CS 321: Programming Languages — Homework 3

1 Functions that Accept Multiple Arguments

Start with the `F1WAE` interpreter (WITHOUT deferred substitution), and extend the implementation to support any number of arguments (including zero) to a function, and any number of arguments (including zero) in a function application:

```
<FunDef> = {deffun {<id> <id>*} <FnWAE>}
<FnWAE> = <num>
        | {+ <FnWAE> <FnWAE>}
        | {- <FnWAE> <FnWAE>}
        | {with {<id> <FnWAE>} <FnWAE>}
        | <id>
        | {<id> <FnWAE>*}
```

Since you must change the `F1WAE` datatype, and since different people may change it in different ways, you must provide a `parse` function this time, which accepts a quoted expression and produces an `FnWAE` value. For parsing, assume that any symbol other than `'+`, `'-`, or `'with` can be a function name for a function call. Also, you must provide a `parse-defn` function that takes one (quoted) `deffun` form and produces a `FunDef` value.

Some examples:

```
(test (interp (parse '{f 1 2})
              (list (parse-defn '{deffun {f x y} {+ x y}})))
      3)
(test (interp (parse '{+ {f} {f}})
              (list (parse-defn '{deffun {f} 5})))
      10)
```

Hint: remember that the PLAI language provides the following useful function:

- `map` - takes a function and a list, and applies the function to each element in the list, returning a list of results. For example, if `sexps` is a list of S-expressions to parse, `(map parse sexps)` produces a list of `FnWAE`s by parsing each S-expression.

2 Errors

At run-time, a new error is now possible: function application with the wrong number of arguments. Your `interp` function should detect the mismatch and report an error that includes the words "wrong arity".

As with the interpreter from class, the free identifier error is still also possible and must be detected. Your `interp` function should detect free variables and report an error that includes the words "free identifier".

A function would be ill-defined if two of its argument `<id>`s were the same. To prevent this problem, your `parse-defn` function should detect this problem and reports a "bad syntax" error. For example, `(parse-defn '{deffun {f x x} x})` should report a "bad syntax" error, while `(parse-defn '{deffun {f x y} x})` should produce a `FunDef` value.

Similarly, your interpreter must check to see if the function position of an application is defined before evaluating the arguments. If not, an "undefined function" error should be raised.

Some examples:

```
(test/exn (interp (parse '{with {x y} 1})
                  (list))
          "free identifier")
(test/exn (interp (parse '{f 1 2})
                  (list (parse-defn '{deffun {f x x} {+ x x}})))
          "bad syntax")
(test/exn (interp (parse '{f x})
                  (list (parse-defn '{deffun {g a b c} c})))
          "undefined function")
(test/exn (interp (parse '{f 1})
                  (list (parse-defn '{deffun {f x y} {+ x y}})))
          "wrong arity")
```

Your interpreter must evaluate all of the argument expressions in an application expressions before signaling any arity errors. For example:

```
(test/exn (interp (parse '{f x})
                  (list (parse-defn '{deffun {f a b c} c})))
          "free identifier")
```

So overall, the order in which errors should be raised for a given function call is:

- First, "undefined function"
- Second, any errors that are raised while evaluating arguments
- Third, "wrong arity"
- Last, any errors that are raised while evaluating the function body

If the list of definitions contains multiple definitions with the same name, use just the first one (ignoring the others). In particular, there is no overloading in this language: if there are two definitions with the same name and different arities, the first definition is always used regardless.

For example, if you have the following list of definitions:

```
(list (parse-defn '{deffun {f a}   5})
      (parse-defn '{deffun {f a b} {+ a b}}))
```

and the program:

```
{f 3 4}
```

since the first definition of `f` takes a single argument, this program should raise a "wrong arity" error. The fact that another definition for `f` that takes two arguments exists doesn't matter.

Your interpreter and parser will not be given any other kinds of erroneous programs besides the ones described in this section.

3 Handin Instructions

Provide definitions for `parse`, `parse-defn`, and `interp`, as above.

Have the 8 rules from the Provost's website (see homework 1 for more details).

Submit your code via Canvas.

Your submission must include your test cases; submissions without test cases will get a grade of 0.