Extend

Language Reference Manual

Ishan Kolluri, Kevin Ye, Jared Samet, Nigel Schuster

October 20, 2016

Contents

1. Introduction to Extend

2. Types and Literals

2.1. Primitive Data Types

2.2. Ranges

2.2.1. Range Slicing

2.2.2. The Underscore Symbol

2.3. Integers and Strings

2.3.1. Unary Operations

3. Expressions

3.1. Operators

3.1.1. Multiplication

3.1.2. Addition

3.1.3. Unary

3.2. Booleans

3.3. Variable Declaration

3.4. Variable Assignment

3.5. Conditionals

4. Functions

Functions lie at Extend's core; however, they are not *first class objects*. Since it can be verbose to write certain operations in Extend, the language will feature a comprehensive number of built-in

and standard library function. An important built-in function will be I/O (see section **??**).

4.1. Format

Every function in Extend follows the same format, but allows some optional declarations. As in most programming languages the header of the function declares the parameters it accepts and the return type. The simplest function is this:

```
[1,1] foo([1,1] arg) {
  return arg;
}
```

This function simply returns whatever value is passed into it. The leading `[1,1]` marks the return dimensions. `foo` is the function name. In parentheses the function arguments are declared, again with dimensions of the input. The body of the function follows, which in this case is only the return statement.

4.2. Dimension Assignment

Extend will feature gradual typing for function declarations. This will enable users with a weak experience in typing to use the language, but maintains the ability to improve during development.

To avoid specifying the return dimensions, an underscore can be used. This marks a variable range. Thus our function now looks like this:

```
[_,1] foo([5,5] arg1, [1,1] arg2) {
  return arg1[0:arg2 ,0];
}
```

Here we are selecting a range from arg1 that depends on the value of arg2 and can therefore not be known ahead of time.

However Extend will feature even more options to specify ranges. If a certain operation should be applied to a range of numbers of unknown size, the size can be inferred at runtime and match the return size:

```
[m,1] foo([m,1] arg) {
  return arg[0:m-1 ,0] + 1;
}
```

This function will add 1 to each element in arg. Notice, that `m` is used across the function to apply the operation to the range.

Summarizing, we have 3 ways of specifying a return range:

Type	Symbol Example	Description
Number	3	A number is the simplest descriptor. It specifies the absolute return size
Variable	bar	A variable identifier. To use this, the identifier must also be present as a range descriptor in a function parameter.
Underscore	_	This marker is unique, since it is a wildcard. While the other options aim to be specific, the underscore circumvents declaring the range size.

4.3. Application across Dimensions

4.4. Dependencies Illustrated

5. I/O

5.1. File

5.2. Input Arguments

5.2.1. How to run a program

5.2.2. Main function

6. Example Program