POP PROJECT

JAVA EXTENSION TO SUPPORT *"FORCE AND DELAY"* CONSTRUCT

Abstract :-

- Java being a Object-oriented programming language doesn't have inbuilt support for functional programming. One such feature which we want to extend java with is "Delayed Execution".

- This gives us the ease of object-oriented programming and efficiency of functional programming.

- Though this can be done using existing features of java but it is not highly readable. And also, language directly supporting the feature would save the additional overhead of function calls and class instantiation.

- The construct is
  - Expression → …….
    - Delay (Expression) |
    - Force (Expression) |
    - …….

- This delay construct is simply denoting Lazy-evaluation of Expression.
- It will not be computed until its first use, whereafter the computed value can be used.

- The possible impact includes but not only limited to increasing efficiency of the program in terms of cpu time. It also gives the programmer the control over execution of expression.

Motivation :-

The standard case where lazy evaluation terminates whereas eager evaluation do not terminate.

```
class Rou
{
   public int fir()
   {
      for (int i = 0 ; i < 10 ; i++)
      {
```

```
        i--;
      }

      return 1;
   }
};
class Main
{
   public static void main(String[] args)
   {
      Rou h = new  Rou();
      int x = h.fir();
   }
};
```

The above program will not terminate.

We could use delay to mitigate this

int x = delay(h.fir());

Reducing time complexity :

```
class Rou
{
   public int fir(int a,int b,in c)
   {
      if (c == 1 )
      {
         //some code using only value a
      }
      else
      {
         //some code using only value b
      }
      return 1;
   }

   public int time_taking_one()
   {
      //Some function that takes a lot of cycles to return value.
   }

   public int time_taking_two()
   {
      //Some function that takes a lot of cycles to return value.
   }
};
```

```
class Main

{
   public static void main(String[] args)
   {
      Rou h = new  Rou();
      int y = rand() % 2;
      int x = h.fir(delay(h.time_taking_one()),delay(h.time_taking_two()),y);
   }
};
```

Here if we hadn't used delay, we would be unnecessarily calculating one of the values , thus wasting cpu time.

Drawbacks:

- This construct can be time-efficient only when the argument evaluation is computation intensive, else the additional overhead for delayed expression will cause an increase in time -taken.

Semantics :-

- **Rules for JavaTypes:**

  Newtype : …… //all the java primitive types
  | (delay , Newtype)

  (delay,T) <= T //(delay,T) is a sub class of Type T

- **New Type-checking rules** :-

A-| expression: t
A-| Delay (expression) : (delay,t)

A-| expression: (delay,t)
A-| Force (expression): t

- **Operational semantics :-**

A o B -| expression : v
A-| Force (clos(expression, B)): v

v :- value | (delay, v)

A-| expression
A-| delay(expression) : clos(expression, A)

Prior Work :-

- Delay and force are extensively used in the functional programming language Scheme.
- It is used to simulate lazy evaluation and to generate infinite lists.

Implementation Plan :-

- Expression pointer:- Points to the closure of the expression.

- Closure of the expression contains the 'expression' to be evaluated and the 'environment' in which the expression is to be evaluated.

- Value:-

An constant value 0 representing normal value
Pointer to the value
Empty
Empty

- Delayed Expression:-

A constant value 1 representing delayed expression
Pointer to the value is stored here after its computed once (Support for Memoization)
A dummy value (0 if value not computed yet else 1)
Pointer to the closure of the Expression to be evaluated

- The above will be the structure for the types as stored in memory. Now while generating the bytecode for java , we will allocate the structures accordingly and initialize them.

- Whenever delay is applied on an expression, we will construct the delayed expression block , storing the closure of the expression and return it.

- Whenever force is applied on a delayed expression block, firstly check if dummy value is 0,if so then evaluate the expression pointed by the expression pointer in the environment specified in the closure and store the pointer to the value in the block and change the dummy value to 1.

- We might even want to free the Pointer to Expression here since its not needed anymore.
- Then, return the Pointer to the value .

- When normal values are referenced, we need to return the pointer to the value.