CS 321: Programming Languages — Homework 7

1 Exceptions

Build an interpreter for the following `EFAE` language:

```
<EFAE> = <num>
       | {+ <EFAE> <EFAE>}
       | {- <EFAE> <EFAE>}
       | <id>
       | {fun {<id>} <EFAE>}
       | {<EFAE> <EFAE>} ;; function application
       | {if0 <EFAE> <EFAE> <EFAE>}
       | {throw <id> <EFAE>}
       | {try <EFAE1> {catch {tag <id1> as <id2>} <EFAE2>}}
```

`EFAE` features exceptions (which are present in many languages, and which you've probably used). That is, execution can escape from a `try` block into a `catch` block when an exception is `throw`n.

Furthermore, exceptions have *tags* which identify which kind of exception is being thrown or caught. Think "file not found" exception vs "network unreachable" exceptions vs etc. An exception can only be caught by a `catch` block with a tag that matches the tag of the `throw` that threw the exception.

Please use the following datatype definition for this language:

```
(define-type EFAE
  [num (n number?)]
  [add (lhs EFAE?)
       (rhs EFAE?)]
  [sub (lhs EFAE?)
       (rhs EFAE?)]
  [id (name symbol?)]
  [fun (param symbol?)
       (body EFAE?)]
  [app (fun-expr EFAE?)
       (arg-expr EFAE?)]
  [if0 (tst EFAE?)
       (thn EFAE?)
       (els EFAE?)]
  [throw (tag symbol?)
         (throw-expr EFAE?)]
  [try-catch (try-body EFAE?)
             (tag symbol?)
             (exn-name symbol?)
             (catch-body EFAE?)])
```

The new constructs should behave as follows:

- `throw`: Throw an exception with tag `<id>` and value `<EFAE>`. That is, instead of proceeding to evaluate the immediate surrounding context of the `throw` expression, jump to the nearest enclosing `catch` block with the same tag.

  Throwing an exception without a corresponding `catch` block (i.e., with the same tag) should produce an error whose message includes the string `"missing catch"`.

- `try ... catch`: Evaluate the `try` body `<EFAE1>`. If an exception with tag `<id1>` is raised, the catch body `<EFAE2>` should be executed, and its result becomes the result of the entire `try ... catch` expression. Within the `catch` block, the value of the exception (the value of the `<EFAE>` part of `throw`) is bound to `<id2>`.

  If an exception with any other tag is raised, look outwards for a matching `catch` block. If no exception is raised, the result of evaluating the `try ... catch` block is the result of evaluating the `try` body.

  Note that only exceptions thrown during the execution of the `try` block may lead the executing the `catch` block. I.e., if an exception is thrown while executing the `catch` block, it may not be caught by that same `catch` block.

Other constructs behave as they did in earlier homeworks. In particular, closures count as non-`0` when used in the test position of `if0`.

Some examples:

```
(test (interp-expr (parse `{+ 2 {try {+ 4 {throw x 5}}
                                     {catch {tag x as e} {+ 3 e}}}}))
      10)
(test (interp-expr (parse `{try {+ 2 {try {+ 3 {throw y 5}}
                                          {catch {tag x as e} {+ 6 e}}}}
                                 {catch {tag y as e} {+ 10 e}}}))
      15)
(test/exn (interp-expr (parse `{try {throw a 1} {catch {tag a as b} {throw a 1}}}))
          "missing catch")
; you can translate `with` as usual
(test (interp-expr (parse `{with {f {fun {x} {throw a {+ x 1}}}}
                                 {try {throw a {+ {f 3} 10}}
                                      {catch {tag a as j} {+ j 5}}}}))
      9)
```

Your implementation must not use control at the PLAI level. (I.e., no PLAI exceptions.)

2 Hint

A good first step for this assignment is to ignore tags, but get everything else working. So `throw`ing would always go to the nearest enclosing `catch` block, regardless of tag.

Once you get that part working, then you can work on distinguishing between different tags.

3 Conveniences

To make your life easier when testing your interpreter, your parser may recognize an extended version of the `EFAE` language and/or you can write a compiler from an extended language (e.g., including `with` and/or multi-argument functions) to `EFAE`.

We will not be testing those extensions; our tests will only cover the `EFAE` language above, including your parser. Extensions would be strictly for your convenience.

4 Errors

There are four different kinds of errors that can occur (at run-time) in this language and for each error in the input program, your interpreter must signal an error that includes one of the following phrases:

- `"free identifier"`
- `"expected function"`
- `"expected number"`
- `"missing catch"`

Note that each operation in the language that evaluates its sub-expressions immediately must evaluate them from left to right and must evaluate all of them before checking any error conditions.

5 Handin instructions

Provide a definition of `interp-expr : EFAE -> number or 'function`, as above.
Provide a definition of `parse : s-expression -> EFAE`, as above.
You must use the datatype definition for `EFAE` provided 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.