Exercise 9: Relational Programming

Starter code

Download these files from Quercus files.

- `Exercises/ex9/mk.rkt`
- `Exercises/ex9/number.rkt`
- `Exercises/ex9/evalo.rkt`
- `Exercises/ex9/ex9.rkt`

Submission instructions

Submit the file `ex9.rkt` to MarkUs: https://mcsmark.utm.utoronto.ca/csc324f20/

You may not change `mk.rkt`, `number.rkt` and `evalo.rkt`. You won't be able to submit those files since we'll use our own for testing purposes.

Task 1. The relation reverseo [3 pt]

Write the relation `reverseo`, the relational form of the function `reverse` that reverses the elements of a list.

Remember that miniKanren relations cannot use Racket functions on terms that may contain logic variables. Your implementation of `reverseo` should only use `fresh`, `conde`, `==`, `=/=`, calls to other relations, and constructors like `cons` and `list`.

You may optionally use the `appendo` relation from lecture. (There is a version of `appendo` imported from `number.rkt`.)

Task 2. Fixing changeo [3 pt]

The version of `changeo` implementation in the starter code is problematic: it produces repeated results. Modify the definition of the relation, so that each result is only produced *once* in a `run*` query.

You may find our discussion on the implemenation of `lookupo` helpful.

Task 3. Programming by Example [4 pt]

In this exercise, we will use the relational interpreter `evalo` from the lectures and readings to solve **programming by example** problems.

In a programming by example problem, we are given input-output pairs to a function. For example:

Input	Output
'x	'x
'y	'y

A function that is consistent with these input-output pairs is:

```
(lambda (arg) arg)
```

Write a macro (or function) `pbe` that fills in a function definition:

```
(lambda (arg) ___________)
```

...so that it is consistent with the provided input output example pairs. For the example above, we should be able to get:

```
> (pbe ('x 'x) ('y 'y))
```

```
'(arg) ; since (lambda (arg) arg) is an answer

; More examples:
> (pbe ('x 'x) ('y 'x))
'('x) ; since (lambda (arg) 'x) is an answer
> (pbe ('x '(x)) ('y '(x)))
'((list arg)) ; since (lambda (arg) (list arg)) is an answer
```

Use the relational interpreter `evalo`, which has been imported for you.

Note that `pbe` is **not a relation**. We recommend writing `pbe` as a macro, but you can optionally write `pbe` as a function.

Although `pbe` is really fun to play with, you can expect the search to be really, really slow. You should be able to synthesize very simple function (e.g. repeating an element a few times), but not functions that are much more complicated. The search space grows exponentially as your desired function becomes more and more complex.