Properties and Sets

Here are two statements:

* Jane has red hair
* Jane is in the set of redheads

What is the difference in meaning of these two statements? Not a lot, really. The first talks about a person having some attribute or property. The second talks about being a member of a set. But the set is defined as all people with that property. Therefore, being in the set is equivalent to having that attribute.

Here are some more statements:

2a: 13 is prime
2b: 13 is an element of the set of primes

3a: 01110011 ends in 011
3b: 01110011 is in the set of binary strings that end in 011

4a: M is a Turing machine that determines if a binary string ends in 011
4b: M is an element of the set of Turing machines that determine if a binary string ends in 011
4c: M is an element of the set of Turing machines that decide if a binary string is an element of the set of binary strings that end in 011.

5a: Turing machine M accepts string 00011
5b: The pair (M, 00011) belongs to the set of pairs of machine and string: (M, w) where the machine M accepts the string w

As these examples show, we can rephrase any question about having a property (having red hair, ending in 011, being able to determine if a string ends in 011,...) to an equivalent question about something being an element of a set.

That fact is so important, we state it again:

**FACT:** Having a property is equivalent to being in a set.

**Is Sets Necessary?**

Why make this translation? Do we have to? Set theory is a powerful mathematical field. By moving questions about properties into questions about sets, we can use ideas, theorems, and definitions of set theory to solve problems. Furthermore, using a single model (set inclusion) to talk about various problems, we can see similarities more easily.

**Is You Is or Is You Ain't?**

Consider this C++ function:

```cpp
bool is_prime(unsigned int n) {
    for(int i=2; i<n; i++)
        if ( (n % i) == 0 )
            return false;
    return true;
}
```

This function determines if a given non-negative integer value is prime. OR This function determines if a given non-negative integer value belongs to the set of prime numbers. We can represent this question with a simple set diagram:

![Set Diagram](image)

Pay close attention to the main items in this diagram. The large box is the set of all natural numbers. This set is infinite. The smaller box inside the set of natural numbers represents the set of prime numbers. We shall call this set PRIMES. The diagram shows a few elements: 12, 17, 204, 67. Some are in PRIMES, some are not in PRIMES.

The function shown above, *is_prime(n)*, determines if any given number is a member of PRIMES. For any natural number we give the function, the function loops a finite number of times and then returns true or false. We say "*is_prime()* decides **PRIMES**".

One final vocabulary word. We can call the set of prime numbers a **language**. A language is a subset of some larger set. A function that tells you if an element is in a language (i.e. that subset) or not in a language (i.e. not in the subset) is called a **decider** for that language.

Therefore we can say: "*is_prime(n)* is a decider for the language **PRIMES**".

**Properties, Sets, Deciders**

We can translate the question of testing if something has a property into the question of finding a decider for a language. And we can use this general model to ask about all kinds of questions of solutions to problems.
Enumerators

A decider for a language $L$ answers the question: "Is $w$ an element of a language $L$?" The `is_prime()` function decides the language PRIMES.

Here is a different question: "What are all the elements of $L$?". The following code lists all primes:

```cpp
bool list_all_primes(unsigned int n) {
    int i = 3;
    for( int i = 3; ; i++ )
        if ( is_prime(i) )
            cout << i << endl;
}
```

Listing the elements of a language is called enumerating the language. A function (machine) that lists a language $L$ is called an enumerator for $L$.

Note how this enumerator works. Loop through all elements of the larger set and call the decider as a subroutine. If the decider "accepts" the value, the enumerator prints that value. Here, the set of positive integers is infinite, so the enumerator never ends, but each element in the language will be printed sooner or later.

Example 2: Deciding No-Zeros

Now, consider the set of all strings of 0s and 1s. That set has a subset consisting of all strings that have no zeros. Call this subset NO-0S. Here is NO-0S within $\{0,1\}^*$. NO-0S is a language. Can you write a Turing machine that decides this language? Here is one:

```cpp
M_{NZ}:
{ 
    $\Sigma$: \{0,1\}
    $\Gamma$: \{0,1,B\}
    Q: $q_0$, $q_A$, $q_R$
    Start: $q_0$
    $\delta$:
        $<q_0,0> \rightarrow <q_R, 0, \text{Halt}>$
        $<q_0,1> \rightarrow <q_0, 1, R>,$
        $<q_0,B> \rightarrow <q_A, B, \text{Halt}>$
}
```

Could you write a different Turing machine that decides NO-0S? Try it now. How many machines are there that decide NO-0S? (hint: a lot) That means there is a set of deciders for NO-0S.

Can we write an enumerator for NO-0S to print all elements of NO-0S? Output of a Turing machine is the string on the tape when the machine halts, so an enumerator has to write the list of elements on the tape. Output could be:

```
#1#1#11#1111#11111# ...
```

This machine uses the special tape symbol # as a ‘comma’ to separate each value from the next.

Sets of Machines, Languages of Machines

The machine $M_{NZ}$ defined above decides the language NO-0S. Just as you can write different C++ programs that do the same thing, you can write different Turing machines to decide NO-0S. That means we can talk about the set of all Turing machines that decide NO-0S. Call this language DECIDE-NZ. DECIDE-NZ is a subset of the set of all Turing machines. Remember, ‘language’ is just a new word for subset of a larger set.

This diagram contains a core idea for this course. The box at right shows the language NO-0S as a subset of $\{0,1\}^*$. The set, DECIDE-NZ, of all the Turing machines that decide NO-0S, shown at left, is a subset of the set of all Turing machines.

The little gray squares are individual Turing machines, just as 011, 000, 11, 111 are individual strings. Each little square represents an entire Turing machine, some of which determine if any binary string has no zeros.

IMPORTANT QUESTION: Someone gives you a description of a machine (like that given for $M_{NZ}$), can you determine if that machine is in DECIDE-NZ?

EVEN BIGGER QUESTION: Can we write a machine to decide DECIDE-NZ? Input to this machine is a description of a machine. This new machine accepts if its input decides NO-0S and rejects if its input does not decide NO-0S.

And here is the even weirder question: where in the diagram do you put this new machine? What if you wrote the $M_{NZ}$ description as a single binary string by translating each of its chars to ASCII using base 2? Where in the diagram would that string go? Think about this carefully.