

198:515 Programming Languages and Compilers I

Problem Set 4

This homework will not be graded. A sample solution will be made available at a later time.

Problem 2 - Typing

In lecture 9, page 21, we introduced type inference rules for the polymorphic `cons` function and the polymorphic empty list `nil`, together with their representation as type expressions.

Polymorphic `cons`:

$$\frac{E \vdash e_1 : \alpha \quad E \vdash e_2 : list(\alpha)}{E \vdash (cons\ e_1\ e_2) : list(\alpha)}$$

`cons` has the type expression $\forall\alpha.(\alpha \times list(\alpha)) \rightarrow list(\alpha)$

Polymorphic `nil`:

$$E \vdash nil : list(\alpha)$$

`nil` has the type expression $\forall\alpha.list(\alpha)$

Abstraction `lambda(x) e`:

$$\frac{E \cup \{x : \alpha\} \vdash e : \beta \quad E \vdash x : \alpha}{E \vdash (lambda(x)e) : \alpha \rightarrow \beta}$$

1. Give the type inference rules for polymorphic `car` and `cdr`, and their type expressions.

Polymorphic `car`:

$$\frac{E \vdash e : list(\alpha)}{E \vdash (car\ e) : \alpha}$$

`car` has the type expression $\forall\alpha.list(\alpha) \rightarrow \alpha$

Polymorphic `cdr`:

$$\frac{E \vdash e : list(\alpha)}{E \vdash (cdr\ e) : list(\alpha)}$$

`cdr` has the type expression $\forall\alpha.list(\alpha) \rightarrow list(\alpha)$

2. Use the inference rules from the lecture and your inference rules to construct the type of

(a) `(car (cdr (cons 1 (cons 2 nil))))`

$$\frac{E \vdash 2 : \text{integer} \quad E \vdash \text{nil} : \text{list}(\text{integer})}{E \vdash (\text{cons } 2 \text{ nil}) : \text{list}(\text{integer})}$$

$$\frac{E \vdash 1 : \text{integer} \quad E \vdash (\text{cons } 2 \text{ nil}) : \text{list}(\text{integer})}{E \vdash (\text{cons } 1 (\text{cons } 2 \text{ nil})) : \text{list}(\text{integer})}$$

$$\frac{E \vdash (\text{cons } 1 (\text{cons } 2 \text{ nil})) : \text{list}(\text{integer})}{E \vdash (\text{cdr}(\text{cons } 1 (\text{cons } 2 \text{ nil}))) : \text{list}(\text{integer})}$$

$$\frac{E \vdash (\text{cdr}(\text{cons } 1 (\text{cons } 2 \text{ nil}))) : \text{list}(\text{integer})}{E \vdash (\text{car}(\text{cdr}(\text{cons } 1 (\text{cons } 2 \text{ nil})))) : \text{integer}}$$

(b) `(lambda (x) (cdr (cdr (car x))))`

$$\frac{E \vdash x : \text{list}(\alpha)}{E \vdash (\text{car } x) : \alpha}$$

Constraint: $\alpha = \text{list}(\beta)$

$$\frac{E \vdash (\text{car } x) : \text{list}(\beta)}{E \vdash (\text{cdr } (\text{car } x)) : \text{list}(\beta)}$$

$$\frac{E \vdash (\text{cdr } (\text{car } x)) : \text{list}(\beta)}{E \vdash (\text{cdr } (\text{cdr } (\text{car } x))) : \text{list}(\beta)}$$

$$\frac{E \cup \{x : \text{list}(\text{list}(\beta))\} \vdash (\text{cdr } (\text{cdr } (\text{car } x))) : \text{list}(\beta) \quad E \vdash x : \text{list}(\text{list}(\beta))}{E \vdash (\text{lambda}(x)(\text{cdr } (\text{cdr } (\text{car } x)))) : \text{list}(\text{list}(\beta)) \rightarrow \text{list}(\beta)}$$

(c) `(lambda (x) (cons (cdr x) nil))`

$$\frac{E \vdash x : \text{list}(\alpha)}{E \vdash (\text{cdr } x) : \text{list}(\alpha)}$$

$$\frac{E \vdash (\text{cdr } x) : \text{list}(\alpha) \quad E \vdash \text{nil} : \text{list}(\text{list}(\alpha))}{E \vdash (\text{cons } (\text{cdr } x) \text{ nil}) : \text{list}(\text{list}(\alpha))}$$

$$\frac{E \cup \{x : \text{list}(\alpha)\} \vdash (\text{cons } (\text{cdr } x) \text{ nil}) : \text{list}(\text{list}(\alpha)) \quad E \vdash x : \text{list}(\alpha)}{E \vdash (\text{lambda}(x)(\text{cons } (\text{cdr } x) \text{ nil})) : \text{list}(\alpha) \rightarrow \text{list}(\text{list}(\alpha))}$$

Problem 3 - Static vs. Dynamic Typing

Assume that A is declared as follows:

```
A: array(1:100) of integer
```

1. State a condition under which a compiler is not able to prove that the array reference A(k) will be in bound, i.e., that no out-of-bound access is possible.

ANSWER: If k is unknown at compile time then a compiler will not be able to prove that the array reference A(k) is in bound. For example, k is an input parameter of a function in which A(k) is accessed.

2. Give the ILOC code that a compiler needs to generate in order to ensure that reference A(k) will not result in an out-of-bound access.

ANSWER:

```
    // Assume register "r0" contains the address of "k".
    loadI 1      => r1
    loadI 100    => r2
    load r0     => r3
    cmp_LT r3, r1 => r4
    cbr r4      => L1, L2
L2: cmp_LT r2, r3 => r5
    cbr r5      => L1, L3
L3: nop
    // Access A(k)
    br L4
L1: nop
    // Out-of-bound error
    br L-exit
L4: nop
```