CS320 Programming Languages
Exercise #5: MRFVAE

The goal of Exercise #5 is to implement the interpreter of MRFVAE, which is the extended version of FVAE with Multi-arugments and Records. The concrete syntax of MRFVAE is written in the extended Backus-Naur form (EBNF). Note that { } denotes a repetition of zero or more times.

1 Functions with Multiple Arguments

Implement MRFVAE to support any number of arguments to a function (including zero), and any number of arguments (including zero) in a function application:

```
expr  ::=  num
         |  "(" expr "+" expr ")"
         |  "(" expr "-" expr ")"
         |  "{" "val" id "=" expr ";" expr "}"
         |  id
         |  "{" "(" ")" "=>" expr "}"
         |  "{" id "=>" expr "}"
         |  "{" "(" id {"," id} ")" "=>" expr "}"
         |  expr "(" ")"
         |  expr "(" expr {"," expr} ")"
```

For parsing, any alphanumeric string that starts with a non-numeric character and differs from `val` can be an identifier. At run-time, a new error is now possible: function application with the wrong number of arguments. Your interpreter should detect the mismatch and report an error that includes the words `"wrong arity"`:

```
test(run("{ (x, y) => (x + y) }(1, 2)"), "3")
test(run("{ () => (3 + 4) }()"), "7")
testExc(run("{ (x, y) => (x + y) }(1)"), "wrong arity")
```

2 Adding Records

Extend your interpreter to support the construction of records with named fields, and to support field selection from a record:

```
expr  ::=  ...
         |  "{" "}"
         |  "{" id "=" expr {"," id "=" expr} "}"
         |  expr "." id
```

Adding records means that the language now has three kinds of values: numbers, functions, and records. At run-time, an error may occur because a record is misused as a number, a number is supplied to access, or a record supplied to access does not have the named field. Your error message for the last case should include the words `"no such field"`. For any other cases, you can make up your own error messages (or just let primitive error checking handle problems, such as trying to add a record to a number).

```
test(run("{ x = 1, y = 2 }.x"), "1")
testExc(run("{ x = 1, y = 2 }.z"), "no such field")
testExc(run("{ x = { y = 1 }.z }"), "no such field")
```

Since records are now values, the result of the interpreter can be a record, not only a number or a function. For exercise purposes, we don't want to nail down the representation of a record result, because there are many choices. The examples below therefore use `run`, which is a wrapper on your interpreter that just returns a number string if it produces a number, and it returns the string `"function"` if it produces a function value, but it returns the string `"record"` if it produces a record value:

```
test(run("42"), "42")
test(run("{ x => x }"), "function")
test(run("{ x = 1 }"), "record")
```

3 Abstract Syntax

For abstract syntax, we use $\cdots$ to denote repetitions.

$$
\begin{array}{rcl}
e & ::= & n \\
 & | & e + e \\
 & | & e - e \\
 & | & \mathsf{val}\ x{=}e;\ e \\
 & | & x \\
 & | & \lambda x \cdots x.e \\
 & | & e(e,\cdots,e) \\
 & | & \{x = e,\cdots,x = e\} \\
 & | & e.x
\end{array}
\qquad
\begin{array}{rcl}
v & ::= & n \\
 & | & \langle \lambda x \cdots x.e,\sigma \rangle \\
 & | & \{x = v,\cdots,x = v\}
\end{array}
\qquad
\begin{array}{rcl}
n & \in & \mathbb{Z} \\
x & \in & \mathit{Var} \\
v & \in & \mathit{Val} \\
\sigma & \in & \mathit{Var} \to \mathit{Val}
\end{array}
$$

4 Operational Semantics

$\boxed{\sigma \vdash e \Rightarrow v}$

$$\sigma \vdash n \Rightarrow n$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 + e_2 \Rightarrow n_1 + n_2}$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 - e_2 \Rightarrow n_1 - n_2}$$

$$\frac{\sigma \vdash e_1 \Rightarrow v_1 \quad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash \mathsf{val}\ x{=}e_1;\ e_2 \Rightarrow v_2}$$

$$\frac{x \in \mathit{Domain}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

$$\sigma \vdash \lambda x_1 \cdots x_n.e \Rightarrow \langle \lambda x_1 \cdots x_n.e,\sigma \rangle$$

$$\frac{\sigma \vdash e \Rightarrow \langle \lambda x_1 \cdots x_n.e',\sigma' \rangle \quad \sigma \vdash e_1 \Rightarrow v_1 \quad \cdots \quad \sigma \vdash e_n \Rightarrow v_n \quad \sigma'[x_1 \mapsto v_1,\cdots,x_n \mapsto v_n] \vdash e' \Rightarrow v'}{\sigma \vdash e(e_1,\cdots,e_n) \Rightarrow v'}$$

$$\frac{\sigma \vdash e_1 \Rightarrow v_1 \quad \cdots \quad \sigma \vdash e_n \Rightarrow v_n}{\sigma \vdash \{x_1 = e_1,\cdots,x_n = e_n\} \Rightarrow \{x_1 = v_1,\cdots,x_n = v_n\}}$$

$$\frac{\sigma \vdash e \Rightarrow \{\cdots,x = v,\cdots\}}{\sigma \vdash e.x \Rightarrow v}$$