Overview

- Why MATLAB?
- Introduction to MATLAB – challenges
- Overview of the McLab tools
- Resolving names in MATLAB


- 38% of scientists spend at least 1/5th of their time programming.
- Codes often buggy, sometimes leading to papers being retracted. Self-taught programmers.
- Monster codes, poorly documented, poorly tested, and often used inappropriately.
- 45% say scientists spend more time programming than 5 years ago.

A lot of MATLAB programmers!

- Started as an interface to standard FORTRAN libraries for use by students... but now
  - 1 million MATLAB programmers in 2004, number doubling every 1.5 to 2 years.
  - over 1200 MATLAB/Simulink books
  - used in many sciences and engineering disciplines
- Even more “unofficial” MATLAB programmers including those using free systems such as Octave or SciLab.
Why do Scientists choose MATLAB?

MATLAB

FORTRAN

Implications of choosing a dynamic, “scripting” language like MATLAB....

No types and “flexible” syntax

http://imgs.xkcd.com/comics/fourier.jpg

Many run-time decisions ...

Potentially large runtime overhead in both time and space

Most semantic (syntactic) checks made at runtime ... No static guarantees

No formal standards for MATLAB
Scientists / Engineers
- Comfortable with informal descriptions and “how to” documentation.
- Don’t really care about types and scoping mechanisms, at least when developing small prototypes.
- Appreciate libraries, convenient syntax, simple tool support, and interactive development tools.

Programming Language / Compiler Researchers
- Prefer more formal language specifications.
- Prefer well-defined types (even if dynamic) and well-defined scoping and modularization mechanisms.
- Appreciate “harder/deeper/more beautiful” programming language/compiler research problems.

Goals of the McLab Project
- Improve the understanding and documentation of the semantics of MATLAB.
- Provide front-end compiler tools suitable for MATLAB and language extensions of MATLAB.
- Provide a flow-analysis framework and a suite of analyses suitable for a wide range of compiler/soft. eng. applications.
- Provide back-ends that enable experimentation with JIT and ahead-of-time compilation.

Enable PL, Compiler and SE Researchers to work on MATLAB

Culture Gap

Brief Introduction to MATLAB

Functions and Scripts in MATLAB

Basic Structure of a MATLAB function:

```matlab
function [ prod, sum ] = ProdSum( a, n )
prod = 1;
sum = 0;
for i = 1:n
    prod = prod * a(i);
    sum = sum + a(i);
end;
end
```

Basic Structure of a MATLAB script:

```matlab
% stored in file ProdSumScript.m
prod = 1;
sum = 0;
for i = 1:n
    prod = prod * a(i);
end;
```

Primary, nested and sub-functions:

```matlab
function [ prod, sum ] = ProdSum( a, n )
    function [ z ] = MyTimes( x, y )
        z = x * y;
    end
    prod = 1;
    sum = 0;
    for i = 1:n
        prod = MyTimes( prod, a(i) );
        sum = MySum( sum, a(i) );
    end;
end

function [ z ] = MySum( x, y )
    z = x + y;
end
```

**Example usage:**

```matlab
[a,b] = ProdSum([10,20,30],3)  
ans = 6000
b = 60
```

```
>> ProdSum([10,20,30],2)  
ans = 200
>> ProdSum('abc',3)  
ans = 941094
>> ProdSum([97 98 99],3)  
ans = 941084
```
Directory Structure and Path

- Each directory can contain:
  - `.m` files (which can contain a script or functions)
  - a `private/` directory
  - a package directory of the form `+pkg/`
  - a type-specialized directory of the form `@int32/`

- At run-time:
  - current directory (implicit 1st element of path)
  - directory of last called function
  - path of directories
  - both the current directory and path can be changed at runtime (`cd` and `setpath` functions)

Function/Script Lookup Order

(call in the body of a function f)

- Nested function (in scope of f)
- Sub-function (in same file as f)
- Function in `/private` sub-directory of directory containing f.
- 1st matching function, based on function name and type of first argument, looking in type-specialized directories, looking first in current directory and then along path.
- 1st matching function/script, based on function name only, looking first in current directory and then along path.

Function/Script Lookup Order

(call in the body of a script s)

- Function in `/private` sub-directory of directory of last called function (not the `/private` sub-directory of the directory containing s).
- 1st matching function/script, based on function name, looking first in current directory and then along path.

Variables and Data in MATLAB

- `f.m` in `dir1/`
- `s.m` in `dir2/`
- `g.m` in `private/`
- `h.m` in `private/`
- `foo.m`
- `% in s.m`

MATLAB types: high-level

- `any`
- `data` (fnhandle
- `array` cellarray struct

Variables

- Variables are not explicitly declared.
- Local variables are allocated in the current workspace. Global and persistent variables in a special workspace.
- All input and output parameters are local.
- Local variables are allocated upon their first definition or via a load statement.
  - `x = ...`
  - `x(i) = ...`
  - `load ('f.mat', 'x')`
- Local variables can hold data with different types at different places in a function/script.
Variable Workspaces

- There is a workspace for global and persistent variables.
- There is a workspace associated with the read-eval-print loop.
- Each function call creates a new workspace (stack frame).
- A script uses the workspace of its caller (either a function workspace or the read-eval-print workspace).

Variable Lookup

- If the variable has been declared global or persistent in the function body, look it up in the global/persistent workspace.
- Otherwise, look up in the current workspace (either the read-eval-print workspace or the top-most function call workspace).
- For nested functions, use the standard scoping mechanisms.

Other Tricky "features" in MATLAB

- Keyword `end` not always required at the end of a function (often missing in files with only one function).
- Command syntax:
  - `length('x')` or `length x`
  - `cd('mydirname')` or `cd mydirname`
- Arrays can be defined with or without commas: `[10, 20, 30]` or `[10 20 30]`
- Sometimes newlines have meaning:
  - `a = [ 10 20 30 40 50 60 ];` // defines a 2x3 matrix
  - `a = [ 10 20 30; 40 50 60 ];` // defines a 2x3 matrix

“Evil” Dynamic Features

- Not all input arguments required
- `function [ prod, sum ] = ProdSumNargs( a, n )`
- `if nargin == 1 n = 1; end;`
- Do not need to use all output arguments
- `eval, evalin, assignin`
- `cd, addpath`
- `load`

Irritating Front-end "Features"

- Keyword `end` not always required at the end of a function (often missing in files with only one function).
- Command syntax:
  - `length('x')` or `length x`
  - `cd('mydirname')` or `cd mydirname`
- Arrays can be defined with or without commas: `[10, 20, 30]` or `[10 20 30]`
- Sometimes newlines have meaning:
  - `a = [ 10 20 30 40 50 60 ];` // defines a 2x3 matrix
  - `a = [ 10 20 30; 40 50 60 ];` // defines a 2x3 matrix
- `a = [ 10 20 30; 40 50 60 ];` // defines a 2x3 matrix

Evil Feature of the Day - Looking up an identifier

- Old style general lookup - interpreter
  - First lookup as a variable.
  - If a variable not found, then look up as a function.
- MATLAB 7 lookup - JIT
  - When function/script first loaded, assign a "kind" to each identifier. VAR – only lookup as a variable, FN – only lookup as a function, ID – use the old style general lookup.
  - How is the kind assignment done. What impact does it have on the semantics?
McLab – Overall Structure

Analysis Engine

Back-ends, McVM and McFor

How does MATLAB resolve Names?

• No official specification
• Motivating example
Evil Feature of the Day - Recap

Old style general lookup - interpreter
- First lookup as a variable.
- If a variable not found, then look up as a function.

MATLAB 7 lookup - JIT
- When function/script first loaded, statically assign a "kind" to each identifier. VAR – only lookup as a variable, FN – only lookup as a function, ID – use the old style general lookup.
- Compile-time error if, within the body of a function or script, an identifier has kind VAR in one place and FN in another.

Does the kind analysis change the semantics?
- Yes, in two ways!

1. New compile-time errors, so programs that would previously execute will not.
2. Different binding at run-time for some identifiers which are assigned a kind of VAR or FN.

Compile-time kind error

Different lookup with old vs MATLAB 7 semantics

1. function [ r ] = KindEx( a )
2. x = a + sum();
3. eval('sum = ones(10);');
4. r = sum(x);
5. end

- Old interpreter semantics:
  - sum, line 2, named function
  - sum, line 4, local variable
- MATLAB 7 semantics gives a static kind of FN to sum
  - sum, line 2, named function
  - sum, line 4, named function
Our approach to the Kind Analysis Problem

- Identify that a kind analysis is needed to match MATLAB 7 semantics.
- Specify and implement a kind assignment algorithm that matches the observed behaviour of MATLAB 7. (both for functions and for scripts)
- Identify any weaknesses in the MATLAB 7 approach and suggest two more clearly defined alternatives, one flow-sensitive and one flow-insensitive.
- Determine if the alternatives could be used without significant change to the behaviour of existing MATLAB programs.

Kind Abstraction

Kind Analysis

1. Collect all identifiers used in function/script and set initial kind approximations for each identifier.
2. Traverse AST applying analysis rules to identifiers.
3. Traverse AST making final kind assignment.

Steps 1 and 3 are different for scripts and functions, step 2 uses the same rules.

Kind Analysis for Functions

- **Initial values**: input and output parameters are initialized to VAR, all other identifiers are initialized as UNDEF.
- **Final values**: for each id occurrence in f do
  - if kind(id) in {ID, MaxVAR}
    - id.kind = ID
  - else /* if kind(id) in {VAR, FN}s */ /*
    - id.kind = kind(id)

Step 2: Kind Analysis Rules

**Definition of identifier x**

\[ \text{kind}[x] \leftarrow \text{kind}[x] \bowtie \text{VAR} \]

**Use of identifier x**

\[
\begin{align*}
\text{if } ((\text{kind}[x] \in \{\text{ID, UNDEF}\}) & \& \text{exists.lib}(x, \text{lib})) \\
\text{kind}[x] & \leftarrow \text{FN} \\
\text{else} \\
\text{kind}[x] & \leftarrow \text{kind}[x] \bowtie \text{ID}
\end{align*}
\]

Kind Analysis for Functions

\[
\begin{align*}
\text{if } ((\text{kind}[x] \in \{\text{ID, UNDEF}\}) & \& \text{exists.lib}(x, \text{lib})) \\
\text{kind}[x] & \leftarrow \text{FN} \\
\text{else} \\
\text{kind}[x] & \leftarrow \text{kind}[x] \bowtie \text{ID}
\end{align*}
\]

**WRITE RULE**

\[
\begin{align*}
\text{if } ((\text{kind}[x] \in \{\text{ID, UNDEF}\}) & \& \text{exists.lib}(x, \text{lib})) \\
\text{kind}[x] & \leftarrow \text{FN} \\
\text{else} \\
\text{kind}[x] & \leftarrow \text{kind}[x] \bowtie \text{ID}
\end{align*}
\]

**READ RULE**

\[
\begin{align*}
\text{if } ((\text{kind}[x] \in \{\text{ID, UNDEF}\}) & \& \text{exists.lib}(x, \text{lib})) \\
\text{kind}[x] & \leftarrow \text{FN} \\
\text{else} \\
\text{kind}[x] & \leftarrow \text{kind}[x] \bowtie \text{ID}
\end{align*}
\]
Kind Analysis for Scripts

- **Initial values:** all identifiers are initialized to MAYVAR
- **Final values:**
  
  - For each id occurrence in a do
    - If id.kind in \{VAR, MAYVAR\}: id.kind = Id
    - else if id.kind must be Fn, it can’t be In or Unkn
      - id.kind = Fn
  
  - **Note:** most identifiers will be mapped to Id

Problems with MATLAB 7 kind analysis

- apparently not clearly documented, in some ways just a side-effect of a JIT implementation decision
- without a clear specification, confusing for the programmer and compiler/tool developer
- loses almost all information about variables in scripts
- some strange anomalies due to a "traversal-sensitive" analysis

Examples of Anomalies

<table>
<thead>
<tr>
<th>Code</th>
<th>Kind in Def</th>
<th>Kind in Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>if (exp) ... = sum(10);</code></td>
<td>(sum, FN)</td>
<td>(sum, VAR)</td>
</tr>
<tr>
<td><code>else sum(10) = ...;</code></td>
<td></td>
<td><em>error</em></td>
</tr>
<tr>
<td><code>size(size(10)) = ...</code></td>
<td>(size, FN)</td>
<td>(size, VAR)</td>
</tr>
<tr>
<td><code>t = size(10);</code></td>
<td>(size, FN)</td>
<td><em>error</em></td>
</tr>
</tbody>
</table>

Flow-sensitive Analysis

- Apply a flow-sensitive analysis that merges at control-flow points.
- Consider explicit loads to be definitions -
  `load ('f.mat', 'x')`
- Map final kinds for scripts using the same algorithm as for functions.

Flow-insensitive Analysis

1. Assign VAR to identifiers that are defined on lhs, or declared global or persistent.
2. Assign FN to identifiers which have a handle taken or used in command syntax.
3. Assign FN to identifiers that have no assignment yet, and which are found in the library.
   *error* if assigned both FN and VAR
Various-sized benchmarks from a wide variety of application areas

<table>
<thead>
<tr>
<th>Benchmark Category</th>
<th># Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single (1 file)</td>
<td>2051</td>
</tr>
<tr>
<td>Small (2-9 files)</td>
<td>848</td>
</tr>
<tr>
<td>Medium (10-49 files)</td>
<td>113</td>
</tr>
<tr>
<td>Large (50-99 files)</td>
<td>9</td>
</tr>
<tr>
<td>Very Large (≥ 100 files)</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>3024</td>
</tr>
</tbody>
</table>

Results for Functions – number of identifiers with each Kind

<table>
<thead>
<tr>
<th>Kind</th>
<th>MATLAB 7</th>
<th>Flow-Sens.</th>
<th>Flow-InSENS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>va</td>
<td>107388</td>
<td>107401</td>
<td>107406</td>
</tr>
<tr>
<td>fn</td>
<td>75533</td>
<td>75533</td>
<td>75533</td>
</tr>
<tr>
<td>ld</td>
<td>2369</td>
<td>2335</td>
<td>2335</td>
</tr>
<tr>
<td>error</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>warn</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Total</td>
<td>185291</td>
<td>185291</td>
<td>185291</td>
</tr>
</tbody>
</table>

Results for Scripts – number of identifier instances with each Kind

<table>
<thead>
<tr>
<th>Kind</th>
<th>MATLAB 7</th>
<th>Flow-Sens.</th>
<th>Flow-InSENS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>va</td>
<td>183044</td>
<td>0</td>
<td>183044</td>
</tr>
<tr>
<td>fn</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ln</td>
<td>222467</td>
<td>68410</td>
<td>68410</td>
</tr>
<tr>
<td>error</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>warn</td>
<td>0</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Total</td>
<td>222467</td>
<td>222467</td>
<td>222467</td>
</tr>
</tbody>
</table>

Conclusions and Ongoing Work

- McLab is a toolkit to enable PL, compiler and SE research on MATLAB (close the gap).
- Release of three main tools: front-end/analysis framework, McVM (Virtual Machine) and McFor (MATLAB to FORTRAN) (tbd). PLDI 2011 tutorial.
- High-level: Refactoring tools for MATLAB. How to help programmers convert their programs to better structured, and more efficient codes?
- Lower-level: static compilation to Fortran90 and new dynamic techniques in McVM/McJIT.