Aidan Leuck

Interpreter Extension

For my interpreter extension I am going to add a simple implementation of a pointer. You will be able to assign a memory address to the pointer, assign the pointer to an address of a variable in the environment, as well as dereference the pointer. Obviously this is a useless thing to do when using Python, but I think it will be an interesting addition.

CFG:

```
assn : id '=' expr              // Current assignment in CFG
| pointer id '=' expr           // Sets a pointers address
| pointer id '=' &id            // Sets a pointers address to the address of an id
| *id '=' expr                  // Dereferences pointer and sets new value

fact: …
| *id                           // Dereferences a pointer

wr…
| wr pointer x                  // Prints pointers address in hex
```

Scanner:

Scanner functionality that needs to be added to recognize the keyword pointer. This will let the parser differentiate between existing fact nodes and it will allow the semantic evaluator know that the assignment is going to be the memory address rather than just a normal variable assignment. It will also have to recognize the symbol &. & will be used to get the address of a variable.

Parser:

The parser will have to be updated substantially with new branching logic and parse functions in order to handle the new grammar. This should simply involve if statements that checks the current tokens lexeme and call appropriate functions. Three new assignment nodes will need to be added, NodeAssnPntr, NodeAssnDereference, and NodeAssnSetPntrVal. NodeAssnPntr will handle pointer id= expr and pointer id = &id. It will hold two values, an expr and an id. The expression value will be used if the pointer is being assigned the value of an expression and the id value will be used if you are setting the pointer to the address of the variable. NodeAssnSetPntrVal will be used for the statement *id = expr. It will hold two values, idPointer and an expr.

One new fact node will need to be added as well, NodeFactDeref. NodeFactDeref will hold simply hold the id of the variable that needs to be dereferenced. The only difference between the existing fact node will be the evaluation function. This node will need to be added in order to handle evaluation differences between the existing NodeFact node.

We will also have to update the wr node so that when evaluating a pointer, it will print out the result in hexadecimal. This will be done by adding a flag passed into the wr node that will tell it whether it is a pointer or not. 0 = not pointer 1 = pointer.

Evaluation:

Most of the work to implement the new grammar is going to involve adding new evaluation for existing and new nodes. All of the nodes will implement this sort of logic in one way or another. The node will either retrieve a value at a memory address, retrieve the memory address, or update the value at a memory address. A Nodes evaluation will use one of these three options in order to return the correct result. Pointers will be stored along with variables in the environment and be treated like a normal variable unless dereferenced. At that point the address will be retrieved from simulated memory, since python does not allow direct access to the memory.

Implementation:

In order to implement this there are a few changes that need to be made to the program as a whole. First of all I will be creating a new class called memory simulator which is simply a hashmap that contains the value {address: id}. Address is the address of the corresponding id. All eval functions will now include a second parameter, memory which is the hashmap containing all memory values. A second exception will also need to be created to catch when there is not an address associated with a variable in memory. This exception will print an appropriate error message about the address not being accessible.

An example of how the program functions will be explained with this example

b = 45; pointer y = &b; *y = 20; wr b

b = 45, ENV {b: 45}.

pointer y = &b.

ENV{b:45, y: (address of b)
memory {(address of b): b}

*y = 20

address = env.get(y) – Retrieve address of b from environment
id = memory.get(address) – Retrieve corresponding id for that address.
Env.put(id, 20) - Set new value for b.

Using this design implementation a crude implementation of pointers should be able to be created. It is mostly useless, especially since there will be no array implementation in the parser. If an address is assigned to pointer and that address does not exist we may run into exceptions when trying to dereference. An error message should be displayed appropriately. One thing to note is that this is not a true implementation of pointers, because python does not allow memory access. For this reason setting a value of a pointer manually and dereferencing will not work unless there is a variable in the environment with the specified address.

Test Cases:

b = 45; pointer y = &b; *y = 20; wr b (Expected result will be 20)
pointer x = 30000; wr x (Expected result will be 30000 in hex) // This will simply be used in order to verify that we can manually set an address to 30000.
pointer x = 30000; wr *x (Throw an exception, address is not set to a id in the environment)
a = 20; pointer y = &a; wr *y (Expected result is 20)
b= 20; y = 5; pointer x = &y; wr *x * b (Expected result is 100)
b =5; pointer x = &b; wr pointer x (Expected result is the address of b in hex)
x = 40; pointer y = &x; *y = *y + 1; def abc (y) = 2* *y; wr abc(y) (Excepted result is 82)