LabMate: Supporting Types for Matlab

Conor McBride[1], Georgi Nakov[1], Fredrik Nordvall Forsberg[1],
André Videla[1], Alistair Forbes[2], Keith Lines[2]

[1]University of Strathclyde, UK
[2]National Physical Laboratory, UK

August 26, 2024

The Problem

- Loads of computational software in science and engineering is written in MATLAB
  - may contain errors and bugs, as with any software
- Developers often leave comments what are the corresponding physical systems of their data and how it should be interpreted, e.g., units of measure for quantities.
- MATLAB is oblivious to these high-level, semantic comments, and instead performs low-level compatibility checks during execution.

Our Plan

Can we do better?

- Make these developers' comments formal
- . . . and create a tool to make use of them — LabMate
  - keep the existing MATLAB code and toolchains, no need to rewrite in a new language
- Distill the essence of the developers' comments in LabMate's expressive type system
  - a set of logical rules that assign a domain of admissible values to the expressions in our program
- Run LabMate multiple times while writing the code to get instant feedback and guidance, do not delay until execution

How does LabMate Work?

- LabMate is a program transducer: reads Matlab code with formal comments, and outputs a modified version of the input
- These formal comments are directives — they start with %<
- Input the program:

```
%> rename n x
n = 5;
display(n);
```

How does LabMate Work?

- LabMate is a program transducer: reads Matlab code with formal comments, and outputs a modified version of the input
- These formal comments are directives — they start with %<
- Input the program:

```
%> rename n x
n = 5;
display(n);
```

this is an input directive to rename a variable

How does LabMate Work?

- LabMate is a program transducer: reads Matlab code with formal comments, and outputs a modified version of the input
- These formal comments are directives — they start with %<
- Input the program:

```
%> rename n x
n = 5;
display(n);
```

How does LabMate Work?

- LabMate is a program transducer: reads Matlab code with formal comments, and outputs a modified version of the input
- These formal comments are directives — they start with %<
- Input the program:

```
%> rename n x
n = 5;
display(n);
```

- Run LabMate to get:

```
%< LabMate 0.2.0.0
%< renamed n x
x = 5;
display(x);
```

How does LabMate Work?

- LabMate is a program transducer: reads Matlab code with formal comments, and outputs a modified version of the input
- These formal comments are directives — they start with %<
- Input the program:

```
%> rename n x
n = 5;
display(n);
```

- Run LabMate to get:

```
%< LabMate 0.2.0.0
%< renamed n x
x = 5;
display(x);
```

LabMate response to the input directive

How does LabMate Work?

- LabMate is a program transducer: reads Matlab code with formal comments, and outputs a modified version of the input
- These formal comments are directives — they start with %<
- Input the program:

```
%> rename n x
n = 5;
display(n);
```

- Run LabMate to get:

```
%< LabMate 0.2.0.0
%< renamed n x
x = 5;
display(x);
```

mark the file as processed by LabMate

Matrix Types

- Matrices feature heavily in MATLAB code

Matrix Types

- Matrices feature heavily in MATLAB code
- LABMATE support type annotations for matrices

```
%> A :: [ 1 x 2 ] int
A = [ 2 3 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      3 4 5 6 ]
C = A * B
```

Matrix Types

- Matrices feature heavily in MATLAB code
- LABMATE support type annotations for matrices

```
%> A :: [ 1 x 2 ] int
A = [ 2 3 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      3 4 5 6 ]
C = A * B
```

Matrix Types

- Matrices feature heavily in MATLAB code
- LABMATE support type annotations for matrices

```
%> A :: [ 1 x 2 ] int
A = [ 2 3 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      3 4 5 6 ]
C = A * B
```

type annotations
at declaration of A

Matrix Types

- Matrices feature heavily in MATLAB code
- LABMATE support type annotations for matrices

```
%> A :: [ 1 x 2 ] int
A = [ 2 3 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      3 4 5 6 ]
C = A * B
```

type annotations
at declaration of B

Matrix Types

- Matrices feature heavily in MATLAB code
- LABMATE support type annotations for matrices

```
%> A :: [ 1 x 2 ] int
A = [ 2 3 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      3 4 5 6 ]
C = A * B
```

- We can ask for type information

```
%> typeof C
%< C :: [Matrix 1 4 int]
```

Matrix Types

- Matrices feature heavily in MATLAB code
- LABMATE support type annotations for matrices

```
%> A :: [ 1 x 2 ] int
A = [ 2 3 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      3 4 5 6 ]
C = A * B
```

- We can ask for type information

```
%> typeof C
%< C :: [Matrix 1 4 int]
```

Matrix Types

- Matrices feature heavily in MATLAB code
- LABMATE support type annotations for matrices

```
%> A :: [ 1 x 2 ] int
A = [ 2 3 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      3 4 5 6 ]
C = A * B
```

- We can ask for type information

```
%> typeof C
%< C :: [Matrix 1 4 int]
```

query for the type of C

Matrix Types

- Matrices feature heavily in MATLAB code
- LABMATE support type annotations for matrices

```
%> A :: [ 1 x 2 ] int
A = [ 2 3 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      3 4 5 6 ]
C = A * B
```

- We can ask for type information

```
%> typeof C
%< C :: [Matrix 1 4 int]
```

LabMate can infer the dimensions

Matrix Types

- Matrices feature heavily in MATLAB code
- LABMATE support type annotations for matrices

```
%> A :: [ 1 x 2 ] int
A = [ 2 3 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      3 4 5 6 ]
C = A * B
```

- We can ask for type information

```
%> typeof C
%< C :: [Matrix 1 4 int]
```

- . . . and can easily spot incompatible sizing

```
D = B * A
%> typeof D
%< The expression D is quite a puzzle
```

Matrix Types

- Matrices feature heavily in MATLAB code
- LABMATE support type annotations for matrices

```
%> A :: [ 1 x 2 ] int
A = [ 2 3 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      3 4 5 6 ]
C = A * B
```

- We can ask for type information

```
%> typeof C
%< C :: [Matrix 1 4 int]
```

- . . . and can easily spot incompatible sizing

```
D = B * A
%> typeof D
%< The expression D is quite a puzzle
```

Matrix Types

- Matrices feature heavily in MATLAB code
- LABMATE support type annotations for matrices

```
%> A :: [ 1 x 2 ] int
A = [ 2 3 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      3 4 5 6 ]
C = A * B
```

- We can ask for type information

```
%> typeof C
%< C :: [Matrix 1 4 int]
```

- . . . and can easily spot incompatible sizing

```
D = B * A
%> typeof D
%< The expression D is quite a puzzle
```

LabMate can point out an error with D

Matrix Types

- LabMate processes arbitrary Matlab code

Matrix Types

- LabMate processes arbitrary Matlab code
  - LabMate does not rely on constant values, works on variables as well

Matrix Types

- LabMate processes arbitrary Matlab code
  - LabMate does not rely on constant values, works on variables as well

```
function B = f(A)
  %> B :: [ 1 x 3 ] int
  B = [ 1 A ]

  %> typeof A
  %< A :: [Matrix int 1 2]
end

%> typeof A
%< The expression A is quite a puzzle
```

Matrix Types

- LabMate processes arbitrary Matlab code
  - LabMate does not rely on constant values, works on variables as well

```
function B = f(A)
  %> B :: [ 1 x 3 ] int
  B = [ 1 A ]

  %> typeof A
  %< A :: [Matrix int 1 2]
end

%> typeof A
%< The expression A is quite a puzzle
```

Matrix Types

- LabMate processes arbitrary Matlab code
  - LabMate does not rely on constant values, works on variables as well

```
function B = f(A)
  %> B :: [ 1 x 3 ] int
  B = [ 1 A ]

  %> typeof A
  %< A :: [Matrix int 1 2]
end

%> typeof A
%< The expression A is quite a puzzle
```

LabMate infers A from the type annotation on B

Matrix Types

- LabMate processes arbitrary Matlab code
  - LabMate does not rely on constant values, works on variables as well

```
function B = f(A)
  %> B :: [ 1 x 3 ] int
  B = [ 1 A ]

  %> typeof A
  %< A :: [Matrix int 1 2]
end

%> typeof A
%< The expression A is quite a puzzle
```

tracks Matlab scope

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q({ `Mass })
```

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q({ `Mass })
```

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q(1 `Mass)
```

define some base set of dimensions

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q({ `Mass })
```

and a canonical unit of measure

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q({ `Mass })
```

can be arbitrary group expression over V

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q({ `Mass })
```

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q({ `Mass })

%<{
kg = 1;
%<}
```

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q({ `Mass })

%<{
kg = 1;
%<}
```

this is a "magic" response that LabMate emits

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q({ `Mass })

%<{
kg = 1;
%<}
```

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q({ `Mass })

%<{
kg = 1;
%<}
```

- we can then use the unit of measure to create quantities in our program:

```
y = 5*kg
%> typeof y
%< y :: Quantity (Enum [`Mass, `Time, `
   Length]) {`Mass}
```

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q({ `Mass })

%<{
kg = 1;
%<}
```

- we can then use the unit of measure to create quantities in our program:

```
y = 5*kg
%> typeof y
%< y :: Quantity (Enum [`Mass, `Time, `
   Length]) {`Mass}
```

Dimensions and Quantities

- LabMate has support for arbitrary quantities

```
%> dimensions V for Q over `Mass, `Time,
   `Length
%> unit kg :: Q({ `Mass })

%<{
kg = 1;
%<}
```

- we can then use the unit of measure to create quantities in our program:

```
y = 5*kg
%> typeof y
%< y :: Quantity (Enum [`Mass, `Time, `
   Length]) {`Mass}
```

turn a value of a dimensionless type into a quantity

Dimensional Consistency for Matrices

- Most MATLAB program does not work on uniform matrices, the type of the entry $e_{i,j}$ might depend on the indices i and j

Dimensional Consistency for Matrices

- Most MATLAB program does not work on uniform matrices, the type of the entry $e_{i,j}$ might depend on the indices i and j
- A common scenario when working with matrices of quantities

```
%> dimensions V for Q over `Length, `Mass
   , `Time
%> unit metre :: Q({ `Length })
%> unit kg :: Q({ `Mass })
%> unit sec :: Q({ `Time })
```

Dimensional Consistency for Matrices

- Most MATLAB program does not work on uniform matrices, the type of the entry $e_{i,j}$ might depend on the indices i and j
- A common scenario when working with matrices of quantities

```
%> dimensions V for Q over `Length, `Mass
   , `Time
%> unit metre :: Q({ `Length })
%> unit kg :: Q({ `Mass })
%> unit sec :: Q({ `Time })
```

- Work in progress: LABMATE support for such matrices

```
% > x :: [ i <- [{} {`Time}]
%          x j <- [{} {`Length}]
%          ] Q({`Mass * j / i})
x = [ 2*kg          5*kg*metre
      3*kg/sec      4*kg*metre/sec ]
```

Implementation Details

- MATLAB program are modelled as trees of commands, rather than sequence of commands

Implementation Details

- MATLAB program are modelled as trees of commands, rather than sequence of commands
  - the type information is propagated (consistently) throughout the tree; can put type annotations after variable declaration

Implementation Details

- MATLAB program are modelled as trees of commands, rather than sequence of commands
  - the type information is propagated (consistently) throughout the tree; can put type annotations after variable declaration
  - not every annotation is needed, document the interesting ones

Implementation Details

- MATLAB program are modelled as trees of commands, rather than sequence of commands
  - the type information is propagated (consistently) throughout the tree; can put type annotations after variable declaration
  - not every annotation is needed, document the interesting ones
- MATLAB expressions are translated to LABMATE internal core type theory:

Implementation Details

- MATLAB program are modelled as trees of commands, rather than sequence of commands
  - the type information is propagated (consistently) throughout the tree; can put type annotations after variable declaration
  - not every annotation is needed, document the interesting ones
- MATLAB expressions are translated to LABMATE internal core type theory:
  - Matrix types are parametrised over 5 parameters with dependencies between them

Implementation Details

- MATLAB program are modelled as trees of commands, rather than sequence of commands
  - the type information is propagated (consistently) throughout the tree; can put type annotations after variable declaration
  - not every annotation is needed, document the interesting ones
- MATLAB expressions are translated to LABMATE internal core type theory:
  - Matrix types are parametrised over 5 parameters with dependencies between them
  - Quantities are modelled as the free Abelian group over a base set of dimensions

Implementation Details

- MATLAB program are modelled as trees of commands, rather than sequence of commands
  - the type information is propagated (consistently) throughout the tree; can put type annotations after variable declaration
  - not every annotation is needed, document the interesting ones
- MATLAB expressions are translated to LABMATE internal core type theory:
  - Matrix types are parametrised over 5 parameters with dependencies between them
  - Quantities are modelled as the free Abelian group over a base set of dimensions
  - The typechecker understands some nontrivial algebraic properties

Current Progress & Future Plans

- LabMate is under active development

Current Progress & Future Plans

- LabMate is under active development
  - available on GitHub, please get it in touch if interested

Current Progress & Future Plans

- LabMate is under active development
  - available on GitHub, please get it in touch if interested
- Work in the pipeline:

Current Progress & Future Plans

- LabMate is under active development
  - available on GitHub, please get it in touch if interested
- Work in the pipeline:
  - Uniqueness of representation: currently, a matrix with quantities can have more than one corresponding type; this might lead to odd behaviour during typechecking

Current Progress & Future Plans

- LabMate is under active development
  - available on GitHub, please get it in touch if interested
- Work in the pipeline:
  - Uniqueness of representation: currently, a matrix with quantities can have more than one corresponding type; this might lead to odd behaviour during typechecking
  - Quality of life improvements: better messages and more readable responses from LabMate

Current Progress & Future Plans

- LabMate is under active development
  - available on GitHub, please get it in touch if interested
- Work in the pipeline:
  - Uniqueness of representation: currently, a matrix with quantities can have more than one corresponding type; this might lead to odd behaviour during typechecking
  - Quality of life improvements: better messages and more readable responses from LabMate
- We want to extend our coverage to loops and conditionals in the future.

Current Progress & Future Plans

- LabMate is under active development
  - available on GitHub, please get it in touch if interested
- Work in the pipeline:
  - Uniqueness of representation: currently, a matrix with quantities can have more than one corresponding type; this might lead to odd behaviour during typechecking
  - Quality of life improvements: better messages and more readable responses from LabMate
- We want to extend our coverage to loops and conditionals in the future.

Current Progress & Future Plans

- LabMate is under active development
  - available on GitHub, please get it in touch if interested
- Work in the pipeline:
  - Uniqueness of representation: currently, a matrix with quantities can have more than one corresponding type; this might lead to odd behaviour during typechecking
  - Quality of life improvements: better messages and more readable responses from LabMate
- We want to extend our coverage to loops and conditionals in the future.