

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 1 - let and let* in TINY

1. How can the `let` and `let*` constructs be expressed in the TINY language? In other words, can you rewrite a `let` and `let*` with TINY language constructs? If so, specify how to do this.
2. Modify the statically scoped closure interpreter available on the ilab cluster (`~uli/cs515/examples/scheme/ValueStatic.ss`) to include a `let` expression in TINY:

$x \in Variables$	
$n \in Integers$	
$c ::= n \mid \#t \mid \#f \mid + \mid - \mid * \mid /$	constants
$v ::= c \mid (\text{lambda } (x \dots) e)$	values
$e ::= v \mid x \mid (e e_1 \dots e_k) \mid (\text{if } e_1 e_2 e_3) \mid (\text{let } ((x e_1) e_2))$	expressions
$p ::= e$	program

You will need to change multiple components of the TINY interpreter, including the parser, unparser, and `ev` functions.

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.

1. Give the corresponding type inferences rules for polymorphic `car` and `cdr`, and their type expressions.
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))))`
 - (b) `(lambda (x) (cdr (cdr (car x))))`
 - (c) `(lambda (x) (cons (cdr x) nil))`

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 at compile time is able to prove that the array reference A(k) will be in bound, i.e., that no out-of-bound access is possible (static typing).
2. For the general case, 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 (dynamic typing).