CS 381, Spring 2025 Homework 3 (Semantics)

Important Reminders!

1. Upload your solution as a single Elm file (ending in `.elm`) to **Canvas**. (*Note*: No pdf; don't use Pensieve.)
2. **Only submit files that compile without errors!** (Put all non-working parts in comments.)
3. You must do all homework assignments by yourself, without the help of others. Also, you must not use services such as Chegg or Course Hero. If you need help, simply ask on Canvas, and we will help!
4. You can work in teams of ***up to four*** students to create and submit a common homework solution. To this end, first create an alphabetically sorted (by last name) list of the team members, and add this information at the beginning of the submitted Elm file, like so.

```
{- GROUP:
   Helena Eagen
   Dylan George
   Mark Scout
-}
```

The other group members **must not** submit a solution. All group members will receive the same grade.

Exercise 1. A Stack Language

Consider the stack language defined by the following grammar.

$$\begin{aligned} \textit{prog} &::= \textit{op} \mid \textit{op}\ \texttt{;}\ \textit{prog} \\ \textit{op} &::= \texttt{LD}\ \textit{num} \mid \texttt{ADD} \mid \texttt{MULT} \mid \texttt{DUP} \end{aligned}$$

A stack program essentially consists of a (non-empty) sequence of operations, as given by the nonterminal *op*. The meaning of a stack program is to start with an empty stack and to perform its first operation on it, which results in a new stack to which the next operation in is then applied, and so on. The stack that results from the application of the last operation is the result of the program.

The operation `LD` loads its integer parameter onto the stack. The operation `ADD` removes the two topmost integers from the stack and puts their sum onto the stack. If the stack contains fewer than two elements, `ADD` produces an error. Similarly, the operation `MULT` takes the two topmost integers from the stack and puts their product on top of the stack. It also produces an error if the stack contains fewer than two elements. Finally, the operation `DUP` places a second copy of the stack's topmost element on the stack. (You can find out the error condition for `DUP` yourself.) Here is a definition of the abstract syntax that you should use.

```
type Op = LD Int | ADD | MULT | DUP
type alias Prog = List Op
```

Integer stacks should be represented by the type `List Int`, that is, your program should contain and use the following definition.

```
type alias Stack = List Int
```

Define the semantics for the stack language as an Elm function `semProg` that yields the semantics of a stack program. Note that the semantic domain has to be defined as a function domain (since the meaning of a stack program is a transformation of stacks) *and* as an error domain (since operations can fail). Therefore, `semProg` has the following type where you have to find an appropriate type definition for `D`.

```
type (alias) D = ...

semProg : Prog -> D
```

As support for the definition of `semProg` you should define an auxiliary function `semOp` for the semantics of individual operations, which has the following type.

```
semOp : Op -> D
```

Hint. Test your definitions with the stack programs `[LD 3,DUP,ADD,DUP,MULT]` and `[LD 3,ADD]` and the empty stack `[]` as inputs.

Exercise 2. Mini Logo

Consider the simplified version of Mini Logo (without macros), defined by the following abstract syntax.

```
type alias Point = (Int,Int)
type Mode = Up | Down

type Cmd = Pen Mode
         | MoveTo Point
         | Seq Cmd Cmd
```

The semantics of a Mini Logo program is a set of drawn lines. However, for the definition of the semantics a "drawing state" must be maintained that keeps track of the current position of the pen and the pen's status (`Up` or `Down`). This state should be represented by values of the following type.

```
type alias State = (Mode,Point)
```

The semantic domain representing a set of drawn lines is represented by the type `Lines`.

```
type alias Line = (Point,Point)
type alias Lines = List Line
```

Define the semantics of Mini Logo via two Elm functions. First, define a function `semCmd` that has the following type.

```
semCmd : Cmd -> State -> (State,Lines)
```

This function defines for each `Cmd` how it modifies the current drawing state and what lines it produces. After that define the function `lines` with the following type.

```
lines : Cmd -> Lines
```

The function `lines` should call `semCmd`. The initial state is defined to have the pen up and the current drawing position at $(0,0)$.

A Note on Testing Your Mini Logo Function Definitions

You can test your Mini Logo semantics as follows.

(1) If you haven't done already, initialize Elm in your current directory with the command `elm init` to ensure the presence of a proper `elm.json` file and the subdirectory `src` that contains your homework Elm files.
(2) Install the Elm SVG package with the following shell command `elm install elm/svg`.
(3) Download the file with the name `HW3_MiniLogoTest.elm` from Canvas into the `src` subdirectory. It looks as follows.

```
module HW3_MiniLogoTest exposing (..)

...

----- BEGIN HW3 solution

type alias Point = (Int,Int)

...

semCmd : Cmd -> State -> (State,Lines)

lines : Cmd -> Lines

logoResult : Lines
logoResult = lines (Seq (Seq (Seq (Pen Up) ...
```

(4) Insert your function definitions after the `BEGIN HW3 solution` comment.

(5) In the current directory (that is, in the `src` subdirectory), execute the command `elm reactor`.

(6) In your web browser, enter the URL `http://localhost:8000`. This will allow you to load the file `HW3_MiniLogoTest.elm`, which will then render the `Lines` value `logoResult` in your browser.

(7) **IMPORTANT!** ***Do NOT submit this file to Canvas. Submit only the Elm file that runs in the REPL.***