Section 9: Compiling Constraints

In this section, we will practice compiling 401_{CP} programs into SAT.

401_{CP} is an extension of our 401 language. It has the following additional constructs which allow us to express program constraints:

- `assert(E);` // Add a constraint stating that E is true
- `choose();` // Resolves to a value that satisfies all program constraints

SAT is our language for expressing constraints. It has the following syntax:

- `(a Int)` // Variable declaration. Supported types are Int and Bool
- `(a + b)` // Arithmetic operators (+, -, * …)
- `(a = b)` // Equivalence operator. Not assignment!
- `(a=b ∨ ¬(a=10))` // Logical connectives (∨, ∧, ¬)
- `ct(a=b)` // Defines constraint that a must be equal to b
- `ite(a>b,a=1,a=2)` // Syntactic sugar for (a > b ∧ a=1) ∨ (¬(a > b) ∧ a=2)

Exercise 1: Polynomials

a) Using the translation rules from the last lecture, convert the following 401_{CP} code for solving the polynomial $\mathbf{x^2 + 2y + 3}$ to SAT.

```
def x;
def y;
def poly;
x = choose();
y = choose();
poly = x * x + 2 * y + 3;
assert(poly == 0);
```

Answer:

b) Here is another version of 401_{CP} code computing the same polynomial:

```
def x;
def y;
def poly;
X = choose();
y = choose();
poly = x * x;
poly = poly + 2 * y;
poly = poly + 3;
assert(poly == 0);
```

Explain briefly, why this code needs to be re-written before it can be translated to SAT?

Re-write the code to eliminate the problem:

Exercise 2: Sorting

a) The following 401_{CP} code uses bubble sorting algorithm to sort the array **arr**.

```
lambda bubbleSort(array){
    // range(0,3) iterates 0 <= index < 3
    for(index in range(0,3) {
        if(arr[index] > arr[index + 1]){
            def temp = arr[index];
            arr[index] = arr[index + 1];
            arr[index + 1] = temp;
        }
```

```
    }
}

def arr = {};
arr[0] = 10;
arr[1] = 20;
arr[2] = 15;
arr[3] = 5;

bubbleSort(arr);
```

Re-write the above code such that it may be translated to SAT. *Hint: Generate a new variable for each index of the array.*

b) In this example, the length of **arr** is static. Explain briefly, how would you handle the case where the length of the array was dynamic / unknown?

c) Translate the rewritten code from part (a) to SAT.

Exercise 3: Map Coloring

In this exercise we want to find out a way to color the map in such a way that any two countries on the map that share a border have a different color. A country can only be colored either red, green or blue. The following 401_{CP} code defines the constraints.

```
// 5 Countries on the map
def countries = {0=A, 1=B, 2=C, 3=D, 4=E}

// Dictionary mapping countries to the list of neighbours
def neighbours = {}
neighbours[A] = {0=B, 1=D}
neighbours[B] = {0=A, 1=D, 2=E}
neighbours[C] = {0=E}
neighbours[D] = {0=A, 1=B, 2=E}
neighbours[E] = {0=B, 1=C, 2=D}

// Colors of each country. Some unknown
def colors = {}
colors[A] = Red
colors[B] = Blue
colors[C] = Choose()
colors[D] = Choose()
colors[E] = Choose()

// Define constraints
for(country in countries){
    for(neighbour in neighbours[country]){
        assert(colors[country] != colors[neighbour]);
    }
}
```

Translate the code to SAT.