

Problem Set 8: Types

Problem 1

We can define the Scheme syntax of the simply typed lambda calculus with one built-in type `Int` using the following BNF specification:

```
<Exp> ::= (<Identifier> <Type>)
```

```
(t-id id type)
```

```
| ((lambda (<Identifier> <Type>) <Exp>) <Type>)
```

```
(t-abs id id-type body type)
```

```
| ((<Exp> <Exp>) <Type>)
```

```
(t-app ftn arg type)
```

```
<Type> ::= Int
```

```
t-int
```

```
| (<Type> -> <Type>)
```

```
(t-ftn op arg)
```

The meta-symbol `<Identifier>` stands for a Scheme symbol that is different from the symbol `lambda`.

We can think of every typed lambda calculus term as a pair of a value (of the untyped lambda calculus) and a tag to indicate its type, written `(exp type)`.

- Using the `define-datatype` abstraction, define the abstract syntax of data types `Exp` and `Type` according to the BNF specification given above. Use as variant names the name given in the boxes.
- Define the procedure `(parse-exp lst)` that when applied to a list representation of a typed lambda calculus expression will return its abstract syntax tree. Apply the principle "Follow the Grammar". That is, you need to define a procedure for every syntactic category.
- Given a typed lambda calculus expression $e \in \langle \text{Exp} \rangle$ define the predicate `type-annotations-consistent?`, which when applied to e returns `#t` if all

type annotations in e are consistent. Otherwise, the predicate returns $\#f$. You have to implement a case analysis using the `cases` statement:

- o $e == (\nu t) : \#t$
- o $e == ((\text{lambda } (x\ t_1) b) t_2)$:
 - t_2 is a function type &&
 - $t_1 ==$ argument type of t_2 &&
 - $(\text{type-annotations-consistent? } b)$ &&
 - type of $b ==$ result type of t_2
- o $e == ((\text{ftn } \text{arg}) t)$:
 - $(\text{type-annotations-consistent? } \text{ftn})$ &&
 - $(\text{type-annotations-consistent? } \text{arg})$ &&
 - type of ftn is a function type &&
 - argument type of type of $\text{ftn} ==$ type of arg &&
 - $t ==$ result type of type of ftn

Examples:

```
> (define var1 '(y ((Int -> (Int -> Int)) -> (Int -> Int))))
> (define abs1 '((lambda (x (Int -> Int)) (x (Int -> Int)))
                ((Int -> Int) -> (Int -> Int))) )
> (define appl1 '(((lambda (x (Int -> Int)) (x (Int -> Int)))
                  ((Int -> Int) -> (Int -> Int)))
  (y (Int -> Int)))
  (Int -> Int) ) )
> (define applE1 '((x (Int -> Int)) (y (Int -> Int))) Int) )
> (parse-exp var1)
(t-id y
  (t-ftn
    (t-ftn (t-int) (t-ftn (t-int) (t-int)))
    (t-ftn (t-int) (t-int))))
> (parse-exp abs1)
(t-abs
  x
  (t-ftn (t-int) (t-int))
  (t-id x (t-ftn (t-int) (t-int)))
  (t-ftn (t-ftn (t-int) (t-int)) (t-ftn (t-int) (t-int))))
> (parse-exp appl1)
(t-app
  (t-abs
    x
```

```
(t-ftn (t-int) (t-int))
(t-id x (t-ftn (t-int) (t-int)))
(t-ftn (t-ftn (t-int) (t-int)) (t-ftn (t-int) (t-int)))
(t-id y (t-ftn (t-int) (t-int)))
(t-ftn (t-int) (t-int)))

> (parse-exp applE1)
(t-app
 (t-id x (t-ftn (t-int) (t-int)))
 (t-id y (t-ftn (t-int) (t-int)))
 (t-int))

> (type-annotations-consistent? (parse-exp var1))
#t
> (type-annotations-consistent? (parse-exp abs1))
#t
> (type-annotations-consistent? (parse-exp appl1))
#t
> (type-annotations-consistent? (parse-exp applE1))
#f
```

Submission deadline: Thursday, April 26, 2007, 2:10 p.m.

Submission procedure: on paper in class.