SOL – Set Operation Language

Language Proposal

COMS 4115 Professor Edwards

Taylor Brown

tbb2107@columbia.edu

Introduction

Set theory is used by software engineers on a regular basis. From data structures to relational databases, sets of elements are used constantly. However, in most languages, sets are either add-ons or have to be approximated by other constructs. The goal of SOL is to provide an expressive and powerful way to describe and manipulate sets.

Overview

SOL is a functional, strongly typed language. The motivation is to be able to perform operations on sets that would otherwise be difficult or require several lines of code in a c-like language. All variables will be sets, which will consist of elements or sets. All basic set operations, as enumerated below, will be implemented. Due to time constraints, both in the project and computability, sets will be broken down into finite and infinite, as well as filtered. An infinite set will be represented by either a filter. A filter or generator may create a finite set, but the compiler will still treat it as infinite. A filter is simply a set defined as a function with the ability to accept or reject a given element as being in the set. A filter may be combined with an enumerated set as well. All data is immutable. Booleans are evaluated as sets – the empty set is false, the non-empty set is true.

Key words

filter – a filter for a set, similar to filter in other functional languages – returns set of elements qualified by the filter.

function – a function

print – built in function to pipe input to output, returns {}

Built in types

Enumerated Set – a set as defined by strings, numbers and sets

Filtered set – a set defined by a filter

Strings

Numbers – integer and decimal (not floating point)

Operators

+ union for sets, addition for numbers, concat for strings

& intersection for sets only

- difference for sets, subtraction for numbers

* Cartesian product for sets, multiplication for numbers

< as in, x < y tests if x is a subset of y, or less than for numbers

> as in, x>y tests if x is a superset of y, or greater than for numbers

!< as in, x!<y x is not a subset of y

= equality – both sets contain same elements, equal for numbers, and value equals for strings

!= disjoint set, or no common elements, not equal for numbers, not value equal for strings

// comment

{x,…} used to delineate enumerated set

{ x| filter} used to create a set based on a filter

Example code

{} // empty set

a = {1,2,3}

b = {3,4,5,6}

c = a+b // {1,2,3,4,5,6}

c = a-b // {1,2}

a = {1,2}

b = {3,4}

c = a*b // {{1,3}{1,4}{2,3}{2,4}}

```
a != b //true

a = b //false


a = {1,2,3}

b = {2,3}

a < b //false

a > b //true


function union x y:

        x+y


union a b // returns {1,2,3}


filter evens x:

        x%2 = 0


filter tens x:

        x = 10


filter over20 x:

        x > 20


filter sets x:

        {} < x // returns set of sets
```

```
filter notSets x:
    {} !< x // returns set of only non-sets


a = {x | evens x} // syntax for creating a filter on a set - this represents all evens.  it is only applied when operated
    //upon with regards to another set.


b = {1,2,3,4}

c = a&b // returns enumerable set {2,4}

c = a-b // returns {1,3}

c = a+b // returns a filter checking for evens and a "sub filter" checking for existance in {1,2,3,4} - this behaviour is determined by the operator, so all unions on a filter will return a filter


function evensOver20 x:
    {y | over20 {x | evens x}}


a = {18, 20, 21, 22}

c = evensOver20 a  // returns {22}


print a // print {18, 20, 21, 22}

function listSubelements x:
    nons = { x| notSets x}
    setSet = {x | sets x}
    a + listSubelements setSet
```