DCC888 - Static Analysis of Programs

Name: ____ **ID**: ________________

By turning this exam in, I give my word that I have done it alone, on the understanding that I am allowed to consult any material publicly available, except material disclosed by other colleagues who are taking this course, or who have taken it in the past.

This exam concerns a simple reactive language, that uses the following grammar:

```
P :: = S; P
     | S
S ::= V = E
     | bind(V)(C){P}
     | bind(V, I)(C){P}
E ::= V ⊕ V, ⊕ in {+, -, *, /, %}
     | V[V]
     | c, c is numeric or boolean const
     | IN(T), T in {Int, Float, Bool}
     | OUT(E)
C ::= V ⊕ E, ⊕ in {<, <=, >, >=, !=, ==}
```

Examples of programs are given below:

Program 1: Prints the range [0, x]

```
x = IN(int)
bind(x)(true){b = 1}
bind(b)(b < x){OUT(b), b = b+1}
```

Program 2: multiply a float for an integer using additions

```
x = IN(float)
y = IN(float)
bind(x)(true){b = 0}
bind(b)(b < x){y = y + x; b = b + 1.0}
bind(y)(b == x){OUT[y]}
```

Program 3: prints the contents of a vector w, in parallel. Any ordering is possible

```
x = IN(int)
bind(x)(x >= 0){
  x = x - 1;
  bind(v, x)(true){OUT(w[x])}
  v[x] = true;
}
```

Program 4: prints all the positive or negative numbers

```
x = IN(bool)
bind(b)(x == true){b = b + 1; OUT(b);}
bind(b)(x == false){b = b - 1; OUT(b);}
```

The construction bind(V)(C){P} sets an event. Whenever the location V (be it a variable or a position in an array) is assigned, the condition C is verified. If the condition is true, then the program P executes. Notice that the execution environment is totally parallel: execution ensues as soon as the event happens, and every IN operation that is not guarded by an event happens in parallel. However, commands inside the body of a bind statement happen sequentially. The only difference between bind(V)(C){P} and bind(V, I)(C){P} is that the latter waits for assignments at cell I of an array V. The other constructions in the language are standard expressions: binary operations, memory access, constants, input, output and boolean comparisons. Assume that every variable is initialized with zero.

Question 1 (10 Points): if two variables of different types are combined in a binary operation, then we have a type error. Design a static analysis to find out if a program is type safe. See the two examples below:

Program 5: this program contains a type error

```
x = IN(int)
y = IN(float)
bind(x)(true){b = x + y}
```

Program 6: this program does not contain the type error:

```
x = IN(float)
y = IN(float)
bind(x)(true){b = x + y}
```

Question 2 (10 Points): if a program contains two accesses to the same location, and at least one of these accesses is a write operation, then we might have a race condition. Write a static analysis to find out if a program has a race condition. Examples:

Program 7: this program contains a race condition

```
x = IN(int)
x = IN(float)
```

Program 8: another data race:

```
x = IN(float)
bind(x)(true){b = 0; c = 0;}
bind(b)(true){d = 0;}
bind(c)(true){d = 0;}
```

Program 9: no data race

```
x = IN(float)
bind(x)(true){b = 0; c = 0;}
bind(b)(true){OUT(x);}
bind(c)(true){OUT(x);}
```

Only one Extra question will be considered. You can choose which one you want to be graded.

Extra A (1 Point): our language allows programs that will never terminate. An example is Program 4, above. Write a static analysis to find out programs that ***always*** terminate: if your analysis says that a program terminates, it must terminate for any input. Notice that this problem, in general, is undecidable. Therefore, your analysis must be conservative (but not trivial).

Extra B (1 Point): imagine that we have type converters, e.g., a = float(b), a = int(b), a = bool(b). Design an optimization that changes a program that is not type safe into a program that is type safe.

Extra C (1 Point): implement a grammar for our language in any programming language, and implement at least one of these static analysis using this infrastructure.