HW4 – Interpreters and Exceptions

CS 476, Fall 2018
Due Oct. 5 at 2 PM

1 Instructions

Begin by downloading the file `hw4-base.ml` from the course website, and renaming it to `hw4.ml`. This file contains the functions that you will use and modify in the homework. You should not need to define any new functions, or change any functions other than `step_cmd`. Submit your completed `hw4.ml` via Gradescope. As always, please don't hesitate to ask for help on Piazza (`https://piazza.com/class/jkh8q52qrh06v`).

2 Extending the Step Function with Exceptions

The file `hw4-base.ml` defines the types `exp` of expressions and `cmd` of commands. It also defines two main functions: `eval_exp`, a big-step-style interpreter for expressions, and `step_cmd`, a small-step-style interpreter for commands.

The function `eval_exp : exp -> state -> exp_res option` takes an expression e and a state s and returns either:

- `Some (Val` v`)`, if e in the state s evaluates to the value v, that is, $(e, s) \Downarrow v$
- `Some Exc`, if evaluating e in the state s produces an exception, that is, $(e, s) \Downarrow \text{exc}$, which will happen when e involves dividing by zero
- `None`, if evaluating e in the state s produces an error, such as trying to add `bools` or evaluate an undefined variable

The function `step_cmd : cmd -> state -> (cmd * state) option` takes a command c and a state s and returns either:

- `Some (`c'`,` s'`)`, if $(c, s) \to (c', s')$
- `None`, if there is no step that (c, s) can take

`eval_exp` has already been updated to produce and handle exceptions, and the `cmd` type already includes constructors `Throw`, for the throw command, and `Try of cmd * cmd`, for the try-catch command.

(15 points) Extend the `step_cmd` so that it handles exceptions and the try-catch command. Use the small-step rules for exceptions and the try-catch command, shown below. For instance,

```
step_cmd (Seq (Assign ("x", Div (Int 1, Int 0)), Skip)) empty_state
```

should return `Some (Seq (Throw, Skip), empty_state)`, and

```
step_cmd (Try (Throw, Assign ("x", Int 2))) empty_state
```

should return `Some (Assign ("x", Int 2), empty_state)`. (Note that OCaml will display `empty_state` (and any other state) as `<fun>` in the output.)

You should not need to change any existing cases of `step_cmd`, but you may need to add cases to both the top-level match statement and the inner match statements. The throw command itself, like the skip command, does not step to anything.

$$\frac{(e, \sigma) \Downarrow \text{exc}}{(x := e, \sigma) \rightarrow (\text{throw}, \sigma)} \qquad \frac{}{(\text{throw}; c_2, \sigma) \rightarrow (\text{throw}, \sigma)}$$

$$\frac{(e, \sigma) \Downarrow \text{exc}}{(\text{if } e \text{ then } c_1 \text{ else } c_2, \sigma) \rightarrow (\text{throw}, \sigma)}$$

$$\frac{(c_1, \sigma) \rightarrow (c_1', \sigma')}{(\text{try } c_1 \text{ catch } c_2, \sigma) \rightarrow (\text{try } c_1' \text{ catch } c_2, \sigma')}$$

$$\frac{}{(\text{try skip catch } c_2, \sigma) \rightarrow (\text{skip}, \sigma)} \qquad \frac{}{(\text{try throw catch } c_2, \sigma) \rightarrow (c_2, \sigma)}$$