AngelaZ

Angel invests on those who awaits and prepare for the Zen of Matrix
People know Matrix.
We know the ZEN!
• Yeah, sit there doing nothing!
• But, not really…..
• Our language focus on Matrix Computation with operators and build your Customized Matrix Computation with great EASE.
Int i;
Boolean b;
Matrix main2(Int argc, String argv) {
Matrix m3(2,2);
Matrix m(2,2);
    m[0][0]=1;
    m[0][1]=2;
    m[1][0]=3;
    m[1][1]=4;
    m3 = (((m + m') * m^) * .. 4)+.. m^;
return m3;
}

Void main(Int argc2, String m)
{
Matrix result(2,2);
result=main2(0, "str");
printM(result);
}

Columbia Students are one-liners.
So, Make it happen!
A series of operators

• “+”, “-”: positive/ negative sign

• “*”, “/”, “+.”, “-.”, “+..”, “-..”: primary type level, matrix level, and matrix & primary level
Demo 2

- `Int i;`
- `Boolean b;`
- `Structure main2(Int argc, String argv) {
  Structure s={a="1", b= toString(argc)};
  i=toInt(s -> a);
  return s;
}

Void main(Int argc2, String m) {
  Structure result={};
  result=main2(0, "str");
  print(result);
}
Structure holds customized data

- **Universal**: Anything that can be expressed as a String. Can be a String or a variable of String type
- **No Overhead**: Anything your care to use without OOP overhead that a financial user does not care to know
- **Extensibility**: Easily extended to other disciplinary without much effort
Demo 3

- Float i;
- Option main2(Int argc, String argv) {
  Option s={strike="100.0", stock="150.0", interestRate="0.1", period="1.0", sigma="2.0", optionType="call"};
  i=toFloat(s -> strike);
  return s;
}

- Void main(Int argc2, String m) {
  Option result={};
  result=main2(0, "str");
  Float d;
  d=price(result);
  print(d);
}
Demo 3 extended

- Matrix main3(Int a) {
  - Matrix strike(1,2);
  - strike[0][0]=10;
  - strike[0][1]=20;
  - Matrix stock(1,2);
  - stock[0][0]=15;
  - stock[0][1]=25;
  - Matrix interestRate(1,2);
  - interestRate[0][0]=0.4;
  - interestRate[0][1]=0.1;
  - Matrix period(1,2);
  - period[0][0]=3;
  - period[0][1]=4;
  - Matrix sigma(1,2);
  - sigma[0][0]=0.1;
  - sigma[0][1]=0.2;
  - Matrix s(0,0);
  - s=priceM(strike,stock,interestRate, period, sigma);
  - return s;
}
In Financial District

- Easy to use: One of the application of extensible language

- Make complex things easy: Don’t know Black-Shole or anything alike.

- Matrix short-cut for large portfolio
Black-Scholes equation

\[ u(x, \tau) = Ke^{x + \frac{1}{2}\sigma^2 \tau} N(d_1) - KN(d_2) \]

where

\[ d_1 = \frac{1}{\sigma \sqrt{\tau}} \left[ \left( x + \frac{1}{2}\sigma^2 \tau \right) + \frac{1}{2}\sigma^2 \tau \right] \]

\[ d_2 = \frac{1}{\sigma \sqrt{\tau}} \left[ \left( x + \frac{1}{2}\sigma^2 \tau \right) - \frac{1}{2}\sigma^2 \tau \right] \]
Scanner/Parser

- In Scanner, translate characters to tokens
  
```plaintext
rule token = parse
  [ ' ' '	' '
' '' 'n'] { token lexbuf } (* Whitespace *)
| "Matrix" { MATRIX }
| "'" { TRANSPOSE }
| '~' { INVERSION }
| '^' { DETERMINANT }
```

- In Parser, pattern matching and pattern reduction to build an AST tree
  
```plaintext
eexpr:
  ID { Id($1) }
/* matrix_unary: */
| expr TRANSPOSE { MatUnary_op($1, MTranspose) }
| expr INVERSION { MatUnary_op($1, MInversion) }
| expr DETERMINANT { MatUnary_op($1, MDeterminant) }
stmt:
  expr SEMI { Expr($1) }
```
• In AST, define structure corresponding to each pattern in Parser

```ocaml
type mat_uop = MTranspose | MInversion | MDeterminant
type expr =
  Id of string
| MatUnary_op of expr * mat_uop

(* "Pretty printed" version of the AST *)
let rec string_of_expr = function
  Id(s) -> s
| MatUnary_op(e, o) ->
  (match o with
   MTranspose -> "Transpose" | MInversion -> "Inversion" | 
   MDeterminant -> "Determinant"
   ) ^ "(" ^ string_of_expr e ^ ")"
```
Sast/Typechecking

- **Annotate Ast:**

```haskell
type expr_t =
    Binary_op_t of expr_t * bin_op * expr_t * dataType
  | MatBinary_op_t of expr_t * mat_op * expr_t * dataType
  | Id_t of string * dataType
```

- **Environment/Scopes:**

```haskell
type matrix_table = {
    matrix_name : string; (*name of a matrix*)
    msize : size_of_matrix; (*size of a matrix*)
}
type symbol_table = {
    (*general symbol table for variables*)
    parent : symbol_table option;
    mutable variables : (string * Ast.dataType) list;
    mutable structs : struc_table list;
    mutable options : option_table list;
    mutable matrixes : matrix_table list;
}
type environment = {
    mutable func_return_type : Ast.dataType; (* Function return type *)
    scope : symbol_table; (* symbol table for variables *)
    mutable functions : (string * Ast.dataType list * Ast.dataType) list;
}```
Sast/Typechecking

• Basic checks about types and consistency
  • Types of operations/expressions are consistent
    • int convert to float is allowed, reverse is not allowed
    • +.. -> left side be of Matrix type, right side be of Float
  • Variables and functions are defined within scope and in the right type
  • Statements
    • if(expr)—expr can only be of boolean type;
    • for(e1;e2;e3)— e1 and e3 can only be noexpr or assignment expr
Sast/Typechecking

- Checks for specific data type
- Structures/Options
  - fields within structure must be declared ahead
  - No duplicate fields declaration
  - Option has built-in function
- Matrices
  - dimension matches for matrices operations
    - +, -, *
      - ./
Code Generation (1)

• Challenge:
  1. No operator overload in java
  2. Exceptions (division by zero)
  3. Access member element of Struct
  4. Java initialization requirements (in global not in arguments)
Code Generation (2)

• Solutions:
  1. Operator $\rightarrow$ method
  2. Try/catch $\rightarrow$ catch need to return the same type as function definition $\rightarrow$ match pattern return type
  3. Member access $\rightarrow$ HashMap
  4. Match for different type and initialize
Tests

- Unit test for each developing phase: AST, Parser and scanner/ SAST/ JavaGen
- Integration test for the linked modules.
- Shell script is used to automatically run the test cases and compare output.
- Pass and fail test cases are designed separately.
Thanks for the semester!

<table>
<thead>
<tr>
<th></th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanner, Parser, AST</td>
<td>Jiayi Yan (major); Fei Liu; Taikun Liu</td>
</tr>
<tr>
<td>SAST, Typecheck</td>
<td>Taikun Liu</td>
</tr>
<tr>
<td>javagen, java codes</td>
<td>Fei Liu</td>
</tr>
<tr>
<td>Test cases, test scripts</td>
<td>Mengdi Zhang; Fei Liu</td>
</tr>
<tr>
<td>LRM, final report</td>
<td>Mengdi Zhang (major); Jiayi Yan (major); Taikun Liu; Fei Liu</td>
</tr>
</tbody>
</table>