{Setʇy}: It's sets all the way down!

Outline

Outline

What is {Setɟy}?

{Setɟy} is a programming language (just like Python or C0). In most languages, you have a varied set of primitive types (int, float, string, boolean). In {Setɟy}, as you might have guessed, the only primitive type is set!

Background

Logic and Sets are two of the early topics in most introductory discrete math courses. Here's a bunch of problems students routinely run into:

- They seem very unmotivated
- It's very hard to test understanding
- It's hard to learn the language
- They often don't even realize there is a "grammar"!
- Complicated constructions (powerset, cartesian product) feel unmotivated and are hard to understand
- They don't understand the difference between predicates and functions! Or how to define them!

Introducing {Setʇy}

{Setʇy} is an attempt to fix these problems by giving students a computational environment for mathematical language.

I have a functioning {Setʇy} compiler.

There's a couple of angles I am approaching this from as research:

- Set-based languages are relatively untapped in the compilers community.
- Ideally, we'll be able to show that {Setʇy} helps students do better with some of the issues I mentioned previously.

Today

I have thought of a progression of several {Setʇy}exercises that I believe will help students with the problems they usually run into. I'd like us to go through some of the exercises and attempt/discuss them.

Getting Everyone Set up!

First, `ssh` to unix.andrew.cmu.edu and add {Setʇy} to your path.

```
ssh AndrewID@unix.andrew.cmu.edu
export PATH=/afs/cs.cmu.edu/academic/class/15151-f12/bin:$PATH
```

Now, you should be able to run {Setʇy} by using the `setty` command.

Basic Sets in $\{\text{Set}\text{ɟ}\text{y}\}$

```
# Comments in setty begin with hash symbols (like python)
# We can print sets using print and @ represents the empty set.
print @
print {@}
print {{@}}
```

```
# Setty sets come equipped with the normal set operations:
print @ union @
print {@} intersect @
print {@, {@}} minus {@}
```

Question

(a) Find two sets A and B that demonstrate that {Setɟy} sets remove duplicates.

(b) Find two sets C and D that demonstrate that {Setɟy} sets are unordered.

“Unary” Set Operations

In addition to the “normal” set operations, {Setꟷy} supports unary versions.

The idea is that we can union (or intersect) all the elements of a set to get a new set.

For instance,

$$\bigcup x, y, z = x \cup y \cup z$$

Question

Consider the following set expressions (where a is an arbitrary set):

$$\bigcup\varnothing \qquad \bigcap\varnothing \qquad \bigcup\{a\} \qquad \bigcap\{a\} \qquad \bigcup\{\varnothing,\varnothing\} \qquad \bigcap\{\varnothing,\{\varnothing\}\}$$

What do they evaluate to? Use {Setꟷy} to check your answers and understanding.

Definitions

```
# When we're doing mathematics, we will often need to define
# variables, functions, and boolean tests. This is easy in \Setty{}
empty := @
print empty

# Here's a function:
single(x) := {x}
print single({{@}})

# Here's a boolean test
has_empty(x) := @ in x
print has_empty(@)
print has_empty({@})
```

Natural Numbers

Background

Since the only objects we have in setty are sets, it would be nice if we could somehow define **numbers** in terms of sets. The big take-away is

We can make everything we need for programming from just sets!

The **Von Neumann** definition of the natural numbers as sets is the following:

- 0 is $\varnothing$
- $n+1$ is $n \cup \{n\}$

Question

{Setɟy} has been designed to let you use numbers once you've defined what the naturals are.
Define `zero`(n) and `succ`(n) using the Von Neumann definition.

(Here's a gotcha: `zero` must take an argument because {Setɟy} insists every function have exactly one argument. When implementing `zero`, just ignore the argument.)

Natural Numbers Continued

Now that you have defined numbers, let's explore them. First, {Setʇy} has a command `numerals_on` which will make it display numbers instead of sets whenever it can. It won't work if you haven't defined naturals though!

Question

Write a program for less-than x by doing the following:

```
zero(n) := <your definition>
succ(n) := <your definition>
numerals_on

x := 10
ltx(y) := <define this>
print ltx(7)
print ltx(100)
```

Pairs

Wouldn't it be great if we could give our functions multiple arguments?
Let's define what the ordered pair, (x,y) means as a set.
The Kuratowski definition is $(x,y) := \{\{x\},\{x,y\}\}$.
In addition to the "pairing" function, we need to define two more:

- $\pi_1((x,y)) = x$, and
- $\pi_2((x,y)) = y$

Question

(a) Fill in the following code. Don't worry too much about pi2.

```
(x, y) := {{x}, {x,y}}
pi1(p) := <fill this in>
pi2(p) := union {x in (union p) |
                   (x not in {pi1(p)}) or
                   forall (y in (union p)). y in {pi1(p)}}
pi2bad(p) := (union p) minus {pi1(p)}
```

(b) pi2bad doesn't actually work! Which pairs does it fail on?

Logical Statements

{Setɿy} comes with `and`, `or`, and `not` built in.

Question

Write a logical test `implies`$((x, y))$ and test it with:

```
print T implies T
print T implies F
print F implies T
print F implies F
```

{Setɿy} also supports $\forall$ and $\exists$. For instance,

```
print forall (x in [5]). x in 6
print exists (x in [5]). x in 3
```

Question

Define a predicate `isprime`(x) which tests if x is prime.

(You could, in theory, implement $+$ and $\times$, but {Setɿy} also has them built in.)