Homework 4
CSE6049 Program Analysis, Spring 2021

Woosuk Lee

due: 6/9(Wed), email-to-TA (bbumbuul@yahoo.com)

- The goal of this assignment is to implement a sound static analyzer based on a transitional semantics. In particular, the main goal is to implement the worklist algorithm.
- Skeleton code is provided and accessible through the course website. Before you start, see `README.md` to understand how to proceed.
- Please send a ZIP file titled "HW4_[Your Student ID].zip" to TA via email, and the zipped file should contain OCaml source files `interpreter.ml` and `domains.ml`.

Consider the extended version of the `miniC` language used in the previous assignment. The language is almost identical to the previous language except the goto statement whose target label is not fixed in the program text but to be computed during execution by its argument expression. The label statement is used for specifying potential jumped-to locations. The syntax is depicted in Figure 1.

Given a program, each statement is assigned a unique label as an integer. The language is represented as the following data type in OCaml.

```
type label = int
type var = string
type stmt = cmd * label

and cmd =
  | ...
  | GOTO of exp
  | LABEL of label
```

n	$\in$	$\mathbb{V}$	scalar values
$\mathtt{x}$	$\in$	$\mathbb{X}$	program variables
$\odot$	::=	$+ \mid - \mid *$	binary operators
$\oslash$	::=	$< \mid \leq \mid == \mid > \mid \neq$	comparison operators
E	::=		scalar expressions
	$\mid$	n	scalar constant
	$\mid$	$\mathtt{x}$	variable
	$\mid$	$E \odot E$	binary operation
B	::=		boolean expressions
	$\mid$	$\mathtt{x} \oslash n$	comparison of a variable with a constant
	$\mid$	$\neg\, B$	negated condition
C	::=		commands
	$\mid$	$\cdots$	... same as before ...
	$\mid$	$\mathtt{goto}\ E$	goto with dynamically computed label
	$\mid$	$\mathtt{label}\ n$	user-specified label

Figure 1: Grammar of the extended `miniC` language

...

In this setting, our goal is to build a static analyzer for obtaining label-wise abstraction; we want to obtain a table from each program label to an abstract memory

$$\mathbb{L} \to \mathbb{M}^\#$$

where $\mathbb{M}^\# = \mathbb{X} \to D_I$ is the space of abstract memories based on the intervals abstract domain D_I. The `Table` module in file `domains.ml`

```
module Table =
struct
  include LabelMap (* implemented in c.ml *)
  let string_of_t t = ...
end
```

implements the space $\mathbb{L} \to \mathbb{M}^\#$ and the module `IntervalMem` implements the space of interval abstract memories.

Exercise 1 Implement the transitional-style abstract interpreter based on the intervals abstraction.

In particular, you first implement the function `abs_trans` in file `interpreter.ml` which implements the abstract state transition relation $\hookrightarrow^\#$ defined at page 45 of the lecture slide at `http://psl.hanyang.ac.kr/~wslee/courses/cse6049/lecture8.pdf`. You can use the following helper functions already implemented and provided to the `abs_trans` function as arguments.

-
```
next:  label -> label
nextTrue:  label -> label
nextFalse:  label -> label
```

These functions will give a next label to be executed for a given label (if no successor label exists for a given label, the exception `NoSuccessor` will be raised). The functions determine the execution order between the statements except for the goto statement.

- `cmdOf : label -> cmd`

  This function gives a command associated with a given program label.

- `all_labels : label list`

  list of all program labels of a given input program.

Then, using the abstract state transition relation, implement `analyze` function in `interpreter.ml`. Implement the worklist algorithms with widening/-narrowing at pages 83 – 84 in the lecture slides.

You can use the option `-pp` to check how labels are assigned to statements of a program. For example, for a program `test1.c` in the `test` folder in the skeleton code

```
read(x);
x := x + x;
y := x * 2
```

if you type the following

```
$ ./run -pp test/test9.c
```

you will see the following result showing how each individual statement is assigned a label.

```
[5: [4: [2: [0: read x]; [1: x := x + x]]; [3: y := x * 2]]; [-1: skip]]
```