SetC: A Concise Set Language

Julian Kocher (jk3813), Frank Ling (ffl2107), Heather Preslier (hnp2108)

February 8, 2017

1 Introduction

The motivation for writing a language inspired by set theory is because set notation offers concise syntax when performing powerful and complex operations. By modeling the syntax of our language this way, we are able to accomplish two things: carrying out set calculations clearly and efficiently and defining operations offered by other popular languages succinctly. Our language, SetC, stands for Set Concise and we plan to compile SetC to LLVM. The following proposal details some of the specifics of our language.

2 Language Overview

The SetC language is dynamically typed; it is designed to closely mimic the natural structure of set notation to make code reading and writing more concise for programmers performing set calculations. Each program must include a main function.

2.1 Built-in Types

SetC includes four primitives: integer, float, string and boolean. Integers are whole, non-fractional numbers. Floats are floating point numbers. Strings are as they are in the standard sense; a sequence of characters enclosed in double quotes, "". Boolean is true or false. Sets can be collections of any type: any combination of primitives as well as other sets.

Primitives	Example
integers, floats	`1, 1.112`
strings	`"the setc language"`
boolean	`true, false`
Collections	**Example**
sets	`{1, 2, 3, 4}`

Table 1: Built-in Types

2.1.1 Building a Set

The following examples are multiple ways to construct a set in SetC:

```
set1={1,2,3};                    ~~ bracket notation
set2={2*i-1 | 1<=i<=5};          ~~ yields {1,3,5,7,9}
set3={(x,y) | 1<=x<=3, y=x+1};   ~~ set builder notation
```

2.2 Operators

The SetC language includes the standard relational operators, arithmetic operators and logical operators. When declaring a set, the values in the set will be enclosed in curly braces, **{}**. The set related operators cover the following operations: intersection, union, difference, and cardinality. SetC allows the testing for the existence of an element in a set, returning a boolean value. The "such that" operator **|** is used in two ways: first, when writing set notation; second, when evaluating an expression for a range of values to populate a set. The indexing and slicing operations use zero-based indexing. The "dot" operator . is used to distinguish between "for all" and "for any", as in set notation, when the variable can take on a range of values in boolean expressions. If the **.** operator is present before a variable, the truth value of an expression will evaluate to true only if the expression is true for all elements. The **in** keyword is shorthand for iterating over all elements in a set.

Operators	Operation Type
==, <, >, <=, >=	Numeric relational
+,-,*,/, %	Arithmetic
&&, ||, !	Logical

Table 2: Integer Operations

Operators	Operations	Examples
+	intersection	a+b
*	union	a*b
-	symmetric difference	a-b
#	cardinality	#a
,	chain elements together	a,b
{}	declaration	{a, b}
?	existence of an element	a?b
|	"such that", used in set notation	{i | 1<i<5}
[]	indexes a set, returns the element	a[1]
:	slices a given set	start : len
in	iterates through all elements	x in a
.	'for all', precedes a variable	0<=.x<=5

Table 3: Set Operations

2.3 Control Flow

The SetC language supports `if, elif, else` statements. The boolean expression that follows the `if` and `elif` keywords must be enclosed by parentheses and can be any boolean statement. Commas are used to separate `if, elif` and `else` statements.

2.3.1 Constructing an if/else statement

```
if (value > 3) value = value - 3, else value = value + 2;
if (1 <= i <= 5 | i == 2) print(i); ∼∼ prints 2
if (x in A | x % 2 == 0) print("even numbers in A");
```

SetC supports two looping constructs. Each of the looping constructs need to be enclosed by parentheses. The first is the standard while loop and

the second is expressed through a bounded variable, a "such that" | operator, and an optional boolean expression. If no boolean expression is entered, a boolean value of true is implicit and the | operator can be omitted.

2.3.2 Constructing a Loop

```
while (boolean_expression) print();  ~~ standard while loop
(0 <= i <= 5) print(i);              ~~ prints 12345
(0 <= i <= 5 | true) print(i);       ~~ equivalent to above
```

;	end of statement
if, elif, else	standard conditional statements
while	standard looping construct
~~	comments

Table 4: Control Flow

2.4 Built-in Functions

By default, `the to_set()` and `to_string()` functions will tokenize by white space. The `to_set()` operation has two parameters: the first parameter specifies the string to be converted and the optional second parameter specifies the token delimiter expressed as a single character enclosed in double quotes, ie. "/".

print()	Displays output	print("hello");
to_set()	Converts a string to a set	to_set(a);
to_string()	Converts a set to a string	to_string("hello");

Table 5: Built-in Functions

2.5 User Defined Functions

To declare a function, the keyword def is followed by the name of the function. The arguments for the function are declared within parentheses following the function name. Each variable for the arguments are separated by a comma. Any number of argument variables may be used provided they have been declared. The body of the function should be fully enclosed in curly braces.

Example:

```
def func(a, b){ print(a); }
func(9, 10); ~~prints 9
```

3 Sample Program

```
 1: def main() {
 2:     long_string = "onomatopoeia";
 3:     sub = "mat";
 4:
 5:     ~~searches for the substring sub in long_string
 6:     if (1 <= index <= (#long_string - #sub + 1) |
 7:     long_string[index:#sub] == sub),
 8:         print("found at", index)
 9:     else print("not found");
10:
11:     ~~ prints "The panda eats shoots and leaves."
12:     sentence = "The panda eats, shoots, and leaves.";
13:     set1 = to_string(to_set(sentence, "") - {","});
14:     print(set1);
15: }
```

References

http://setl.org/setl/
https://esolangs.org/wiki/Hyper_Set_Language