Automatic translation of Matlab to Python 3

Patrik Grman Supervisor: RNDr. Jana Kostičová PhD.

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

26.08.2021

Tranlators

automated tools

convert code from one programming language to another

equivalent functionality

work around language differences

Matlab2Python

available on GitHub as Matlab2Python

created for the year project

from MATLAB to Python3

fixed limited input set

attempted natural-looking output

ANTLR4 + StringTemplate4

Antlr4

on GitHub as Antlr grammars-v4

mostly context–free

lexer and parser rules

special sections

predicates

Grammar features

```
fragment NUM : [0-9] | [1-9][0-9]+ ;
NUMBER : ('+' | '-')? NUM ;
WS : [ \t] + -> skip;
STRING_LITERAL
    : {isStringPossible()}?
        '\'' ( ~ ( '\'' | '\n' ) ) * '\'' ;
expression: NUMBER
  | '(' expression ( '+' expression ) + ')'
  ;
array_list: array_element
    ({aWhitespace()}? array_element) * ;
```

Grammar

was ambiguous

“2.3 You Can’t Put Too Much Water into a Nuclear Reactor”

removed rule

incorrect parse results

modified rule priority

consider whitespace in list definitions

Code

fix tests for CRLF systems

match the modified grammar

“Do not repeat yourself”

visitor factory

Name detection

arrays vs functions

MATLAB (_) (_)

Python [_ -1] (_)

guess by context

Constant evaluation

indexing difference

MATLAB 1

Python 0

$a(b, 2)$

$a[((b) - 1), ((2) - 1)]$

$a[((b) - 1), 1]$

New built–in

goniometric functions

sin cos tan

asin acos atan

cot as $1/tan$

PyPy

CPython alternative with JIT

pypy.org

"If you want your code to run faster, you should probably just use PyPy."

Guido van Rossum

Results

loop sum

PyPy 100× CPython

Fibonacci

PyPy 14× CPython

matrix multiplication

PyPy 0.6× CPython

Operation

Fragment

build time generate parser

process arguments

parse into tree

translate Visitor

render and output

The problems

required whitespace ignored in lists

array_element had 2 rules, *array_expression*, *expression*

array_expression should have been useless

$$expression \rightarrow or\_expression \rightarrow and\_expression \rightarrow$$
$$\rightarrow equality\_expression \rightarrow relational\_expression \rightarrow$$
$$\rightarrow additive\_expression \rightarrow multiplicative\_expression \rightarrow$$
$$\rightarrow array\_mul\_expression \rightarrow unary\_expression \rightarrow$$
$$\rightarrow postfix\_expression \rightarrow array\_expression$$

The solutions

removed *array_expression* option

modified *postfix_expression* to adjust rule priority

tests verify correct behaviour

added a parser predicate for whitespace checking

modified rule to enforce whitespace in lists

$$array\_element(\{aWhitespace()\}?array\_element)*$$

```
private boolean aWhitespace(){
        TokenStream ts=getTokenStream();
        int type=ts.get(ts.index()-1).getType();
        return (type==WS)||
                (type==LINE_CONTINUATION);
}
```

The problem

reusing a single visitor

name detection requires statefulness

potential parallel translation

The solution

added `PTVFactory`

handles `debug`

`getNew`

modified runner, tests

The problem

no syntactic difference in MATLAB

`unknown(1)`

different syntax in Python

`unknown[0]` array

`unknown(1)` function

Original algorithm

left–hand side is indexing

not a known built–in is indexing

known built–in is specific

The solution

added `IdentifierType`, `IdentifierTypeStorage`

save when translating assignment

left–hand side must be *IDENTIFIER*

lambda means function, otherwise assumed array

used for not-builtin names

added a test

factory provides storage clones

The problem

different start index

different range bounds

correct by subtracting or adding 1

$a(b, 2)$

$a[((b) - 1), ((2) - 1)]$

The solution

examine subtree

if single integer leaf evaluate

otherwise as previously

$a(b, 2)$

$a[((b) - 1), 1]$

same for range bounds, + 1

MATLAB source

```
a = sin(5);

myFun = @(x) x+1;
myArr = [1 2 3];

myFun(2);
myArr(2);
```

Before

```
import numpy as np

def array(arg):
    ...

a = sin[(5 - 1)]
myFun = lambda x: x + 1
myArr = array([1, 2, 3])
myFun[(2 - 1)]
myArr[(2 - 1)]
```

After

```
import numpy as np

def array(arg):
    ...

a = np.sin(5)
myFun = lambda x: x + 1
myArr = array([1, 2, 3])
myFun(2)
myArr[1]
```

What we achieved

more built–ins

better translations

constant evaluation

name detection

cleaner code

Future work

CLI options for name lists

tic, toc

better switch

multi–function files

parse failures on multiline calls

Other translators

SMOP

emulation, not strict translation

“There is an attempt to follow the original matlab semantics as close as possible. Matlab language definition (never published afaik) is full of dark corners, and SMOP tries to follow matlab as precisely as possible.”

SMOP

```
a = sin_(5)

myArr = matlabarray([1, 2, 3])
myArr[2]
```

Bracket problem

SMOP

similar as we implemented

“To figure out which is which, SMOP computes local use-def information, and then applies the following rule: undefined names are functions, while defined are arrays.”

Questions