{SO}{SL}

Set Operations Simplification Language
Language Reference Manual

Garrison Grogan (gg2652) - Language Guru
Trisha Maniar (trm2144) - Tester
Ryan Chun (ec3255) - System Architect
Ryan Koning (rjk2153) - Manager

TABLE OF CONTENTS

Data Types

int

A series of digits [0-9] to be read in decimal form that can range from −2,147,483,648 to 2,147,483,647 (32-bit signed integer).

- Example: int num = 32;

char

A single character (digit, letter, etc.) enclosed in single quotes ('x').

- Example: char c = 'a';

boolean

A binary variable that has two possible values: true and false.

- Example: boolean flag = true;

void

An empty data type used for methods that have no return value.

set

A sequence of integers or characters separated by commas within braces ({x, y, z}). A set can also be a sequence of subsets which hold integers or characters.

- Example: s{} = {1, 2, 3, 4};

array

A sequence of integers or characters separated by commas within brackets ([x, y, z]).

- Example: a[] = [2, 3, 4, 5];

Operators

Basic Operators

+(plus): Binary operator defined as addition on integers

Valid types: Integer, Set

Examples: a + b, (a + b) + c

Returns: Integer if left and right hand sides are integers. If the left and right hand sides are sets, returns a set.

When sets are given, the operator returns a set whose elements are the additions of all the

elements of the left and right hand side sets. Sets and Integers cannot be mixed and will return an error. Sets of different cardinalities will also return an error if they are added together. All the elements of the sets must have the operator defined for their type. Adding two types that do not have addition defined for each other will also return an error.
Ex: a+b, a = 5, b = 3, returns 8
Ex: A + B, A={1,2}, B={3,4}, returns {4,6}
Ex: A + B, A = {{1,2},1}, B = {{3,4},7}, returns {{4,6},8}
Ex: A + B, A = {1,{1}}, B={{1},1} will return an error

-(minus): binary operator defined as subtraction on integers
Valid types: Integer, Set
Examples: a - b, (a -b) -c
When a set is given, the operator returns a set whose elements are the subtractions of all the Elements of the left and right hand side sets. Sets and Integers cannot be mixed and will return an Error. Sets of different cardinalities will also return an error if they are subtracted together. All the elements of the sets must have the operator defined for their type. Subtracting two types that do not have - defined for each other will also return an error
Ex: a-b, a = 5, b = 6, returns -1.
Ex: A - B, A={1,2}, B={3,4}, returns {-2,-2}
Ex: A - B, A = {{5,6},1}, B = {{3,4},7}, returns {{2,2},-6}
Ex: A - B, A = {1,{5,6}}, B = {7,{3,4}} will return an error

**(star):* binary operator defined as multiplication on integers as well as sets
Valid types: Integer, Set
Examples: a - b, (a -b) -c
When a set is given, the operator returns a set whose elements are the product of all the Elements of the left and right hand side sets. Sets and Integers cannot be mixed and will return an Error. Sets of different cardinalities will also return an error if they are multiplied together. All the elements of the sets must have the operator defined for their type. Multiplying two types that do not have * defined for each other will also return an error.
Ex: a*b, a = 5, b = 6, returns 30.
Ex: A * B, A={1,2}, B={3,4}, returns {3,8}
Ex: A * B, A = {{5,6},1}, B = {{3,4},7}, returns {{15,12},7}
Ex: A * B, A = {1,{5,6}}, B = {7,{3,4}} will return an error

/(divide): binary operator defined as division on integers and sets
Valid types: Integer, Set
Examples: a/b, (a /b) / c
When a set is given, the operator returns a set whose elements are the quotient of all the

Elements of the left and right hand side sets. Sets and Integers cannot be mixed and will return an Error. Sets of different cardinalities will also return an error if they are divided together. All the elements of the sets must have the operator defined for their type. Multiplying two types that do not have / defined for each other will also return an error. Integers are rounded down always.
Ex: a/b, a = 5, b = 6, returns 0
Ex: A / B, A={3,4}, B={2,2}, returns {1,2}
Ex: A / B, A = {{5,9},7}, B = {{3,4},2}, returns {{1,2},3}
Ex: A / B, A = {1,{5,6}}, B = {7,{3,4}} will return an error

%(modulus): binary operator defined as modulus on integers and sets.
Valid types: Integer, Set
Examples: a%b, (a%b) % c
When a set is given, the operator returns a set whose elements are the modulus of all the Elements of the left and right hand side sets. Sets and Integers cannot be mixed and will return an Error. Sets of different cardinalities will also return an error if they are moduloed together. All the elements of the sets must have the operator defined for their type. Taking the mod of two types that do not have / defined for each other will also return an error.
Ex: a%b, a = 23, b = 6, returns 5
Ex: A % B, A={3,4}, B={2,2}, returns {1,0}
Ex: A % B, A = {{5,9},13}, B = {{3,4},7}, returns {{2,1},6}
Ex: A % B, A = {1,{5,6}}, B = {7,{3,4}} will return an error

:u(union): binary operator defined as the union of two sets
Valid types: Set
Examples: A:uB, A:u(B:uC)
When a set is given, the operator returns a set whose elements are the union of the left and right hand side sets. If a type is given on either side that is not a set, an error is returned.
Ex: A:uB, A = {1,2}, B={1,3,4}, returns {1,2,3,4}
Ex: {{1,4,5},6,{7,8}}:u {5} = {{1,4,5},6,{7,8},5}

:n(intersection): binary operator defined as the intersection of two sets
Valid types: Set
Examples: A:nB, A:n(B:nC)
When a set is given, the operator returns a set whose elements are the intersection of the left and right hand side sets. If a type is given on either side that is not a set, an error is returned.
Ex: A:nB, A = {1,2}, B={1,3,4}, returns {1}
Ex: {{1,4,5},6,{7,8}}:u {5} = {}
Ex:{{1,2},5,6}:n{{1,2},6,7} = {{1,2},6}

:i(in): binary operator that checks if an element is in a particular set.

Valid types: Set valid on L and R hand side. Integer valid only as Left hand side

Examples: A:iB, A:i(B:uC),6:iC

The operator returns a boolean telling whether or not the LHS is in the RHS. Putting an integer as the RHS returns an error.

Ex: A:iB, A = {1,2}, B={1,3,4}, returns false

Ex 1 :i {1,3,4} returns true

Ex: {2,3} :i 6 returns an error

:c(complement): binary operator that returns the complement of a set given a universe.

Valid types: Sets only, both the L and R hand sides

Examples: A:cB, A:c(B:uC)

The operator returns a Set that is the complement of the LHS argument, given a universe of the RHS argument. If LHS contains elements not in RHS, those are not returned in the resultant set.

Ex: A:iB, A = {1,2}, B={1,3,4}, returns {3,4}

Ex 1 :c {1,3,4} returns error

Ex: {2,3} :c {1,6,7} returns {1,6,7}

Ex: {1,5,6}:c {1,2,3,4,5,6} returns {2,3,4}

|| (cardinality): A unary operator, also a delimiter. When a set is placed between the two bars, an integer Representing the cardinality of the set. Returns an error if not used with a set.

Ex: A{} = {1,2,3,4,5}; |A| returns 5.

A[] = [1,2,3,5,6]: |A| returns error. Use the standard library to get the size of an array.

Comparison Operators

In SOSL, comparison operators allow the user to compare the quantity of elements in a set on the left side of an expression to the quantity of elements in a set on the right side of an expression. Comparison operators return either true or false and are of type boolean.

> *(Greater Than):* Binary operator that returns a boolean from a left to right comparison:

Ex. {2, 3} > {4, 5} --> false; {2, 3} > {4, 5, 6} --> false; {2, 3, 4} > {4, 5} --> true

>= *(Greater Than or Equal to):* Binary operator that returns a boolean from a left to right comparison:

Ex. {2, 3} >= {4, 5} -->true; {2, 3} >= {4, 5, 6} --> false; {2, 3, 4} >= {4, 5} --> true

< *(Less Than):* Binary operator that returns a boolean from a left to right comparison:
Ex. {2, 3} < {4, 5} --> false; {2, 3} < {4, 5, 6} --> true; {2, 3, 4} < {4, 5} --> false

<= *(Less Than or Equal to):* Binary operator that returns a boolean from a left to right comparison:
Ex. {2, 3} <= {4, 5} --> true; {2, 3} <= {4, 5, 6} --> true; {2, 3, 4} <= {4, 5} --> false

==(Equal to): binary operator that returns a boolean from a left to right comparison. Only applies
To Sets compares them element wise.
Ex. {2,3} == {4,5} --> false; {4,5,6} === {4,5,6} --> true
To check equality regarding the cardinality of two sets, use the | | delimiter and ==.

Logical Operators:
!: unary operator that returns a boolean for the logical inverse of its operand.
OR: binary operator that returns a boolean for the logical or of its operands.
AND: binary operator that returns a boolean for the logical and of its operands.

Order of Operations: Mathematical expressions involving integers behave with normal PEMDAS order of operations. Set operations are evaluated left to right in following the following hierarchy: (),:u = :n,:c,:i. :i has the lowest since the left and right sides of an :i expression must be completely evaluated before :i can. Since :u and :n have equal order, they will be evaluated left to right.

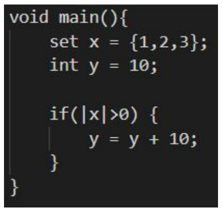
Ex: A :c C :u A :n B is equivalent to A:c((C :u A) :n B).
Ex: A :c C :n D :i B :u C is equivalent to (A: c (C :n D)) :i (B :u C)

Control Flow

Scoping

A scope is a logical region of program where name binding is valid. Inspired by C language, a SOSL program consists of several scopes defined by the nestable curly brackets. Name bindings declared outside of explicit brackets will be defined as global variables and be accessed anywhere in the program.

Example:



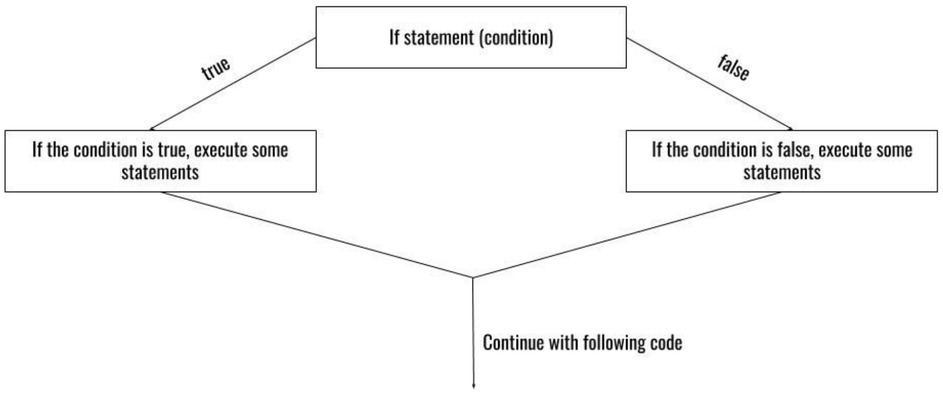
```
void main(){
    set x = {1,2,3};
    int y = 10;

    if(|x|>0) {
        y = y + 10;
    }
}
```

In the above example, two variables, x and y, are defined in the main function scope and one variable z is declared throughout the if-condition scope. Since the if-condition scope is nested in the main function scope, variable x and y can be accessible inside the if scope. But variable z is not accessible outside of the if-condition scope.

Conditional Logic

Execution of statements based on condition met


If statement (condition)
true
false
If the condition is true, execute some statements
If the condition is false, execute some statements
Continue with following code

- Example:
```
Set s = {1, 2, 3, 4};
Set t = {2, 3, 4, 5};
if (s==t)
{
        s = s + {1, 1, 1, 1};
}
if else (t = s + {1, 1, 1, 1})
{
        t = t + {1, 1, 1, 1};
}
else
{
        s = s + {2, 2, 2, 2};
}
```

- After this, s = {1, 2, 3, 4} and t = {3, 4, 5, 6} because the "if else" statement is true. Therefore, the statements within those brackets gets executed. If neither the "if" or the "if else" statement were true, then the statements within the "else" brackets would get executed.

Execution of statements for each element in set

- The forEach method works by parsing through each element in a given set
- Example:

```
Set s = {1, 2, 3, 4};
forEach (e in s) {
        e = e + 1;
}
```

- After this, s = {2, 3, 4, 5}

Execution of statements based on iterating systematically through elements

- The for method works by iterating given 3 specific statements
- The first statement will initialize a variable
- The second statement will be a condition that the variable must meet
- The third statement will modify the variable in some way (usually by incrementing or decrementing)

Jump Statements

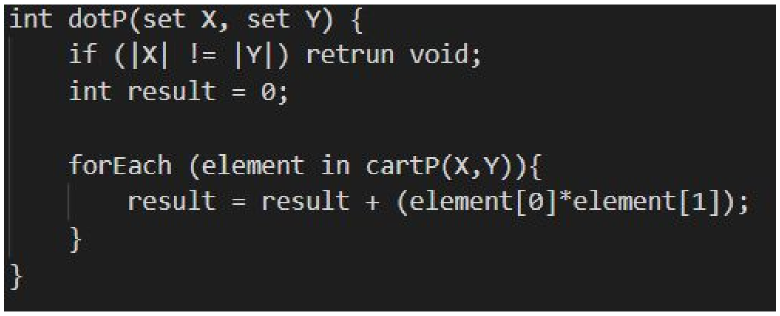
Break: A call to "break" in the middle of a "forEach" loop will terminate that loop and proceed to the next statement outside of the forEach loop.

Standard Library & Sample Program

Standard Library

The SOSL Standard Library provides commonly used set operations. It also provides functions like print. These functions can referenced as built in. Below are some example programs implemented in the standard Library.

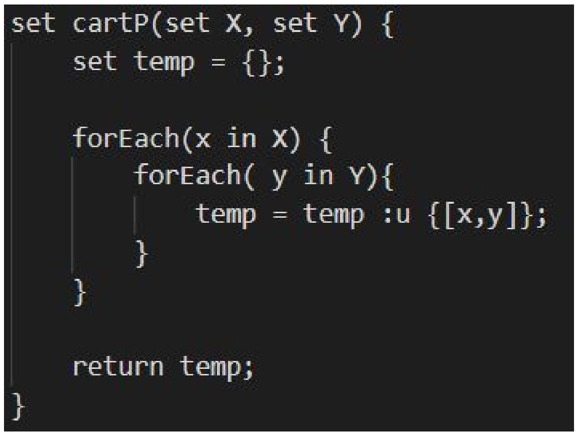
1. **dotP - Dot Product**



```
int dotP(set X, set Y) {
    if (|X| != |Y|) retrun void;
    int result = 0;

    forEach (element in cartP(X,Y)){
        result = result + (element[0]*element[1]);
    }
}
```

2. cartP - Cartesian Product

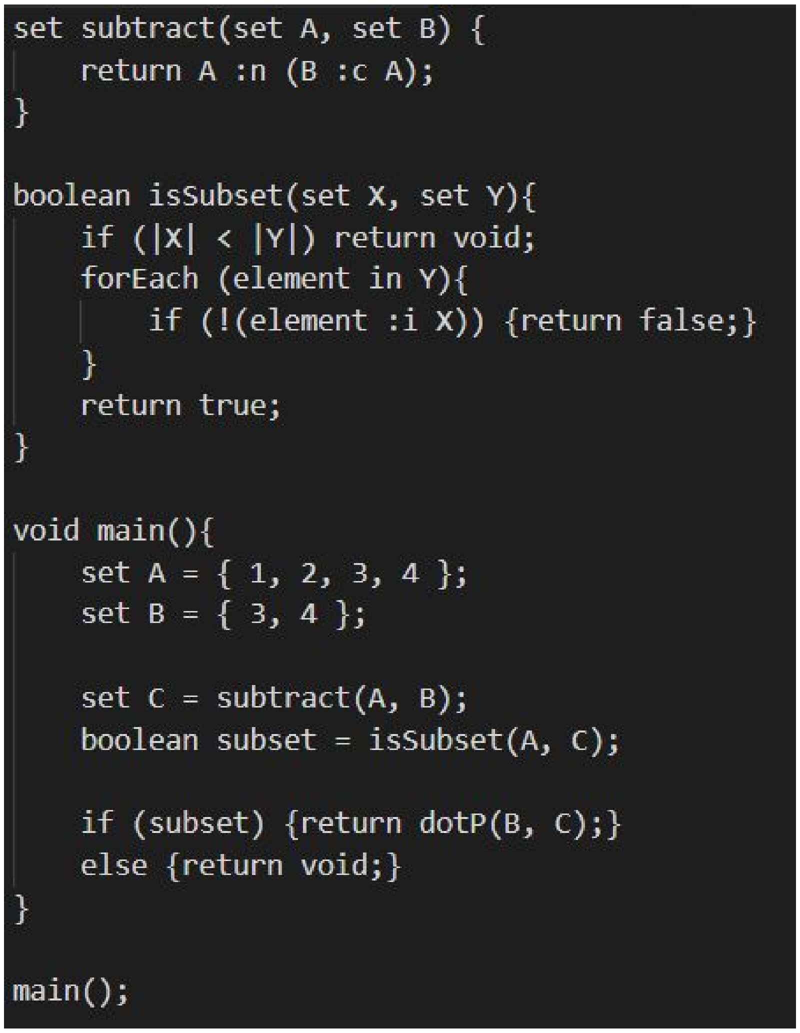


```
set cartP(set X, set Y) {
    set temp = {};

    forEach(x in X) {
        forEach( y in Y){
            temp = temp :u {[x,y]};
        }
    }

    return temp;
}
```

Sample Program



```
set subtract(set A, set B) {
    return A :n (B :c A);
}

boolean isSubset(set X, set Y){
    if (|X| < |Y|) return void;
    forEach (element in Y){
        if (!(element :i X)) {return false;}
    }
    return true;
}

void main(){
    set A = { 1, 2, 3, 4 };
    set B = { 3, 4 };

    set C = subtract(A, B);
    boolean subset = isSubset(A, C);

    if (subset) {return dotP(B, C);}
    else {return void;}
}

main();
```