

Project
SNU 4910.210, Programming Principles Fall 2018
Chung-Kil Hur
due: 12/20(Thu.) 23:59

Problem 1 (50 Points) In Scala, implement an interpreter, `myeval`, for the programming language E given below.

`myeval` : $E \rightarrow V$

A	<code>::=</code>	<code>x</code>	call by value
		<code>(by-name x)</code>	call by name
B	<code>::=</code>	<code>(def f (A*) E)</code>	def
		<code>(val x E)</code>	val
		<code>(lazy-val x E)</code>	lazy val

$E ::=$	x	name
	n	integer
	true	true
	false	false
	(if E E E)	conditional
	nil	list nil
	(cons E E)	pair constructor
	(fst E)	the first component of a pair
	(snd E)	the second component of a pair
	(match-list E E $(hd\ tl)$ E)	pattern matching for nil and cons values
	(let (B^*) E)	name binding of def/val
	(app E E^*)	function call
	(rmk B^*)	Record constructor
	(rfd E x)	Record field access
	(+ E E)	integer addition
	(- E E)	integer subtraction
	(* E E)	integer multiplication
	(= E E)	integer equality
	(< E E)	integer less than
	(> E E)	integer greater than

- Define the value type V , and implement the type class `ConvertToScala` for V .
- For ill-typed inputs, you can return arbitrary values, or raise exceptions.
- X^* denotes that X can appear 0 or more times.
- **let** clauses create a new scope like a ‘block’ in Scala. Name bindings **def**, **val**, and **lazy-val** work the same way as in Scala.
 - **(def** f (A^*) E) assigns name f to expression E with arguments A^* . Examples include **(def** **f** **(a** **(by-name** **b))** **(+ a b)**) and **(def** **g** **()** **3)**.
 - **(val** x E) (respectively, **(lazy-val** x E)) assigns name x to the value obtained by evaluating E (respectively, lazily).
 - You don not have to consider forward reference in **val**. For example, **(val** **x** **(cons 1 x))**.
 - Hint: Implement environment with mutable data structure for **lazyness**.
- **(match-list** E_1 E_2 $(hd\ tl)$ E_3) first evaluates E_1 into value v_1 . If v_1 is **nil**, it evaluates E_2 to get the final value. If v_1 is **(cons** v_1 v_2), it evaluates E_3 with binding $hd := v_1$ and $tl := v_2$ to get the final value.
- **rmk** and **rfd** implement record types.
 - **(rmk** B^*) constructs a record value.
 - **(rfd** E x) projects out the field x of the record value obtained by evaluating E .

You can find examples in `Test.scala`.

Problem 2 (15 Points) Make `myeval` to perform tail recursion optimization. Examples should be found in `Test.scala`. (Hint: You can use Scala's tail recursion optimization.)

Problem 3 (15 Points) Add exception handling to `myeval` by implementing `throw` and `try-catch` following. (Hint: You can use Scala's exception handling)

$$\begin{array}{lcl} E & ::= & \dots \\ & | & (\text{throw } E) \quad \text{throw exception} \\ & | & (\text{try-catch } E (x) E) \quad \text{try-catch} \end{array}$$

- Exception handling consists of `throw` and `try-catch`.
 - `throw E` raises an exception with evaluated value of E to its handler.
 - `try-catch E_1 (x) E_2` tries to evaluate E_1 at first. If the evaluation of E_1 never raises an exception, `try-catch` expression results in the evaluated value of E_1 . Otherwise, the expression results in the evaluated value of E_2 with binding $x := v$, where v is the thrown value.

Problem 4 (20 Points) Implement an parser, `myparser`, for the programming language E mentioned above.

$$\text{myparser} : \text{List}[\text{Token}] \rightarrow E$$

Also, refer to `Lexer` provided by TA. (See `lib/Lexer.scala`)

$$\text{ProjLexer} : \text{String} \rightarrow \text{List}[\text{Token}]$$