least one error with the derivation. There may be more than one error, but you only need to circle one.

(a) (4 points)

$$\dfrac{\dfrac{\dfrac{}{\{\texttt{x:int}\} \vdash \texttt{(+):int -> int -> int}}\textsc{Const} \quad \dfrac{}{\{\texttt{x:int}\} \vdash \texttt{x:int}}\textsc{Var}}{\{\texttt{x:int}\} \vdash \texttt{(+)x:int -> int}}\textsc{App} \quad \dfrac{}{\boxed{\{\texttt{y:int}\}} \vdash \texttt{y:int}}\textsc{Var}}{\{\texttt{x:int}\} \vdash \texttt{(x+y):int}}\textsc{App}}{\{\texttt{x:bool}\} \vdash \texttt{(fun x -> x+y):int -> int}}\textsc{Fun}$$

Select one:

◯ The proof is correct, as much as is shown

√ **The proof is incorrect and I have circled an error**

(b) (4 points)

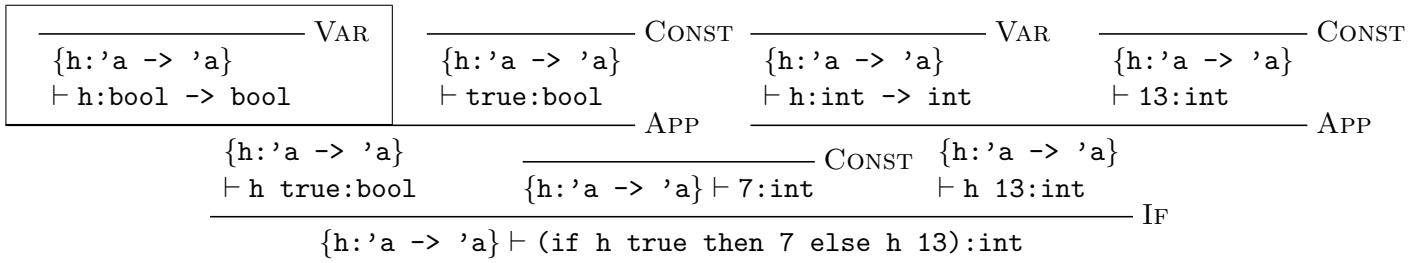
$$\dfrac{\dfrac{\dfrac{}{\{\texttt{x:bool}\} \vdash \texttt{(fun x -> x::[])} : \texttt{'a -> 'a list}} \quad \dfrac{\dfrac{\dfrac{}{\{\texttt{x:bool, s:}\forall\texttt{'a -> 'a list}\} \vdash \texttt{s:bool list -> bool list list}}\textsc{Var}\ \text{with } [\texttt{bool list/'a}]}{\{\texttt{x:bool, s:}\forall\texttt{'a -> 'a list}\} \vdash \texttt{s:bool -> bool list}}\textsc{Var}\ \text{with } [\texttt{bool/'a}] \quad \dfrac{}{\{\texttt{x:bool, s:}\forall\texttt{'a -> 'a list}\} \vdash \texttt{x:bool}}\textsc{Var}}{\{\texttt{x:bool, s:}\forall\texttt{'a -> 'a list}\} \vdash \texttt{s x:bool list}}\textsc{App}}{\{\texttt{x:bool, s:}\forall\texttt{'a -> 'a list}\} \vdash \texttt{s(s x):bool list list}}\textsc{App}}{\{\texttt{x:bool}\} \vdash \texttt{let s = fun x -> x::[] in s(s x):bool list list}}\textsc{Let}$$

Select one:

√ **The proof is correct, as much as is shown**

◯ The proof is incorrect and I have circled an error

(c) (4 points)

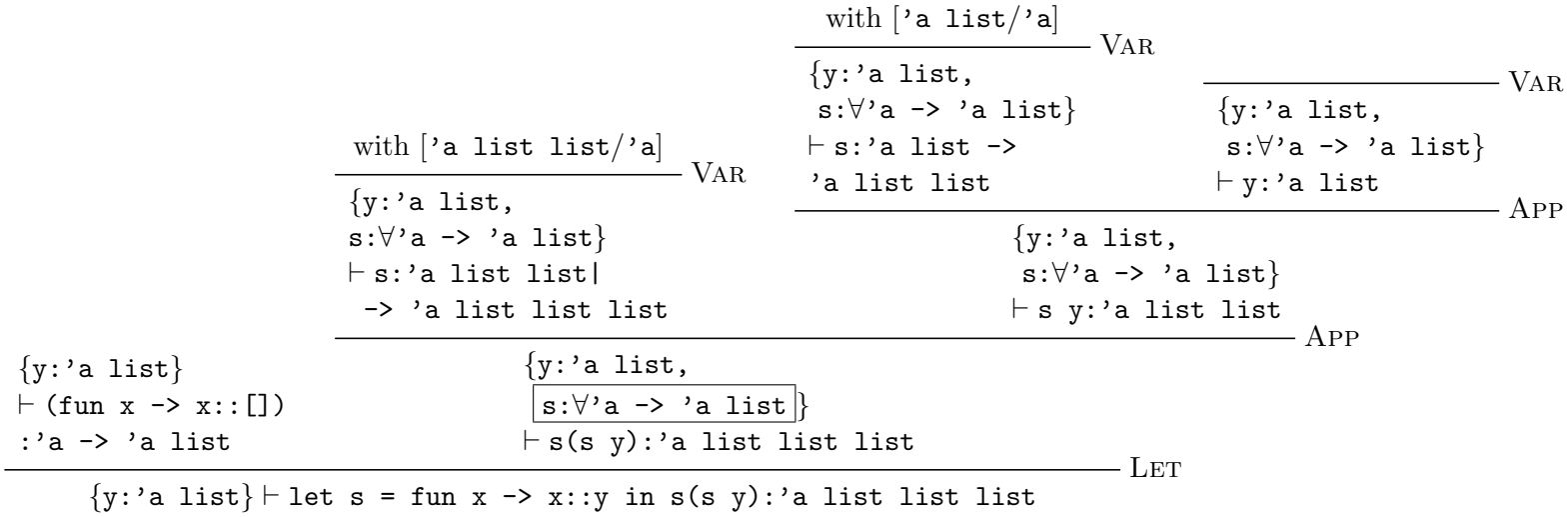


Select one:

◯ The proof is correct, as much as is shown

√ **The proof is incorrect and I have circled an error**

(d) (4 points)



Select one:

◯ The proof is correct, as much as is shown

√ **The proof is incorrect and I have circled an error**

Problem 2. (16 points)

Use the unification algorithm described in class and in MP7 to answer which of these given equations holds for the stated reason, and when it diesn't, to indicate why not. In this problem, we use = as the separator for constraints. The uppercase letters X, Y, and Z, denote variables of unification. The lowercase letters g, f, p, and l are term constructors of arity 3, 2, 2, and 1 respectively (*i.e.* take three, two, two or one argument(s), respectively). The letters b and i are constants (constructor of arity 0). For each problem you are asked to check if the equation is correct for the stated reason(s), and if not, briefly why not. For a reason why not, you may underline a portion and write in the given space what it should have been if this will yield an approrpiate reason.

(a) (2 points)

$\mathsf{Unify}\{(l(Z) = l(f(X,X))); (g(f(X,p(A,A)),p(X,Y),b) = g(Z,p(Y,X),b))\}$
$= \mathsf{Unify}\{(Z = f(X,X)); (g(f(X,\underline{p(X,X)}),p(X,Y),b) = g(Z,p(Y,X),b))\}$
by Decompose with $(l(Z) = l(f(\overline{X,X})))$

Select one:

◯ The step is correct for the stated reason

√ **The right-hand side does not equal the left-hand side for the stated reason because**

(a) $p(A,A)$ **should have been** $p(X,X)$

(b) (2 points)

$\mathsf{Unify}\{(Z = f(X,X)); (g(f(X,p(X,X)),p(X,Y),b) = g(Z,p(Y,X),b))\}$
$= \mathsf{Unify}\{(g(f(X,p(X,X)),p(X,Y),b) = g(f(X,X),p(Y,X),b))\} o \{Z \mapsto f(X,X)\}$
by Eliminate with $(Z = f(X,X))$

Select one:

√ **The step is correct for the stated reason**

◯ The right-hand side does not equal the left-hand side for the stated reason because

(b) _____

(c) (2 points)

$\mathsf{Unify}\{(g(f(X,p(X,X)),p(X,Y),b) = g(f(X,X),p(Y,X),b))\}o\{Z \mapsto f(X,X)\}$
$= \mathsf{Unify}\{(f(X,p(X,X)) = f(X,X)); (p(X,Y) = p(Y,X))\}o\{Z \mapsto f(X,X)\}$
by Decompose with $(g(f(X,p(X,X)),p(X,Y),b) = g(f(X,X),p(Y,X),b))$

Select one:

◯ The step is correct for the stated reason

√ **The right-hand side does not equal the left-hand side for the stated reason because**

(c) **The second part is missing the equation** $(b = b)$

(d) (2 points)

$= \mathsf{Unify}\{(f(X,p(X,X)) = f(X,X)); (p(X,Y) = p(Y,X))\}o\{Z \mapsto f(X,X)\}$
$\mathsf{Unify}\{(X = X); (p(X,X) = X); (p(X,Y) = p(Y,X))\}o\{Z \mapsto f(X,X)\}$
by Decompose with $(f(X,p(X,X)) = f(X,X))$

Select one:

√ **The step is correct for the stated reason**

◯ The right-hand side does not equal the left-hand side for the stated reason because

(d) _____

(e) (2 points)

$\mathsf{Unify}\{(X = X); (p(X,X) = X); (p(X,Y) = p(Y,X))\}o\{Z \mapsto f(X,X)\}$
$= \mathsf{Unify}\{(p(X,X) = X); (p(X,Y) = p(Y,X))\}o\{X \mapsto X\}o\{Z \mapsto f(X,X)\}$
by Eliminate with $(X = X)$

Select one:

◯ The step is correct for the stated reason

√ **The right-hand side does not equal the left-hand side for the stated reason because**

(e) **The reason is wrong. It should be "by Delete** $(X = X)$**".**

(f) (2 points)

$\mathsf{Unify}\{(p(X,X) = X); (p(X,Y) = p(Y,X))\} o \{Z \mapsto f(X,X)\}$
$= \mathsf{Unify}\{\underline{(X = p(X,X))}; (X = Y); (Y = X)\} o \{Z \mapsto f(X,X)\}$
by Decompose with $(p(X,Y) = p(Y,X))$

Select one:

◯ The step is correct for the stated reason

√ **The right-hand side does not equal the left-hand side for the stated reason because**

(f) $(p(X,X) = X)$ **can not be swapped around in Decompose.**

(g) (2 points)

$\mathsf{Unify}\{(X = p(X,X)); (X = Y); (Y = X)\} o \{Z \mapsto f(X,X)\}$
$= \mathsf{Unify}\{(Y = p(Y,Y))\} o \{Z \mapsto f(X,X)\} o \{X \mapsto Y\}$
by Eliminate with $\underline{(p(X,Y) = p(Y,X))}$

Select one:

◯ The step is correct for the stated reason

◯ The right-hand side does not equal the left-hand side for the stated reason because

(g) $(p(X,Y) = p(Y,X))$ **doesn't exist in the constraint set, and it isn't appropriate for Eliminate**

(h) (2 points)

$\mathsf{Unify}\{(Y = p(Y,Y))\} o \{Z \mapsto f(Y,Y); X \mapsto Y\}$
$= \{Z \mapsto f(p(Y,Y), p(Y,Y)); X \mapsto p(Y,Y); Y \mapsto p(Y,Y)\}$by Eliminate with $(Y = p(Y,Y))$

Select one:

◯ The step is correct for the stated reason

√ **The right-hand side does not equal the left-hand side for the stated reason because**

(h) $Y = p(Y,Y)$ **fails the occurs check test for Eliminate.**

Problem 3. (14 points)

The code given for MP7 in the `Mp7common` module includes the following data types to represent the types of PicoML and type substitutions:

```
type typeVar = int
type monoTy = TyVar of typeVar | TyConst of (string * monoTy list)
type substitution = (typeVar * monoTy) list
```

(a) (6 points) Implement the function `subst_fun:substitution -> typeVar -> monoTy` that takes a substitution as a list and returns a function that takes a type variable as input and returns the replacement type as given by the substitution. If no substitute `monoTy` is given in the substitution for the type variable, then the `monoTy` corresponding to the type variable is returned. You may use Library functions, but you must use them correctly for credit.

Solution:

```
let rec subst_fun subst m =
    match subst with [] -> TyVar m
    | (n,ty) :: more -> if n = m then ty else subst_fun more m
```

(b) (8 points) A substitution ϕ, when lifted to a `monoTy`, replaces all the type variables occurring in the `monoTy` with the corresponding `monoTy`s. Implement the function `monoTy_lift_subst:substitution -> monoTy -> monoTy` for lifting substitutions to generic `monoTy`s.

Solution:

```
let rec monoTy_lift_subst subst monoTy =
    match monoTy
    with TyVar m -> subst_fun subst m
    | TyConst(c, typelist) ->
      TyConst(c, List.map (monoTy_lift_subst subst) typelist)
```

Problem 4. (13 points)

Consider the set of all strings over the alphabet { 0, 1, [,], ; } that represent OCaml lists of 0's and 1's, where a list in OCaml begins with a single left square bracket ([),followed by a possibly empty sequence of 0's and 1's separated by single semicolons and terminated by a single right square bracket. There is no semicolon following the last digit in the list.

(a) (6 points) Write a regular expression describing the set of binary list given above. In writing a regular expression describing this set of strings, you may use the notation for basic regular expressions (Kleene's notation), or you may use omcallex syntax, but these are the only syntax allowed.

Solution:

$$([\,]) \vee ([(0 \vee 1)((;(0 \vee 1))^*)])$$

(b) (7 points) Write a (right) regular grammar describing the same set of strings.

Solution:

$$\begin{aligned} < list > ::= & [< first > \\ < first > ::= &] \\ & \mid 0 < semi\_or\_done > \\ & \mid 1 < semi\_or\_done > \\ < semi\_or\_done > ::= &] \\ & \mid ; < bit > \\ < bit > ::= & 0 < semi\_or\_done > \\ & \mid 1 < semi\_or\_done > \end{aligned}$$

Problem 5. (16 points)

Given the following BNF grammar, for each of the following strings, give a parse tree for it, if it parses starting with $< ty >$, or write None exists if it does not parse starting with $< ty >$. The terminals for this grammar are { `a`, `b`, `int`, `list`, `*`, `(`, `)` }. The non-terminal are $< ty >$, $< m >$, and $< at >$.

$< ty >$::= $< at >$ | $< at >< m >$
$< m >$::= `*` $< ty >$ | `list` $< m >$ | `list`
$< at >$::= `a` | `b` | `int` | `(` $< ty >$ `)`

(a) (5 points) `int list int * int list`

Solution: None exists. An invariant of the grammar is that the only thing that can come right before an `int` is a `(` or nothing.

Given the following BNF grammar, for each of the following strings, give a parse tree for it, if it parses starting with $< ty >$, or write None exists if it does not parse starting with $< ty >$. The terminals for this grammar are { `a`, `b`, `int`, `list`, `* (`, `)` }. The non-terminal are $< ty >$, $< m >$, and $< at >$.

$$
\begin{array}{lll}
< ty > & ::= & < at > \mid < at > < m > \\
< m > & ::= & \texttt{*} < ty > \mid \texttt{list} < m > \mid \texttt{list} \\
< at > & ::= & \texttt{a} \mid \texttt{b} \mid \texttt{int} \mid \texttt{(} < ty > \texttt{)}
\end{array}
$$

(b) (5 points) `b * ( int * list ) * a list`

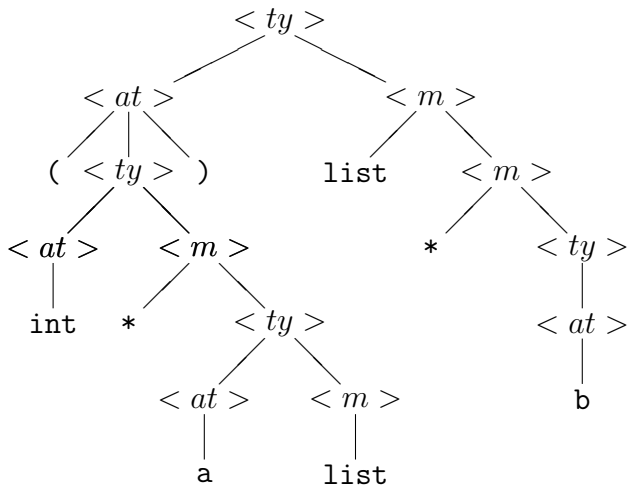
Solution: None exists. `list` is a $< m >$, and a $< m >$ can only be preceded by an $< at >$ and no $< at >$ can end in a `*`.

Given the following BNF grammar, for each of the following strings, give a parse tree for it, if it parses starting with $< ty >$, or write None exists if it does not parse starting with $< ty >$. The terminals for this grammar are { `a`, `b`, `int`, `list`, `*`, `(`, `)` }. The non-terminal are $< ty >$, $< m >$, and $< at >$.

$< ty >$::= $< at >$ | $< at >< m >$
$< m >$::= `*` $< ty >$ | `list` $< m >$ | `list`
$< at >$::= `a` | `b` | `int` | `(` $< ty >$ `)`

(c) (6 points) `( int * a list ) list * b`

Solution: This time one does exist.



Problem 6. (25 points)

Consider the following grammar over the terminal alphabet {`int`, `list`, $*$, `'a`, `'b`, (,)} and non-terminal alphabet {`<ty>`, `<var>`}:

$$< \texttt{ty} > \ ::= \ \ int \mid < ty > \ * \ < ty > \ \mid < ty > \ list \mid (< ty >)$$

Let $\mathcal{L}$ be the language generated by `<ty>`. You will be presented with a collection of grammars over the same terminal alphabet with start symbol `<ty>`, and you are ask to determine which of a collection of properties the new grammar has.

(a) (5 points) Nonterminals: {`<ty>`, `<r>`, `<at>`}, Grammar:

$$\begin{array}{lll} < \texttt{ty} > & ::= & < \texttt{r} > \mid < \texttt{ty} > \texttt{list} \\ < \texttt{r} > & ::= & < \texttt{at} > \mid < \texttt{at} > * < \texttt{r} > \mid < \texttt{at} > \ \texttt{list} * < \texttt{r} > \\ < \texttt{at} > & ::= & \texttt{int} \mid (< \texttt{ty} >) \end{array}$$

- The grammar is unambiguous
  √ **True** ◯ False
- Every string the grammar parses is in $\mathcal{L}$
  √ **True** ◯ False
- The grammar parses every string in $\mathcal{L}$
  ◯ True √ **False**
- The grammar fails to make * always associate to the right or fails to always to make * always have higher precedence than *list*
  √ **True** ◯ False
- The grammar fails to parse at least one of {`int list list`, `int * int list list * int`}
  √ **True** ◯ False

(b) (5 points) Nonterminals: {`<ty>`, `<n>`, `<at>`}, Grammar:

$$\begin{array}{lll} < \texttt{ty} > & ::= & < \texttt{n} > \mid < \texttt{ty} > \ \texttt{list} \mid < \texttt{ty} > \ \texttt{list} * < \texttt{n} > \\ < \texttt{n} > & ::= & < \texttt{at} > \mid < \texttt{at} > \ * \ < \texttt{n} > \\ < \texttt{at} > & := & \texttt{int} \mid (< \texttt{ty} >) \end{array}$$

- The grammar is unambiguous
  √ **True** ◯ False
- Every string the grammar parses is in $\mathcal{L}$
  √ **True** ◯ False
- The grammar parses every string in $\mathcal{L}$
  √ **True** ◯ False
- The grammar fails to make * always associate to the right or fails to always to make * always have higher precedence than *list*
  ◯ True √ **False**
- The grammar fails to parse at least one of {`int list list`, `int * int list list * int`}
  ◯ True √ **False**

(c) (5 points) Nonterminals: {<ty>, <p>, <r>, <at>}, Grammar:

$$
\begin{array}{lcl}
< \texttt{ty} > & ::= & < p > \mid < p > \mathit{list} \\
< \texttt{p} > & ::= & < \texttt{at} > \mid < \texttt{p} > \texttt{list} \mid < \texttt{at} > * < \texttt{r} > \mid < \texttt{p} > \texttt{list} * < \texttt{r} > \\
< \texttt{r} > & ::= & < \texttt{at} > \mid < \texttt{at} > * < \texttt{r} > \\
< \texttt{at} > & := & \texttt{int} \mid (< \texttt{ty} >)
\end{array}
$$

- The grammar is unambiguous
  ◯ True √ **False**
- Every string the grammar parses is in $\mathcal{L}$
  √ **True** ◯ False
- The grammar parses every string in $\mathcal{L}$
  √ **True** ◯ False
- The grammar fails to make * always associate to the right or fails to always to make * always have higher precedence than *list*
  ◯ True √ **False**
- The grammar fails to parse at least one of {`int list list`, `int * int list list * int`}
  ◯ True √ **False**

(d) (5 points) Nonterminals: {<ty>, <p>, <at>}, Grammar:

$$
\begin{array}{lcl}
< \texttt{ty} > & ::= & < at > \mathit{list} \mid < p > \mid < p > * < at > \mathit{list} \\
< \texttt{p} > & ::= & < p > * < at > \mid < at > \\
< \texttt{at} > & := & \texttt{int} \mid (< \texttt{ty} >)
\end{array}
$$

- The grammar is unambiguous
  √ **True** ◯ False
- Every string the grammar parses is in $\mathcal{L}$
  √ **True** ◯ False
- The grammar parses every string in $\mathcal{L}$
  ◯ True √ **False**
- The grammar fails to make * always associate to the right or fails to always to make * always have higher precedence than *list*
  √ **True** ◯ False
- The grammar fails to parse at least one of {`int list list`, `int * int list list * int`}
  √ **True** ◯ False

(e) (5 points) Nonterminals: {<ty>, <n>, <p>, <at>}, Grammar:

$$
\begin{array}{lcl}
< \texttt{ty} > & ::= & < n > \mid < n > * < p > \\
< \texttt{p} > & ::= & < \texttt{at} > \mid < \texttt{at} > * < \texttt{p} > \\
< \texttt{n} > & ::= & < \texttt{at} > \mid < \texttt{ty} > \texttt{list} \\
< \texttt{at} > & := & \texttt{int} \mid (< \texttt{ty} >)
\end{array}
$$

- The grammar is unambiguous
  √ **True** ◯ False
- Every string the grammar parses is in $\mathcal{L}$
  √ **True** ◯ False
- The grammar parses every string in $\mathcal{L}$
  √ **True** ◯ False
- The grammar fails to make * always associate to the right or fails to always to make * always have higher precedence than *list*
  ◯ True √ **False**
- The grammar fails to parse at least one of {`int list list`, `int * int list list * int`}
  ◯ True √ **False**

Workspace

7. (10 points (bonus)) Using recursive descent parsing and the grammar you gave in Problem 4.b, write a function

```
to_list :  char list -> (int list option * char list)
```

that returns the `int list` that is the largest `int list` represented by the front of the input `char list`, paired with the characters not used. Your function should return `None` if there is no list (not even the empty list) represented by the characters at the front of the input list.

Solution: The grammar in 4b may be presented as

$$\begin{aligned} < list > &::= [< first > \\ < first > &::=] \mid ((0 \mid 1) < semi\_or\_done >) \\ < semi\_or\_done > &::=] \mid ; < bit > \\ < bit > &::= (0 \mid 1) < semi\_or\_done > \end{aligned}$$

```
let add_bit c res =
    (match res with (None, chs) -> res
     | (Some lst, chs) ->
        (match c with '0' -> (Some (0::lst), chs)
         | '1' -> (Some (1::lst), chs)))

let rec to_list chrlist =
  match chrlist with [] -> (None, chrlist)
  | ('['::remchl) -> first remchl
  | _ -> (None, chrlist)
and first chrlist =
  match chrlist with [] -> (None, chrlist)
  | (c::remchl) ->
     (match c with ']' -> (Some [], remchl)
      | _ -> if c = '0' || c = '1'
                then add_bit c (semi_or_done remchl)
             else (None, chrlist))
and semi_or_done chrlist =
    match chrlist with (']'::remchl) -> (Some [], remchl)
    | (';'::remchl) -> bit remchl
    | _ -> (None, chrlist)
and bit chrlist =
    match chrlist with [] -> (None, chrlist)
    | (c::remchl) -> if c = '0' || c = '1'
                        then add_bit c (semi_or_done remchl)
                     else (None, chrlist)
```

Workspace

A Polymoprhic Typing Rules

Polymorphic constant signatures:

$\mathsf{sig}(n) = \texttt{int}$ n an integer constant $\quad \mathsf{sig}(\oplus) = \texttt{int} \to \texttt{int} \to \texttt{int}$ for $\oplus \in \{+, -, *, \% \ldots\}$
$\mathsf{sig}(\texttt{true}) = \texttt{bool} \quad \mathsf{sig}(\texttt{false}) = \texttt{bool}$
$\mathsf{sig}(\sim) = \forall\alpha.\, \alpha \to \alpha \to \texttt{bool}$ for $\sim \in \{<, >, =, \leq, \geq\}$
$\mathsf{sig}(\texttt{[ ]}) = \forall\alpha.\, \alpha\texttt{list} \quad \mathsf{sig}((::)) = \forall\alpha.\, \alpha \to \alpha\ \texttt{list} \to \alpha\ \texttt{list}$
$\mathsf{sig}((\ ,\ )) = \forall\alpha\beta.\, \alpha \to \beta \to \alpha * \beta$

Constants:

$$\frac{}{\Gamma \vdash\ c\ :\ \tau'}\ \textsc{Const}$$

where c is a constant listed above, $\mathsf{sig}(c) = \forall\alpha_1 \ldots \alpha_n.\, \tau$ and there exist $\sigma_1, \ldots, \sigma_n$ such that $\tau' = \tau[\sigma_1/\alpha_1; \ldots; \sigma_n/\alpha_n]$

Variables:

$$\frac{}{\Gamma \vdash x : \tau'}\ \textsc{Var}$$

where $\forall\alpha_1 \ldots \alpha_n.\, \tau = \Gamma(x)$ and there exist $\sigma_1, \ldots, \sigma_n$ such that $\tau' = \tau[\sigma_1/\alpha_1; \ldots; \sigma_n/\alpha_n]$

Connectives:

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash e_1\ \texttt{\&\&}\ e_2 : \texttt{bool}}\ \textsc{Conn} \qquad \frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash e_1\ \texttt{||}\ e_2 : \texttt{bool}}\ \textsc{Conn}$$

If_then_else rule:

$$\frac{\Gamma \vdash e_c : \texttt{bool} \quad \Gamma \vdash e_t : \tau \quad \Gamma \vdash e_e : \tau}{\Gamma \vdash \texttt{if}\ e_c\ \texttt{then}\ e_t\ \texttt{else}\ e_e : \tau}\ \textsc{If}$$

Application rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ \textsc{App}$$

Function rule:

$$\frac{[x : \tau_1] + \Gamma \vdash e : \tau_2}{\Gamma \vdash \texttt{fun}\ x\ \texttt{->}\ e : \tau_1 \to \tau_2}\ \textsc{Fun}$$

Let rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad [x : \mathsf{Gen}(\tau_1, \Gamma)] + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \tau_2}\ \textsc{Let}$$

Let Rec rule:

$$\frac{[x : \tau_1] + \Gamma \vdash e_1 : \tau_1 \quad [x : \mathsf{Gen}(\tau_1, \Gamma)] + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let rec}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \tau_2}\ \textsc{Rec}$$