Assignment 6

Task 1: Call-by-reference arguments (5 Points)

In the lecture, you have seen how to support "call-by-reference" using a new keyword "refun" for functions that have "call-by-reference" semantics. This is however dangerous, as it is not obvious to a caller of a function whether the arguments may be modified or not.

We want to make this more obvious by introducing a Ref expression for function applications:

`App(fun-expr, Ref(variable))`

That is, we do not use a different function type, but instead an indication at application time that call-by-reference should be used. Extend the given call-by-value interpreter to support this kind of call-by-reference.

Note: Expressions of type `Ref(identifier)` only need to be supported as wrapper for arguments and should explicitly return a descriptive error in any other context (a MatchError is not sufficient here).

Note: We already added a case (without implementation) in the interpreter for application with Ref variables to reduce the design space of possible solutions.

Mark your modifications as clearly as possible.

Task 2: Weak references (7 Points)

Weak references retain a pointer to a box, but do not prevent the garbage collection of the box. Accordingly, after some time, an only weakly referenced box might be garbage collected. After this has happened, dereferencing the weak reference returns an undefined box (or null).

Implement weak references for our language with mark-and-sweep garbage collection.

For this, the two new expressions `WeakRef` and `TryDeref` are added to our language. The representation of the weak references is done with the help of a new value, `WeakRefV`.

Each garbage collection must trigger every `WeakRefV` in the store to be invalid, i.e. rewrite the reference in the `WeakRefV` to the given value `INVALID_LOC`.

Also have a look at the tests for further specification details.

Mark your modifications as clearly as possible.