The MAGICA Type Inference Engine for MATLAB®*

Pramod G. Joisha and Prithviraj Banerjee

Department of Electrical and Computer Engineering, Northwestern University, USA.
{pjoisha, banerjee}@ece.northwestern.edu

1 Introduction

MAGICA (MAthematica system for General-purpose Inferring and Compile-time Analyses) is an extensible inference engine that can determine the types (value range, intrinsic type and array shape) of expressions in a MATLAB program. Written as a Mathematica application, it is designed as an add-on module that any MATLAB compiler infrastructure can use to obtain high-quality type inferences.

1.1 A Type Inference Using MAGICA

Lines *In[1]* and *Out[2]* below demonstrate a simple interaction with MAGICA through a notebook interface.[1] On line *In[1]*, the MAGICA `type` function object is applied on a representation of the MATLAB expression `tanh(3.78i)`. MAGICA's response, shown on *Out[2]*, is the inferred type of `tanh(3.78i)`. In this case, "type" is the expression $\{v, i, s\}$ where v, i and s are the value range, intrinsic type and array shape of `tanh(3.78i)`; *Out[2]* indicates these to be the point $0.742071\,\mathbb{i}$, the `$nonreal` intrinsic type designator, and the two-dimensional array shape with unit extents along both dimensions—that is, the scalar shape.

```
In[1]:= type[tanh[3.78i]]
Out[1]= {0.742071 i, $nonreal, {⟨1 ,  1⟩, 2}}
```

1.2 Feature Support

The above is an example of a type inference on a single MATLAB expression. MAGICA can infer the types of whole MATLAB programs comprising an arbitrary number of user-defined functions, each having an arbitrary number of statements. User-defined functions can return multiple values, can consist of

* This research was supported by DARPA under Contract F30602–98–2–0144, and by NASA under Contract 276685/NAS5–00212. *Mathematica*® fonts by Wolfram Research, Inc.

[1] The outputs in this paper can be exactly reproduced by typing the code shown against each *In[n]:=* prompt into a notebook interface to version 1.0 of MAGICA, running on Mathematica 4.1.

G. Hedin (Ed.): CC 2003, LNCS 2622, pp. 121–125, 2003.

assignment statements, the `for` and `while` loops, and the `if` conditional statement. (All these MATLAB constructs are explained in [3].) In addition, MAGICA can handle close to 70 built-in functions in MATLAB. These include important Type II operations[2] like `subsref`, `subsasgn` and `colon` that are used in array indexing and colon expressions. For the most part, the full or nearly the full semantics of a built-in function, as specified in [3], is supported. For instance, subscripts in array indexing expressions can themselves be arrays, and arrays can be complex-valued. Not all of MATLAB's features are currently handled; these include structures, cell arrays and recent additions like function handles.

2 Representing MATLAB in MAGICA

MAGICA symbolically represents constructs in MATLAB. An example of this is the Mathematica expression `plus[a, b]`, which is MAGICA's representation of the MATLAB expression `a+b`. On line *In[1]* above, the Mathematica expression `tanh[3.78ii]` was used to denote the MATLAB expression `tanh(3.78i)`. The idea of functionally representing a MATLAB expression can also be used to denote high-level constructs. For instance, the MATLAB assignment statement `l ← cos(3.099)`, where `l` is a MATLAB program variable, is represented in MAGICA as shown on line *In[2]* below.

```
In[2]:= assignment[$$lhs :⇢ l, $$rhs :⇢ cos[3.099]]
Out[2]= assignment($$lhs :⇢ l, $$rhs :⇢ cos(3.099))
```

The expression's *head* is `assignment` and this is used to uniquely identify MATLAB assignments. The *tags* `$$lhs` and `$$rhs` serve to identify the assignment's left-hand side and right-hand side. We call `l` and `cos[3.099]` as *tag values*. A tag value can be any expression; this allows for the representation of arbitrary MATLAB assignments, including the multiple-value assignment [3].

In general, MATLAB statements are represented in MAGICA as

$$h[x_1 :\to y_1, x_2 :\to y_2, \ldots, x_n :\to y_n]$$

where the head h serves as a construct identifier, and where the delayed rules [4] $x_i :\to y_i$ $(1 \leq i \leq n)$ stand for *tag-value pairs*. MAGICA places no significance on the position of a tag-value pair; this point should be kept in mind when making new definitions to extend the MAGICA system. A fair amount of documentation regarding data structure layouts has been coded into MAGICA itself as `usage` messages [4]; this provides a convenient, on-line way of pulling up layout information while interacting with MAGICA.

[2] MATLAB's built-in functions can be classified into one of three groups, based on how the shapes of the outputs are dependent on the shapes of the inputs [1]. Type I built-ins produce outputs whose shapes are completely determined by the shapes of the arguments, if any. Type II built-ins produce an output whose shape is *also* dependent on the elemental values of at least one input. All remaining built-ins fall into the Type III group.

```
In[3]:= ?if
```

```
if[$$condition :> c_, $$then :> t_, $$else :> e_]
  is the functional equivalent of an if statement in MATLAB. Forms such as
  $$condition -> c , $$then -> t and $$else -> e can also be used.
```

3 Transitive Closure of a Graph

The two boxes in Figure 1 display a complete MATLAB program that computes the transitive closure of a graph. The graph is represented in the $N \times N$ adjacency matrix `A`, which is initialized arbitrarily in the function `tclosure`. Its transitive closure is returned in `B`. The shown code is directly from Alexey Malishevsky's thesis [2], with three nontrivial changes: (1) the `tic` and `toc` timing commands were removed, (2) `disp` was used to display `B`, and (3) the original monolithic script was reorganized into two files, one containing the function `driver` and the other containing `tclosure`.

3.1 M-File Contexts

Input files that constitute a MATLAB program are referred to as *M-files* in MATLAB parlance. Every M-file has its own parsed representation in MAGICA, which we call an *M-file context*. Through Mathematica's information-hiding context mechanism [4], MAGICA provides a way to save, and later retrieve, the M-file contexts of a MATLAB program. On line *In[4]* below, the M-file contexts of the two user-defined functions `driver` and `tclosure`, saved in an earlier session of MAGICA, are loaded from disk.[3]

```
In[4]:= Scan[load[#, load$Disk → True]&, {"tclosure`", "driver`"}]
```

An M-file context is basically a collection of Mathematica definitions that capture information about a user-defined MATLAB function. As an example, for a user-defined function f, a definition is made against the `statements` function object so that `statements[`f`]` expands to the function body of f. This is how the `type` object operates on the statements in the body of `driver` on line *In[5]* below.[4]

```
In[5]:= type[statements[driver]] // Timing
Out[5]= {5.47 Second, {_12 N1 → {512., $integer, {⟨1 ,  1⟩, 2}},
          _10 B1 → {⟦0, 1⟧, $boolean, {⟨512 ,  512⟩, 2}},
          {Indeterminate, $illegal, {⟨-1 ,  1⟩, 2}}}}
```

[3] These representations are in ASCII, and can be manually or automatically generated.
[4] The timings are on a 440 MHz Solaris 7 UltraSPARC-IIi having 128MB of main memory.

```
function driver

N ← 512;

B ← tclosure(N);

disp(B);
```

```
function B = tclosure(N)
% Initialization.
A ← zeros(N, N);
for ii = 1:N,
  for jj = 1:N,
    if ii*jj < N/2,
      A(N-ii, ii+jj) ← 1;
      A(ii, N-ii-jj) ← 1;
    end;
    if ii == jj,
      A(ii, jj) ← 1;
    end;
  end;
end;
B ← A;
% Closure.
ii ← N/2;
while ii >= 1,
  B ← B*B;
  ii ← ii/2;
end;
B ← B > 0;
```

Fig. 1. The MATLAB Transitive Closure Program

3.2 Interprocedural Type Inference

The definitions against the `type` object—currently over a 100—take care of propagating information across user-defined function interfaces. Thus the application of `type` on line *`In[5]`* causes type information pertaining to N to be propagated into `tclosure`, resulting in the shown type inference for its output variable `B`. On line *`Out[5]`*, `_12N1` stands for N and `_10B1` for `B`; this renaming is an artifact of the way in which the Mathematica representations for this program were automatically generated. *`Out[5]`* thus shows that the value range of `B` is $[\![0, 1]\!]$, its intrinsic type is `$boolean`, and that its shape is 512×512. The third inference on *`Out[5]`* represents the type of `disp`'s outcome; the shown values reflect the fact that `disp` doesn't return anything.

4 Architecture

MAGICA is used through a *front-end*, which is a separate operating system process that builds a Mathematica representation of an input MATLAB program. The front-end transfers the representation to MAGICA for type analysis; the type inferences that MAGICA generates are transferred back, for use in type-related optimizations, code generation or simply for code annotation and visualization. Exchanges between the front-end and MAGICA happen across an interprocess communication link using the MathLink protocol [4]. Figure 2 shows three existing front-ends to MAGICA. Two of these—the GUI-based notebook and the text-based interface—are shipped with Mathematica. Interacting with MAGICA using them requires either the handcrafting of the program representations that are to be type inferred, or the availability of those representations on disk. (*`In[1]`* and *`In[2]`* are examples of handcrafted representations; *`In[4]`* uses prefabricated representations.) The third front-end, called $\mathbb{M}^{\text{AT}}\mathbb{C}$, is a custom-built one that takes a MATLAB program in its native form and translates it to optimized C; it uses MAGICA as the inference engine to obtain the necessary type information. In fact, it was by using $\mathbb{M}^{\text{AT}}\mathbb{C}$ that the M-file contexts for the example in

§ 3 were produced in advance. Figure 2 also shows the disk image of a sample M-file context.

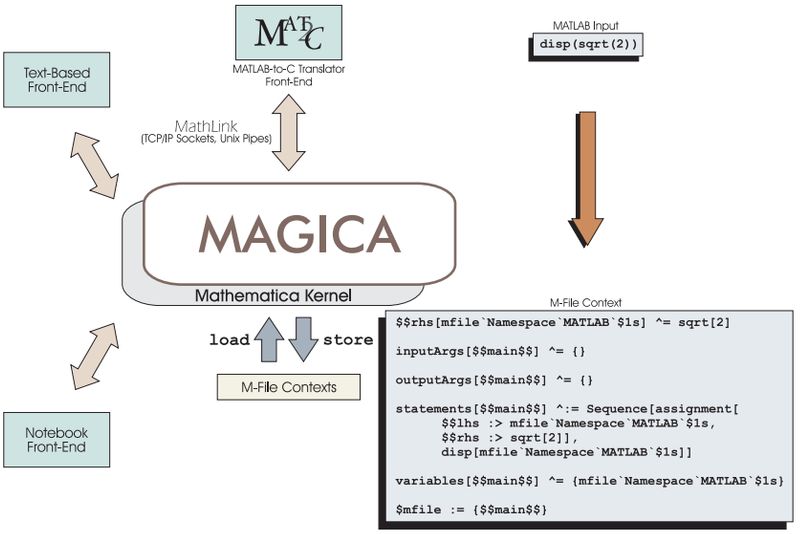

Fig. 2. The MAGICA Architecture

5 Summary

This paper briefly introduced a software tool called MAGICA that infers value ranges, intrinsic types and array shapes for the MATLAB programming language. Though shown in an interactive mode in this paper, MAGICA can also be used in a batch mode from a custom front-end. Currently, MAGICA is being used this way by MAT2C, a MATLAB-to-C translator that converts a MATLAB source to optimized C code.

References

1. P.G. Joisha, U.N. Shenoy, ad P. Banerjee. "An Approch to Array Shape Determination in MATLAB". Technical report CPDC-TR-2000-10-010, Department of Electrical and Computer Engineering, Northwestern University, October 2000.
2. A. Malishevsky. "Implementing a Run-Time Library for a Parallel MATLAB Compiler". M.S. report, Oregon State University, April 1998.
3. The MathWorks, Inc. *MATLAB: The Language of Technical Computing*, January 1997. Using MATLAB (Version 5).
4. S. Wolfram. The Mathematica Book, 4th ed. Wolfram Media, Inc., 1999.