McLab Tutorial
www.sable.mcgill.ca/mclab

Part 6 – Introduction to the McLab Backends

- MATLAB-to-MATLAB
- MATLAB-to-Fortran90 (McFor)
- McVM with JIT

MATLAB-to-MATLAB

- We wish to support high-level transformations, as well as refactoring tools.
- Keep comments in the AST.
- Can produce .xml or .m files from McAST or McLAST.
- Design of McLAST such that it remains valid MATLAB, although simplified.

MATLAB-to-Fortran90

- MATLAB programmers often want to develop their prototype in MATLAB and then develop a FORTRAN implementation based on the prototype.
- 1st version of McFOR implemented by Jun Li as M.Sc. thesis.
  - handled a smallish subset of MATLAB
  - gave excellent performance for the benchmarks handled
  - provided good insights into the problems needed to be solved, and some good initial solutions.
- 2nd version of McFOR currently under development.
  - fairly large subset of MATLAB, more complete solutions
  - provide a set of analyses, transformations and IR simplifications that will likely be suitable for both the FORTRAN generator, as well as other HLL.
- e-mail hendren@cs.mcgill.ca to be put on the list of those interested in McFor.

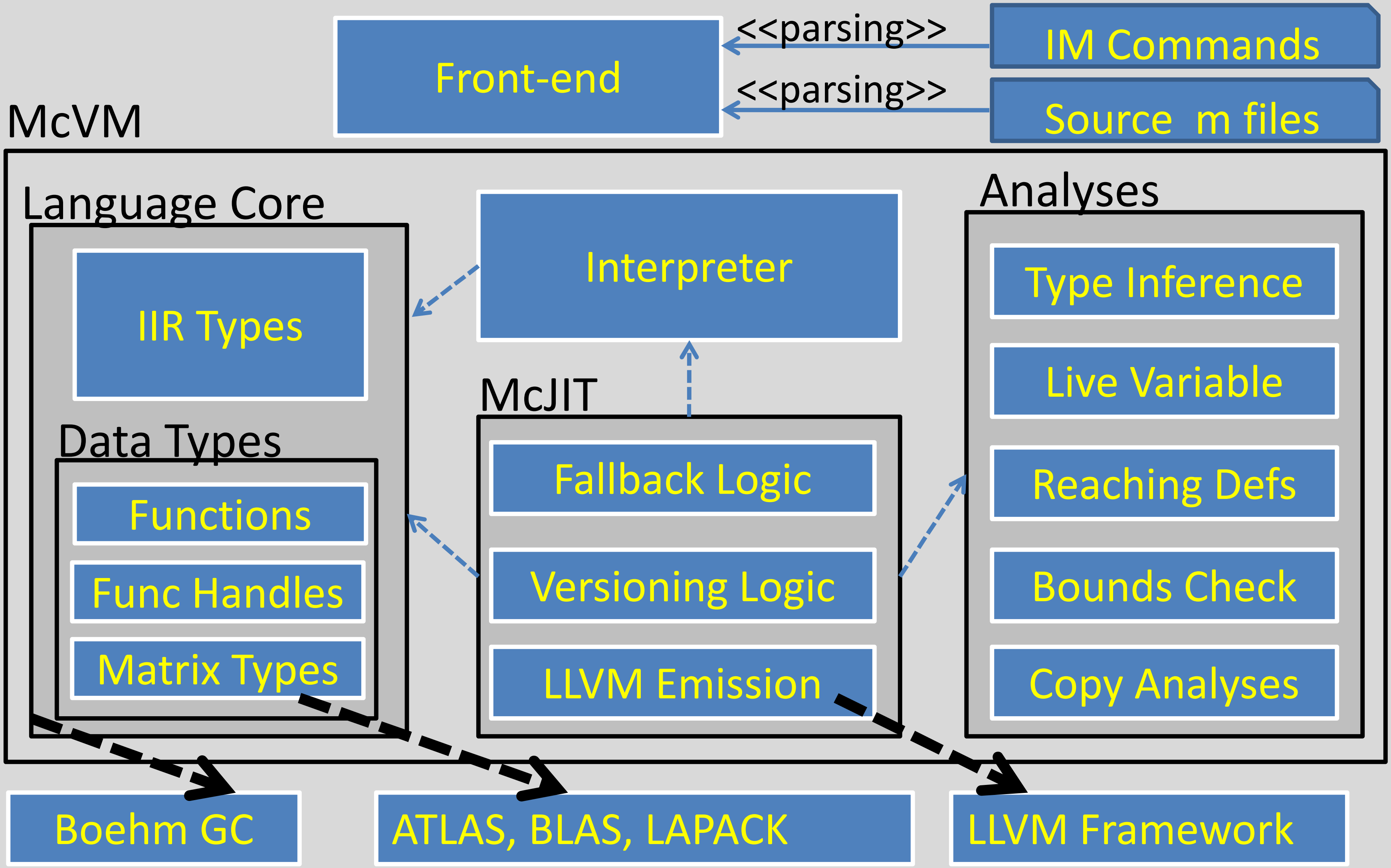
McVM-McJIT

- Whereas the other back-ends are based on static analyses and ahead-of-time compilation, the dynamic nature of MATLAB makes it more suitable for a VM/JIT.
- MathWorks' implementation does have a JIT, although technical details are not known.
- McVM/McJIT is an open implementation aimed at supporting research into dynamic optimization techniques for MATLAB.

McVM Design

- A basic but fast interpreter for the MATLAB language
- A garbage-collected JIT Compiler as an extension to the interpreter
- Easy to add new data types and statements by modifying only the interpreter.
- Supported by the LLVM compiler framework and some numerical computing libraries.
- Written entirely in C++; interface with the McLab front-end via a network port.

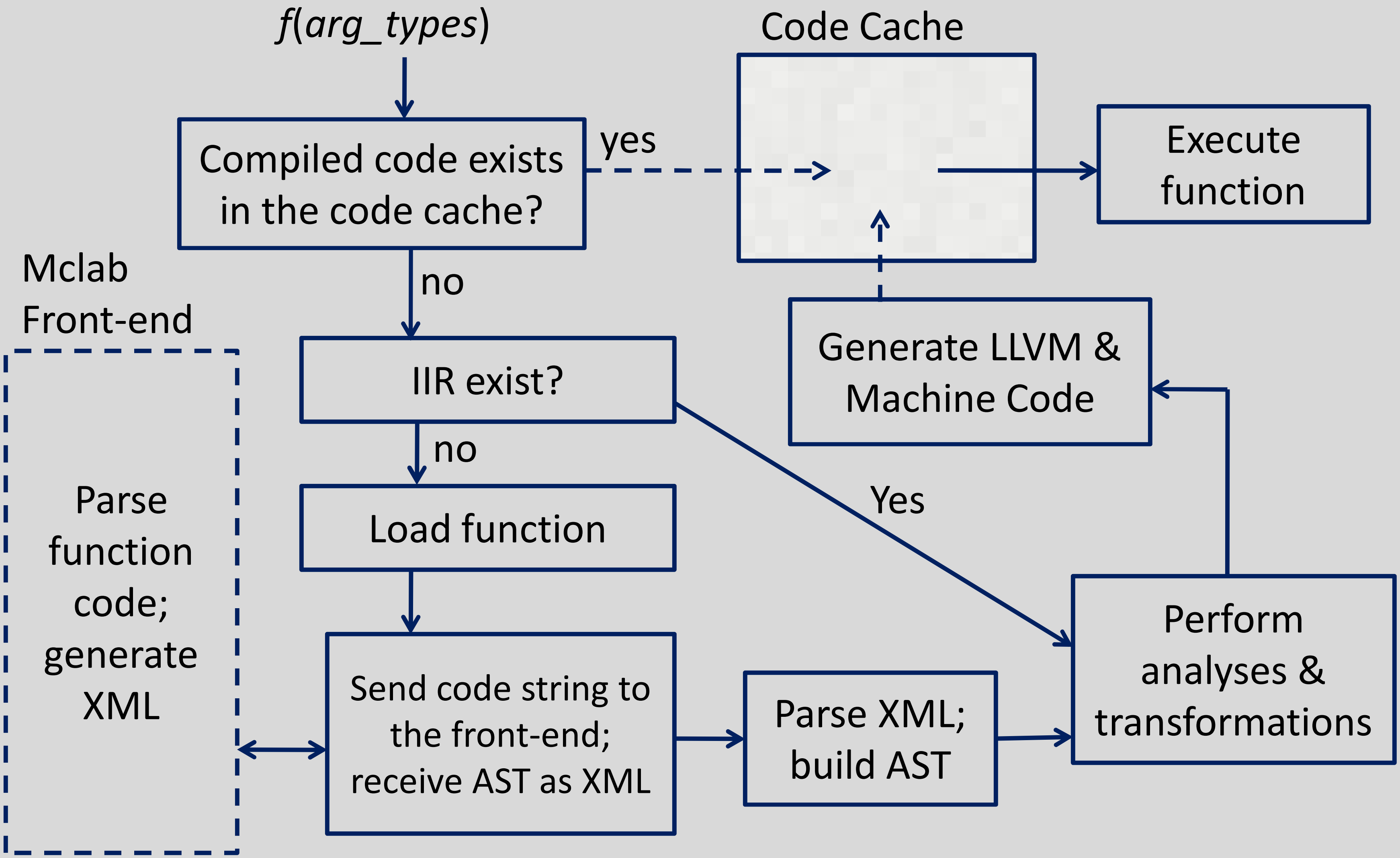
The Structure of McVM



Supported Types

- Logical Arrays
- Character Arrays
- Double-precision floating points
- Double-precision complex number matrices
- Cell arrays
- Function Handles

McJIT: Executing a Function


f(*arg_types*)
Code Cache
Compiled code exists in the code cache?
yes
Execute function
Mclab Front-end
no
IIR exist?
Generate LLVM & Machine Code
no
Load function
Yes
Parse function code; generate XML
Perform analyses & transformations
Send code string to the front-end; receive AST as XML
Parse XML; build AST

Type Inference

- It is a key performance driver for the JIT Compiler:
  - the type information provided are used by the JIT compiler for function specialization.

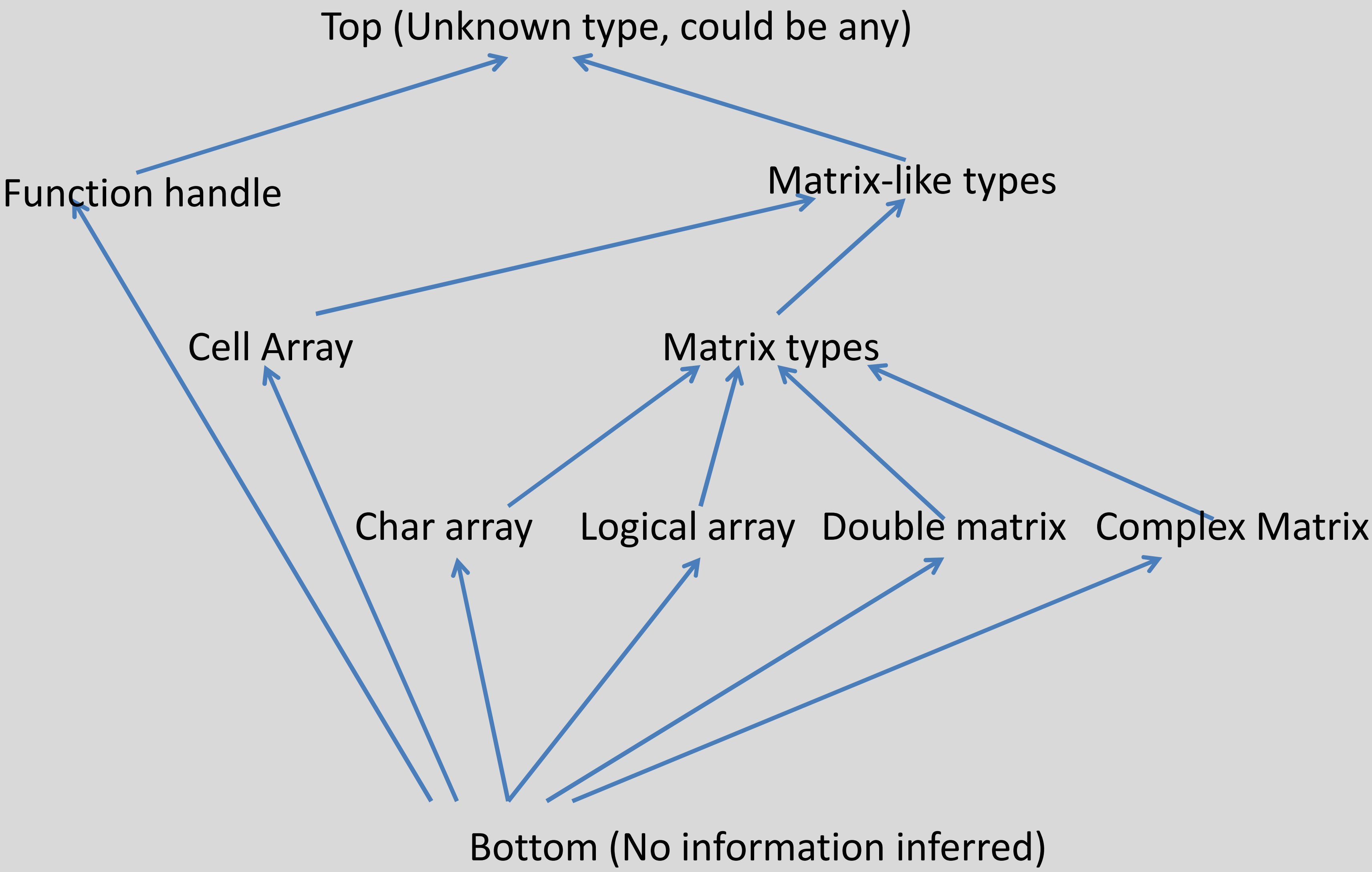
Type Inference

- It is a forward flow analysis: propagates the set of possible types through every possible branch of a function.
- Assumes that:

  for each input argument *arg*, there exist some possible types

- At every program point *p*, infers the set of possible types for each variable
- May generate different results for the same function at different times depending on the types of the input arguments

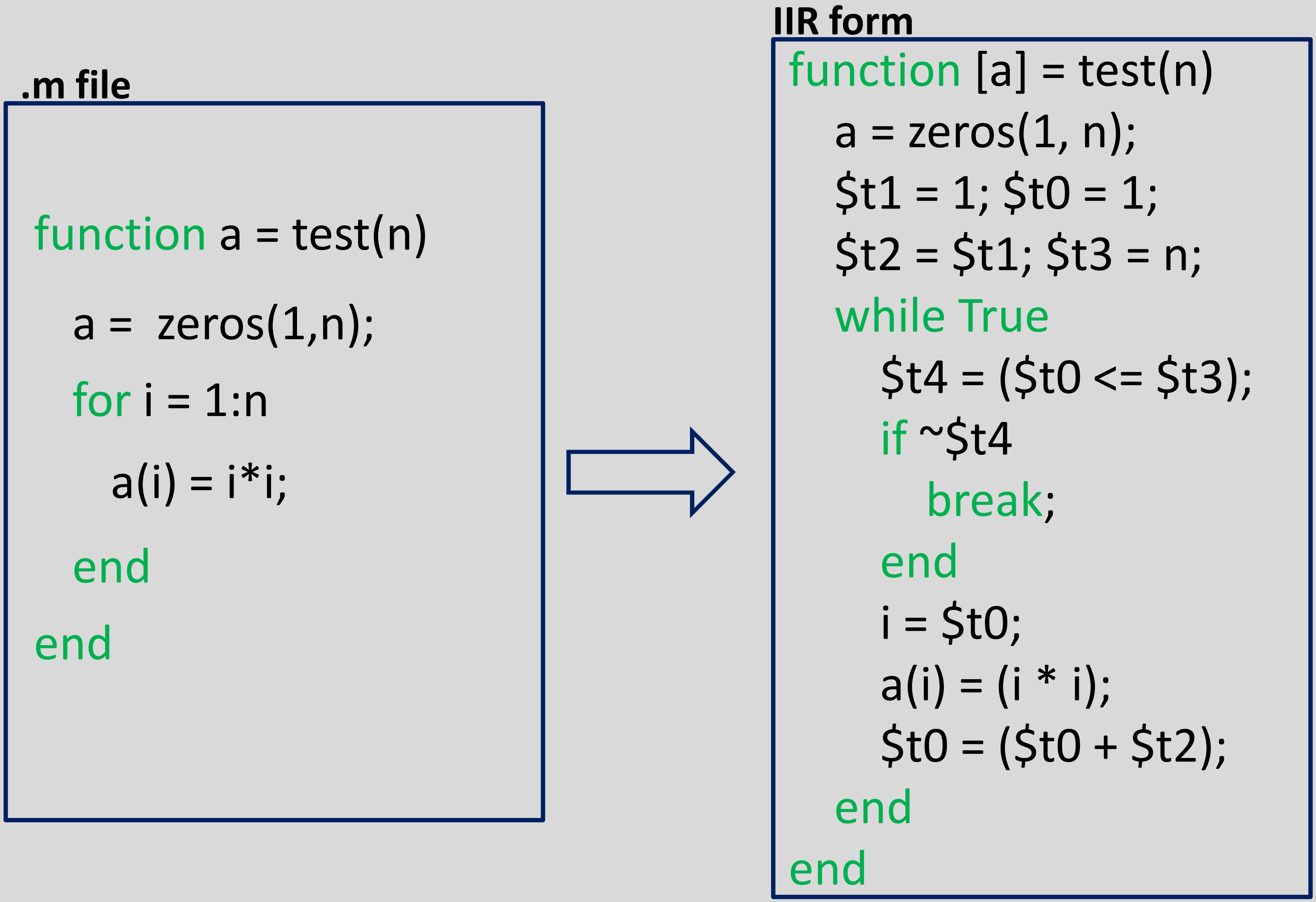
Lattice of McVM types



Internal Intermediate Representation

- A simplified form of the Abstract Syntax Tree (AST) of the original source program
- It is machine independent
- All IIR nodes are garbage collected

IIR: A Simple MATLAB Program



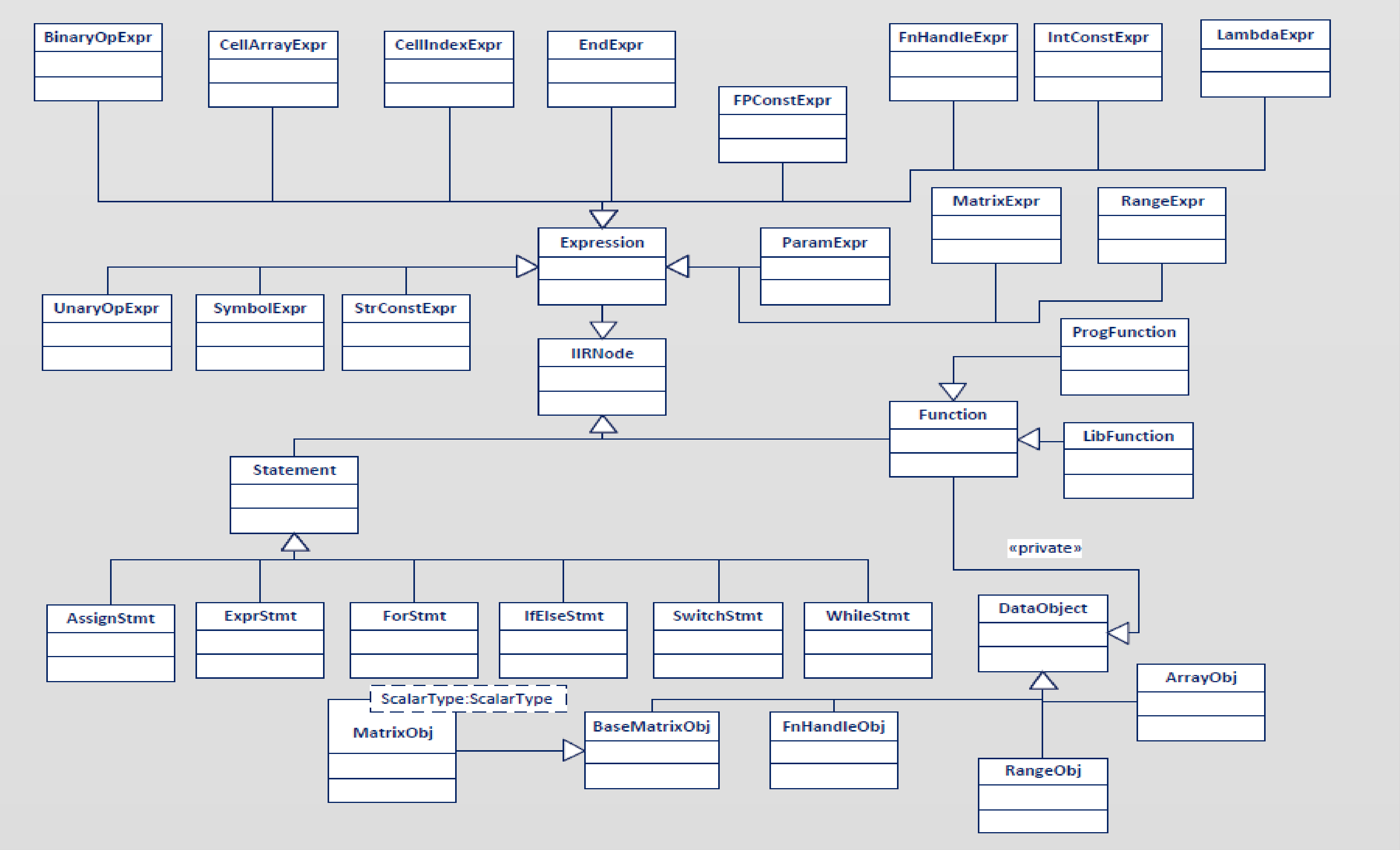
.m file

```
function a = test(n)
  a =  zeros(1,n);
  for i = 1:n
    a(i) = i*i;
  end
end
```

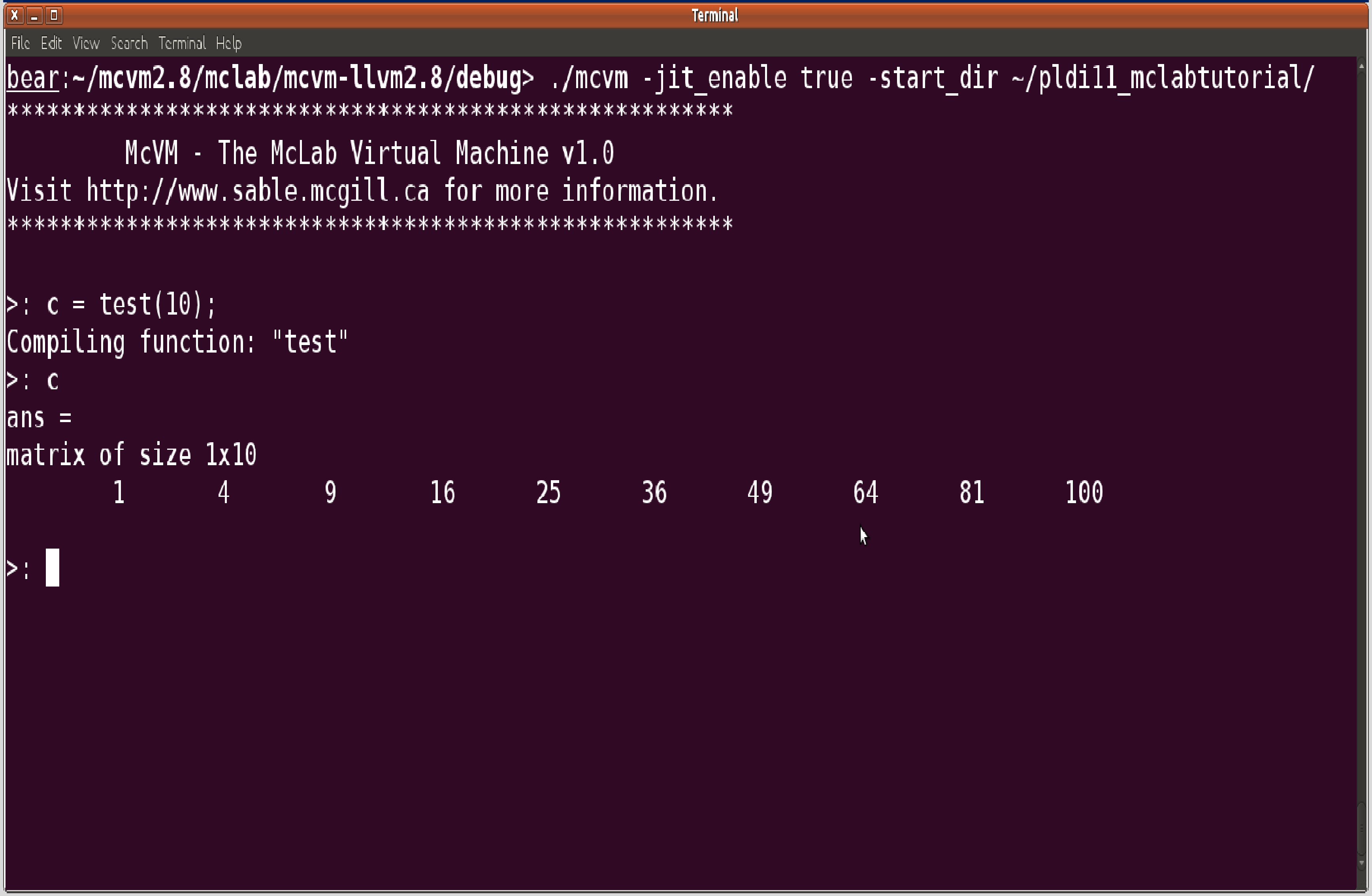
IIR form

```
function [a] = test(n)
   a = zeros(1, n);
   $t1 = 1; $t0 = 1;
   $t2 = $t1; $t3 = n;
   while True
      $t4 = ($t0 <= $t3);
      if ~$t4
         break;
      end
      i = $t0;
      a(i) = (i * i);
      $t0 = ($t0 + $t2);
   end
end
```

McVM Project Class Hierarchy (C++ Classes)



Running McVM



```
bear:~/mcvm2.8/mclab/mcvm-llvm2.8/debug> ./mcvm -jit_enable true -start_dir ~/pldi11_mclabtutorial/
*******************************************************
         McVM - The McLab Virtual Machine v1.0
Visit http://www.sable.mcgill.ca for more information.
*******************************************************


>: c = test(10);
Compiling function: "test"
>: c
ans =
matrix of size 1x10
        1       4       9      16      25      36      49      64      81     100

>: 
```