XIRTAM

Lior Attias, Shida Jing, Annie Wang, Andrew Gorovoy, Bailey
Nozomu Hwa

Xirtam Overview

- Xirtam is a matrix manipulation language with:
  - The functionality of Python’s **Numpy**
  - Syntax of **C** to maximize readability
  - Excel-like function calling
  - Unique features aimed at **saving users’ time**.
- Users: data scientists and frequent excel users (business analysts) who are:
  - Familiar with working with matrices
  - In need of something more powerful than Excel
  - Novice-to intermediate Programmers
- Uses:
  - For the said individuals to learn how to code
  - Quick and easy operations with matrices

4 Data Types

Num: Catch all for numbers, whether ints, floats, or doubles

Xirtam: 2-Dimensional Matrix of Numeric Values or Expressions

Bool: Basic Boolean type

String: Basic String type

Convenient Features

- Few Data Types
- Convenient Error checking
    - Checks for uninitialized variables
- Autocorrection to **Save Time**:
    - Matrices
        - Built-in Matrix functions autocorrect user input where possible
    - Functions
        - Automatic return values if a return value is not specified
    - For main()
        - LLVM expected int type
        - We added hidden int type to make the entry point an int in the back: as long as the user names the entry point main, it will work

OVERVIEW

```
num global1;
num global2;
num add(num a, num b) {
   num c;
   c = a + b;
   return c;
}
num main(){
    string s; bool b; xirtam m;
    s = "Hello";
    global1 = 1;
    global2 = 2;
    b = true;
    m = [[1, 2],[ 4, 5]];
    add(global1, global2);
    printn(global1); /* 1 */
    printm(m);
    /* 1.00 2.00
       4.00 5.00*/
}
```

Global Variable Declaration

Function Declaration

Variable initialization before assignment

Function Call

Comments

Main function should have no return value

Control Flow

If ... else statement

```
if (true && true) {
       printn(1);
   } else {
       printn(0);
   }
```

For loop

```
num i;
 for (i = 0 ; i < 5 ; i = i + 1) {
   printn(i);
 }
```

While loop

```
num i;
 i = 5;
 while (i > 0) {
   printn(i);
   i = i - 1;
 }
```

Matrix Functionalities

- Easy initialization of a matrix
- User can easily get the following attributes of matrices:
    - Transpose of matrix
    - Rows
    - Columns
    - Number of rows
    - Number of columns
- User can easily perform the following transformations
    - Add two matrices
        - Component-wise addition
    - Subtract two matrices
    - Multiply two matrices
    - Get/Set specific index values

***All matrix methods have error checking

Matrix Initialization

Initialization of Matrix

```
num main() {
    xirtam m;
    m = [[1, 2], [3,4], [5,6]];

}
```

```
num main() {
    xirtam m;
    m = autofill(3,3,1);

}
```

Creates 3 x 3 matrix with 1 in each index

Internal Logic

```
matrix* initMatrix(double* listOfValues, int
num_cols, int num_rows) {
  double* matrixValues = malloc(num_rows *
num_cols * sizeof(double*));

    for(int r = 0; r < num_rows; r++) {
      for(int c = 0; c < num_cols; c++) {
        int idx = c + (r * num_cols);
        matrixValues[idx]=listOfValues[idx];
      }
    }


  //return a pointer to matrix struct

}
```

Matrix Attribute Examples

Number of Rows/Columns

```
num main() {
    xirtam m;
    num result;

    m = [[1,2,1,3],[1,2,2,3],[1,2,2,3]];
    result = getrows(m);

    println(result);
}



### 3
```

Transpose

```
num main() {
    xirtam m;
    m = [[1, 2], [3, 4]];

    printm(trans(m));
}

### 1.00  3.00
    2.00 4.00
```

Matrix Function Examples

Add/Subtract

```
num main() {
    xirtam m;
    m = [[1, 2], [3, 4], [5, 6]];
    printm(matadd(m,m));
}
```

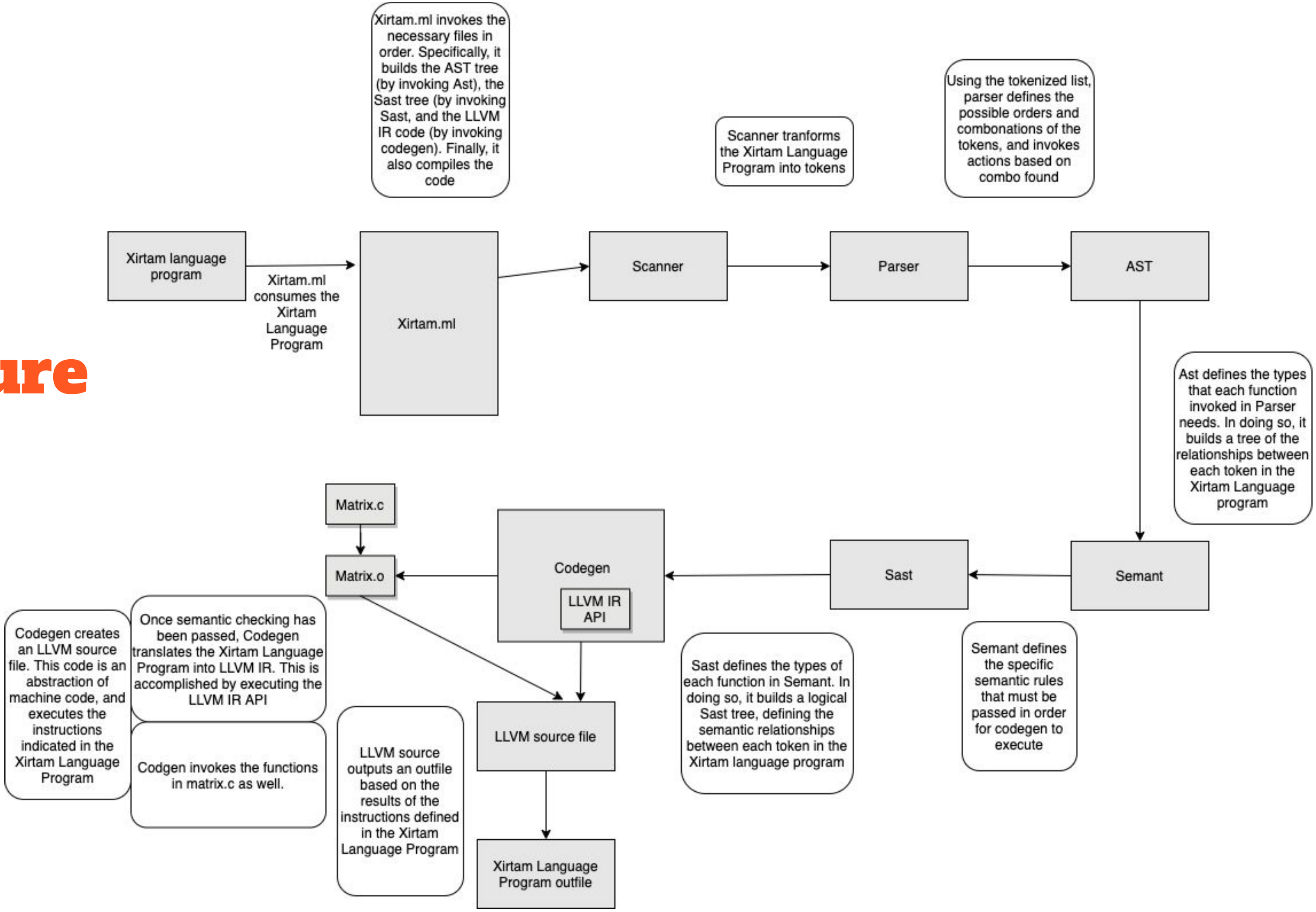
```
###  2.00   4.00
     6.00   8.00
     10.00  12.00
```

Multiplication

```
num main() {
    xirtam m;
    xirtam n;
    xirtam ret;
    num r;
    m = [[4, 2,1], [422, 21], [0.4, 6.2]];
    n = [[1, 2, 3], [0.5, -1.2, 0]];
    ret = matmult(m, n);
}
```

```
###
Fatal error: exception Failure("No staggered
Matrices allowed, rows must be same size")
```

Compiler Architecture with data flow



Xirtam.ml invokes the necessary files in order. Specifically, it builds the AST tree (by invoking Ast), the Sast tree (by invoking Sast, and the LLVM IR code (by invoking codegen). Finally, it also compiles the code

Scanner tranforms the Xirtam Language Program into tokens

Using the tokenized list, parser defines the possible orders and combonations of the tokens, and invokes actions based on combo found

Xirtam.ml consumes the Xirtam Language Program

Ast defines the types that each function invoked in Parser needs. In doing so, it builds a tree of the relationships between each token in the Xirtam Language program

Semant defines the specific semantic rules that must be passed in order for codegen to execute

Sast defines the types of each function in Semant. In doing so, it builds a logical Sast tree, defining the semantic relationships between each token in the Xirtam language program

Once semantic checking has been passed, Codegen translates the Xirtam Language Program into LLVM IR. This is accomplished by executing the LLVM IR API

Codegen creates an LLVM source file. This code is an abstraction of machine code, and executes the instructions indicated in the Xirtam Language Program

Codgen invokes the functions in matrix.c as well.

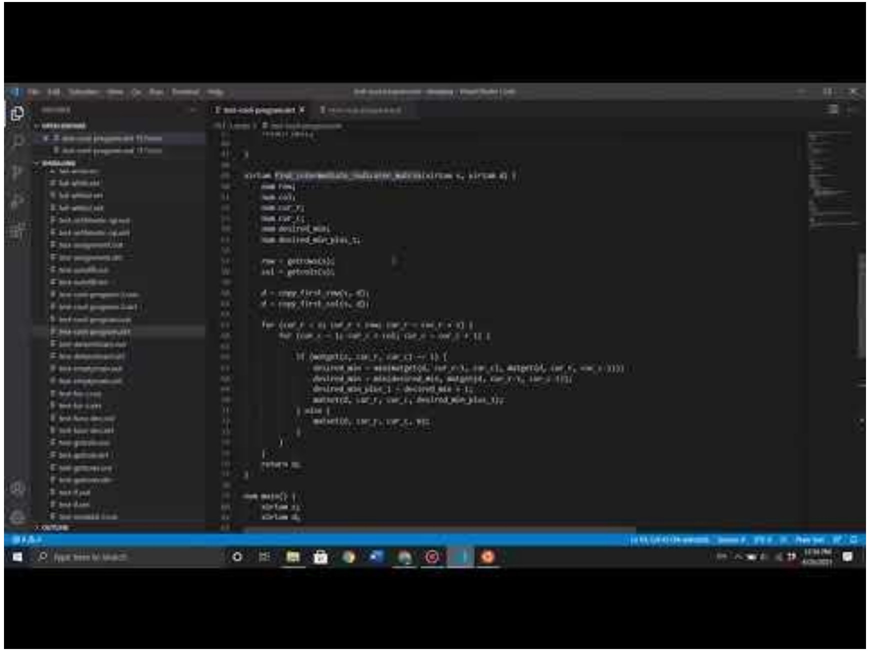
LLVM source outputs an outfile based on the results of the instructions defined in the Xirtam Language Program

Future work

- Add in more complex built in matrix operations
- Increase the flexibility of matrix operations even more! Allow a user to concat matrices of different sizes, etc
- Image processing, tensorflow-- implement the basic functionalities to users to build complex programs

Demo

Problem: given a binary matrix X and a zero matrix Y, build up information about clusterings of 1’s in X, and store the information in Y.



```
        [illegible]
}

virtum find_intermediate_indicator_matrix(virtum x, virtum d) {
    num row;
    num col;
    num cur_r;
    num cur_c;
    num desired_min;
    num desired_min_plus_1;

    row = getrows(x);
    col = getcols(x);

    d = copy_first_row(x, d);
    d = copy_first_col(x, d);

    for (cur_r = 1; cur_r < row; cur_r = cur_r + 1) {
        for (cur_c = 1; cur_c < col; cur_c = cur_c + 1) {

            if (matget(x, cur_r, cur_c) == 1) {
                desired_min = min(matget(d, cur_r-1, cur_c), matget(d, cur_r, cur_c-1));
                desired_min = min(desired_min, matget(d, cur_r-1, cur_c-1));
                desired_min_plus_1 = desired_min + 1;
                matset(d, cur_r, cur_c, desired_min_plus_1);
            } else {
                matset(d, cur_r, cur_c, 0);
            }
        }
    }
    return d;
}

num main() {
    virtum x;
    virtum d;
```

Thank you for your help

Thank you, Professor Edwards and all the TA’s for all the help you provided.

Citation: this language is built upon MicroC and past project Matrx.