RefLang

October 20, 2016

Side Effect

- Pure functional programs can be understood in terms of their input and output. Given the same input a functional program would produce the same output.
- Change the state of the program besides its output
- Examples:
  - **Reading or writing memory locations**: an important feature, design tradeoffs
  - Printing on console, reading user input,
  - File read and file write,
  - Throwing exceptions,
  - Sending packets on network,
  - Acquiring mutual exclusion locks, etc...

Two Concepts

- **Heap**: an abstraction representing area in the memory reserved for dynamic memory allocation
- **References**: locations in the heap

Design Decisions – Heap

Heap size is finite, programming languages adopt strategies to remove unused portions of memory so that new memory can be allocated.

- manual memory management: the language provides a feature (e.g. free in C/C++) to deallocate memory and the programmer is responsible for inserting memory deallocation at appropriate locations in their programs.
- automatic memory management: the language does not provide explicit feature for deallocation. Rather, the language implementation is responsible for reclaiming ununsed memory (Java, C#).

How individual memory locations in the heap are treated:

- untyped heap: the type of value stored at a memory location is not fixed, can be changed during program execution
- typed heap: each memory location has an associated type and it can only contain values of that type, the type of value stored at a memory location doesn't change during the program's execution

Design Decisions – Reference (pointers)

1. Explicit references: references are program objects available to the programmer
2. Implicit references: references only available to implementation of the language
3. Reference arithmetic: references are integers and thus we can apply arithmetic operations
4. Deref and assignment only: get the value stored at that location in the heap, assignment can change the value stored at that location in the heap

Examples:

- C Programming language: manual memory management, explicit reference, untyped heap, reference arithmetic
- Java: automatic memory management, deref and assignment only, untyped heap, implicit reference
- Reflang: manual memory management, deref and assignment, untyped heap, explicit references

RefLang

- Expressions for allocating a memory location, dereferences a location reference, assign a new value to an existing memory location, free previously allocated memory location
- Examples:
  $(ref 1)
  loc: 0
- Value: the location at which memory was allocated (next available memory location)
- Side effect: assign value 1 to the allocated memory location
- Value and type are known from the expression

Reflang Expressions

ref: This expression evaluates its subexpression to a value, allocates a new memory location to hold this value, and returns a reference value that encapsulates information about the newly allocated memory location.

$ (define loc1 (ref 12))
// stores value 12 at some location in memory, creates a reference value to encapsulate (and remember) that location, and stores that reference value in variable loc1

$ (define loc2 (ref 45))

$ loc1 // check the reference value stored in variable loc1
loc:0

$ loc2
loc:1

Reflang Expressions

deref: This expression evaluates its subexpression to a value. If that evaluation evaluates to a reference value, and that reference value encapsulates a location l, then it retrieves the value stored in Heap at location l.

```
$ (deref loc1) // gives the value stored at loc1
12

$ (deref loc2) // gives the value stored at loc2
45

$ (+ (deref loc1) (deref loc2)) //access both values and adds them
57
```

Reflang Expressions

assign: This expression is used to change the value stored on some location in Heap.

$ (set! loc1 23) //previous value 12 is overwritten by 23
23

$ (set! loc2 24) //previous value 45 is overwritten by 24
24

$ loc1 // loc1 still has address 0 but value has changed now
loc:0

$ loc2 // loc2 still has address 0 but value has changed now
loc:1

$ (+ (deref loc1) (deref loc2)) // different value different summation value
47

Reflang Expressions

free: This expression is used to deallocate the reference stored in Heap.

$ (free loc1) // deallocates the memory address 0

$ loc1 // variable loc1 still points to same location loc:0

$ (deref loc1) // dereference loc1
Error:null // invalid because memory location has been freed

$ (free loc2) // deallocates the memory address stored in loc2

$ (deref loc2) // dereference loc2
Error:null // invalid because memory location has been freed

RefLang: More Examples

```
$ (free (ref 1)) // delocate the memory location where 1 is stored
$ (deref (ref 1)) // deref a memory location defined by ref 1
$ (let ((loc (ref 1))) (deref loc))
$ (let ((loc (ref 1))) (set! loc 2))
```

- ref, free, deref, set!

Reflang: Grammar

```
Program     ::= DefineDecl* Exp?                       Program
DefineDecl  ::= (define Identifier Exp)                Define
Exp         ::=                                        Expressions
                Number                                 NumExp
              | (+ Exp Exp+)                           AddExp
              | (- Exp Exp+)                           SubExp
              | (* Exp Exp+)                           MultExp
              | (/ Exp Exp+)                           DivExp
              | Identifier                             VarExp
              | (let ((Identifier Exp)+) Exp)          LetExp
              | ( Exp Exp+)                            CallExp
              | (lambda (Identifier+) Exp)             LambdaExp
              | (ref Exp)                              RefExp
              | (deref Exp)                            DerefExp
              | (set!  Exp Exp)                        AssignExp
              | (free Exp)                             FreeExp
```

Reflang: Extending Values

- RefVal $\neq$ NumVal
  - prevent from accessing arbitrary memory location
  - no arithmetics
  - extra meta data

RefLang: Heap abstraction

$$\text{Heap} : \text{RefVal} \rightarrow \text{Value}$$

```java
public interface Heap {
  Value ref (Value value) ;
  Value deref (RefVal loc) ;
  Value setref (RefVal loc, Value value) ;
  Value free (RefVal value) ;
}
```

Reflang Expression Semantics

- Expression do not affect heap directly or indirectly:
  Constant expression: value e env h = (NumVal n) h
  n is a Number, env is an environment, h is a heap
  Variable expression – look up names for values: value (VarExp var) env h = get(env, var) h
- Indirectly affect heap through their subexpressions
- Directly affect heap

Reflang Expression Semantics: Indirectly affect heap

- the order in which side efects from one subexpression are visible to the next subexpression has significant implications on the semantics of the defined programming language.
- Add expression:

$$\texttt{value (AddExp } e_0 \ \ldots \ e_n\texttt{) env h} = v_0 + \ldots + v_n,\ h_n$$

$$\texttt{if value } e_0 \texttt{ env h} = v_0\ h_0,\ \ldots,\ \texttt{value } e_n \texttt{ env } h_{n-1} = v_n\ h_n$$

$$\texttt{where } e_0,\ \ldots,\ e_n \in \texttt{Exp},\ \texttt{env} \in \texttt{Env},\ \texttt{h},\ h_0, \ldots h_n \in \texttt{Heap}$$

a left-to-right order is used in the relation above for side-effect visibility

Reflang Expression Semantics: Directly affect heap

- ref, set!, free
- deref: read from memory only

Reflang: RefExp

$$\texttt{value (RefExp e) env h = l, h}_2$$

$$\texttt{if value e env h = v}_0\ \texttt{h}_1$$

$$\texttt{h}_2 = \texttt{h}_1 \cup \{\ \texttt{l} \mapsto \texttt{v}_0\ \} \qquad \texttt{l} \notin \texttt{dom(h}_1\texttt{)}$$

where $\texttt{e} \in \texttt{Exp} \quad \texttt{env} \in \texttt{Env} \quad \texttt{h},\texttt{h}_1,\texttt{h}_2 \in \texttt{Heap} \quad \texttt{l} \in \texttt{RefVal}$

Reflang: AssignExp

$$\texttt{value (AssignExp } e_0\ e_1\texttt{) env h = } v_0\texttt{, } h_3$$

$$\texttt{if value } e_1 \texttt{ env h = } v_0\ h_1 \qquad \texttt{value } e_0 \texttt{ env } h_1 \texttt{ = l } h_2$$

$$h_3 = \{\ \texttt{l} \mapsto v_0\ \} \cup (h_2 \setminus \{\ \texttt{l} \mapsto \_\ \}) \qquad \texttt{l} \in \texttt{dom}(h_2)$$

$$\texttt{where e} \in \texttt{Exp} \quad \texttt{env} \in \texttt{Env} \quad \texttt{h}, h_1, h_2, h_3 \in \texttt{Heap} \quad \texttt{l} \in \texttt{RefVal}$$

Reflang: FreeExp

$$\texttt{value (FreeExp e) env h = unit, } \texttt{h}_2$$

$$\texttt{if value e env h = l } \texttt{h}_1 \qquad \texttt{l} \in \texttt{dom}(\texttt{h}_1)$$

$$\texttt{h}_2 = \texttt{h}_1 \setminus \{\ \texttt{l} \mapsto \_\ \}$$

where $\texttt{e} \in \texttt{Exp}$ $\quad \texttt{env} \in \texttt{Env}$ $\quad \texttt{h}, \texttt{h}_1, \texttt{h}_2 \in \texttt{Heap}$ $\quad \texttt{l} \in \texttt{RefVal}$ $\quad \texttt{unit} \in \texttt{Unit}$

Reflang: DerefExp

$$\texttt{value (DerefExp e) env h = v, } \mathtt{h_1}$$

$$\texttt{if value e env h = l } \mathtt{h_1} \qquad \mathtt{l} \in \mathtt{dom}(\mathtt{h_1})$$

$$\{\ \mathtt{l} \mapsto \mathtt{v}\ \} \subseteq \mathtt{h_1}$$

$$\texttt{where e} \in \texttt{Exp} \quad \texttt{env} \in \texttt{Env} \quad \mathtt{h,h_1} \in \texttt{Heap} \quad \mathtt{l} \in \texttt{RefVal} \quad \mathtt{v} \in \texttt{Value}$$

Realizing Heap and Evaluators

See RefLang interpreter Code