Theory Tasks - Module 1, Satisfactory (S)

You should submit your answers as `pdf`, `jpg`, or `png` files.

Regular expressions

Give an example of a regular expression that accepts the following strings:

- `aabbcd`
- `aaaaaacd`
- `abbbbcd`
- `abad`
- `ad`
- `acd`

but it must not accept any of the following

- `bcd`
- `accd`
- `a`
- `cd`

You are only allowed to use the regular expression primitives: symbols, alternations, concatenations, epsilon, and Kleene closure. You are not allowed to use abbreviations, such as ? or +. Try to create a regex that is as short as possible.

Grammar

Given the following EBNF grammar

```
expr -> "(" [ expr { "," expr } ] ")"

expr -> Num
```

write down the corresponding set of productions using ordinary BNF (without the `{ }` and `[ ]` constructs).

Parse trees and abstract syntax trees

Consider the following grammar with start symbol `stmtList`

```
expr -> "(" expr ")"

expr -> expr "-" expr

expr -> expr "/" expr

expr -> Num

expr -> Ident
```

```
stmt -> "if" expr "then" stmt

stmt -> "{" stmtList "}"

stmt -> Ident "=" expr ";"

stmtList ->

stmtList -> stmtList stmt
```

where `Num` represents constant unsigned integers and `Ident` identfiers.

For each of the following sub tasks, write down a solution:

1. Draw clearly a possible *parse tree* for the following program

   ```
   if x then { y = 1 - 2 / z; z = y; }
   ```
2. Is the grammar ambiguous or unambiguous? If the former, how many possible parse trees are there for the example program above? Which are the differences between the parse trees?
3. Give at least three examples of terminal symbols that are part of the parse tree, but would not be part of an abstract syntax tree.

Well-typed terms and type checking

For each of the following terms, state whether the term is well-typed. If it is well-typed, state the type of the term. If it is not well-typed, explain where the type error is located. Clearly motivate your answers.

Note that the addition operator `+:Int->Int->Int` is given in infix form and that we use the keyword `lam` to represent a lambda.

1. Term `(lam x:Int. x + 1) (lam y:Int. y + 1)` within the empty environment.
2. Term `(lam x:Bool. lam y:Int. y + z) true` within the environment `[z:Int, w:Bool]`
3. Term `(lam f:Int->Bool. lam x:Int. f (x + 1)) g 7` within the environment `[f:Int->Bool, x:Bool, g:Bool->Int]`

Static vs. dynamic type checking

Discuss shortly (max one page) the main differences between static type checking and checking types dynamically. Which are the pros and cons? Give examples of languages within both categories. What do you prefer and why?