SetC

A Language for Set Theory

Heather Preslier
Introduction/Motivation

- Language based in C
- Compiled down to LLVM
- Goals:
  - Simplify the formulation of complex algorithms by creating a concise language that mirrors set notation
  - Simplify the handling and manipulating of sets
  - Remove the need for type declarations
  - Maintain functionality for basic programming
Syntax

- Syntax inspired by set theoretic notation
- Full type inference
- Built-in functionality for the manipulation and operation of sets
- Overloaded operators

```python
def remove_duplicates(a):
    b = [];
    (0<=i<#a | a[i]?b == false)
    b = b + [a[i]];
    print(b); /* print the new set */
    return b;
```

Syntax

- Syntax inspired by set theoretic notation
- Full type inference
- Built-in functionality for the manipulation and operation of sets
- Overloaded operators

```python
def remove_duplicates(a):
    b = [];
    (0<=i<#a | a[i]?b == false)
    b = b + [a[i]];
    print(b); /* print the new set */
    return b;
```
Syntax

- File extension: .sc
- Required main function
- Types: bools, ints, floats, strings, sets
- Initialization by assignment
  - Inferred types are bound after initialization
  - Empty set types are bound after first use
- Functions
  - Need only be preceded by the keyword def
  - Functions types and parameters are inferred
  - Sets are pass by reference

example1.sc:

```sc
1 a = "setc"; /* global */
2
3 def main(){
4 b = 3; /* initialization */
5 c = [];
6 c = c + [b]; /* c -> int */
7 d = func(c);
8 print(d); /* true */
9
10 def func(a) {
11 a[0] = 5;
12 return true;
13 }
```
Special Features

- **Optimizations of functions**
  - Some functions will not be semantically checked and code will not be generated

- **Standard Library/Built-In Functionality**
  - Print functionality for all types
  - Intersection (*), union (&), difference(-), append(+), slice(:), set, in(?), cardinality(#) operations for sets
  - Split function: string -> set
  - File I/O: open, close, read, write

```sc
example2.sc:
1   def main(){
2     a = [1,2,3];
3     b = [1,4,2];
4     a * b;  // [1,2] *
5     a + b;  // [1,2,3,1,4,2]*
6     a & b;  // [1,2,3,4] */
7     a - b;  // [3] */
8     a[1:3]; // [2,3] */
9     2?a;   // true */
10    a[1:3]; // [2,3] */
11    #a;    // 3 */
12    print(a);  // 1 2 3 */
13    c = "hello world";
14    d = split(c, " ");
15    /* d -> set */
```
Tasks

Semantics:
● Constraints
  ○ Precedence
● Overloading operators and functions
  ○ Considering the IR
● Type Inference algorithm
  ○ Third time's the charm
● Sets/empty set
  ○ Type inference of an empty set
  ○ Compile time vs. runtime decision

Code Generation:
● Sets
  ○ Pointer implementation
  ○ Length: compile time vs. runtime decision

Testing:
● Unit and Integration testing
● 102 tests total
Lessons Learned

- Need to consider the IR more when making design and implementation choices for the language
Demos

**Demo 1**: Basic functionality
- Bubble Sort Algorithm

**Demo 2**: Function inference
- GCD algorithm
- Euler’s phi function
- Coprimality of sets

**Demo 3**: Algorithm
- Perceptron Learning Algorithm