Damo

A language for symbolic functions

Ian Covert  Abhiroop Gangopadhyay  Srihari Devaraj  Alan Gou

Project Manager  Language Guru  System Architect  Testing
I. Introduction

Why...
CORE FEATURES

- Scripting language
- Symbolic expressions
- Standard library for symbolic function evaluation, automatic differentiation
MOTIVATION

- Ease of development for applications requiring automatic differentiation
- Useful for many kinds of machine learning, such as SGD algorithm for neural networks
- Historical note:
  - Damo was child of Theano – a popular Python deep learning library
II. Project Management
Responsibilities, lessons learned
## EVERYONE BECOMES A DEVELOPER

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abhiroop</td>
<td>Built SAST, semantic checking and tests</td>
</tr>
<tr>
<td>Ian</td>
<td>Implemented parser, standard library and tests</td>
</tr>
<tr>
<td>Hari</td>
<td>Implemented codegen and tests</td>
</tr>
<tr>
<td>Alan</td>
<td>Took the worst classes of his life this semester</td>
</tr>
</tbody>
</table>
LESSONS LEARNED

- Iterative development makes life easier
- Specialization is helpful, but dangerous
- Realistic deadlines are necessary
- Never assume that something works
III. The Damo Language
Speaking our dialect
SYNTAX BASICS

// Single line comment

	/*
	  Multiline comment
	*/

// VARIABLE DECLARATION
int i;
int j = 1;

	/*
OPERATORS
+ - * / ^ _ %
and or not
< > <= >= == !=

TYPES
int
num
bool
string
symbol
*/
print(“Hello world”);

// FUNCTION DECLARATION
def sayHello(string name) : void {
    print(“Hello, “);
    print(name);
}

sayHello(“Stephen”);

num a;
num b;
num c;

a = 1;
b = 2.0;
c = a * b;

print_num(c);
CONTROL FLOW

// C-like loops

int i;

print("Going up");
for (i = 0; i < 10; i = i + 1){
    print_int(i);
}

print("Going down");
while (i > 0){
    print_int(i);
    i = i - 1;
}

// C-like if-elseif-else statements

if (i < 0){
    print("i less than 0");
}
elseif (i < 10){
    print("i less than 10");
}
else {
    print("i greater than 10");
}
// Declare symbols
symbol a;
symbol b;
symbol c;

// Set symbolic expression
a = b + c;
a = a * (b - c);

// Set symbols to constant values
b = 4;
c = 5;
// Function evaluation
symbol a; symbol b; symbol c;

a = b * c;
b = 4;
c = 5;

num result = eval(a);
num deriv = partialDerivative(a, b);
IV. How it works
Implementation details
## OUR COMPILATION PIPELINE

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source code</strong></td>
<td>Program written in Damo</td>
</tr>
<tr>
<td><strong>Linked with stdlib</strong></td>
<td>Standard library prepended</td>
</tr>
<tr>
<td><strong>Symbols</strong></td>
<td>Scanner</td>
</tr>
<tr>
<td><strong>AST</strong></td>
<td>Parser</td>
</tr>
<tr>
<td><strong>SAST</strong></td>
<td>Semantic checking</td>
</tr>
<tr>
<td><strong>LLVM</strong></td>
<td>Codegen</td>
</tr>
<tr>
<td><strong>Executable</strong></td>
<td>C compiler: links with C code</td>
</tr>
</tbody>
</table>
ABSTRACT SYNTAX TREE

- A Damo program is a list of variable declarations, function declarations, and statements
- Arbitrary ordering
- Semantic checking verifies proper usage of variables and functions
Invoke C functions to allocate heap memory

```python
def symbol_malloc = Lllvm.declare_function "createSymbol" (Lllvm.function_type symbol_t [] [] the_module
...
A.Symbol -> let global_variable = L.build_call symbol_malloc [] [] "symbolmal" builder in ignore(L.build_store global_variable s_v builder);
```
THE SYMBOL STRUCT

- Underlying C struct represents symbol type

```c
struct symbol {
    symbol *left;
    symbol *right;
    int isConstant;
    int isInitialized;
    double value;
};
```
LINKING WITH C CODE

- Makefile builds symbol.c, a library we wrote to handle routines relating to symbols
  - Heap memory allocation
  - Accessor, mutator functions
- Damo executables are linked with C standard library, and symbol.o
V. Testing

It works, we promise
UNIT AND INTEGRATION TESTS

- We tested for every feature of the Damo language
  - Operators
  - Functions
  - Global variables
  - Standard library functions
  - Etc.
THE ULTIMATE TEST

- Showing off in our demo – a big integration test
Let’s demo it!