UNIVERSITY OF Southampton

School of Electronics and Computer Science

COMP2212
PROGRAMMING LANGUAGE CONCEPTS

Dr Julian Rathke

LEXING IN HASKELL

THE ALEX TOOL

- The Alex tool is a code generation tool for automatically generating lexers in Haskell.
- The user provides an Alex specification
  - i.e. a list of lexemes and a tokenisation action for each lexeme
- Alex generates a Haskell function named `alexScan` that does the job of a scanner but also identifies tokens and actions to be taken
- Alex is parametrisable in the way it scans and evaluates - in order to customise it you can provide implementations for the following
  - `type AlexInput`
  - `alexGetByte :: AlexInput → Maybe (Word8, AlexInput)`
  - `alexInputPrevChar :: AlexInput → Char`
- And provide an evaluator function `alexScanTokens` that does the evaluation
- This may seem a little complicated but fortunately there is a basic "wrapper" that provides default implementations of these for off-the-shelf use

THE BASIC WRAPPER

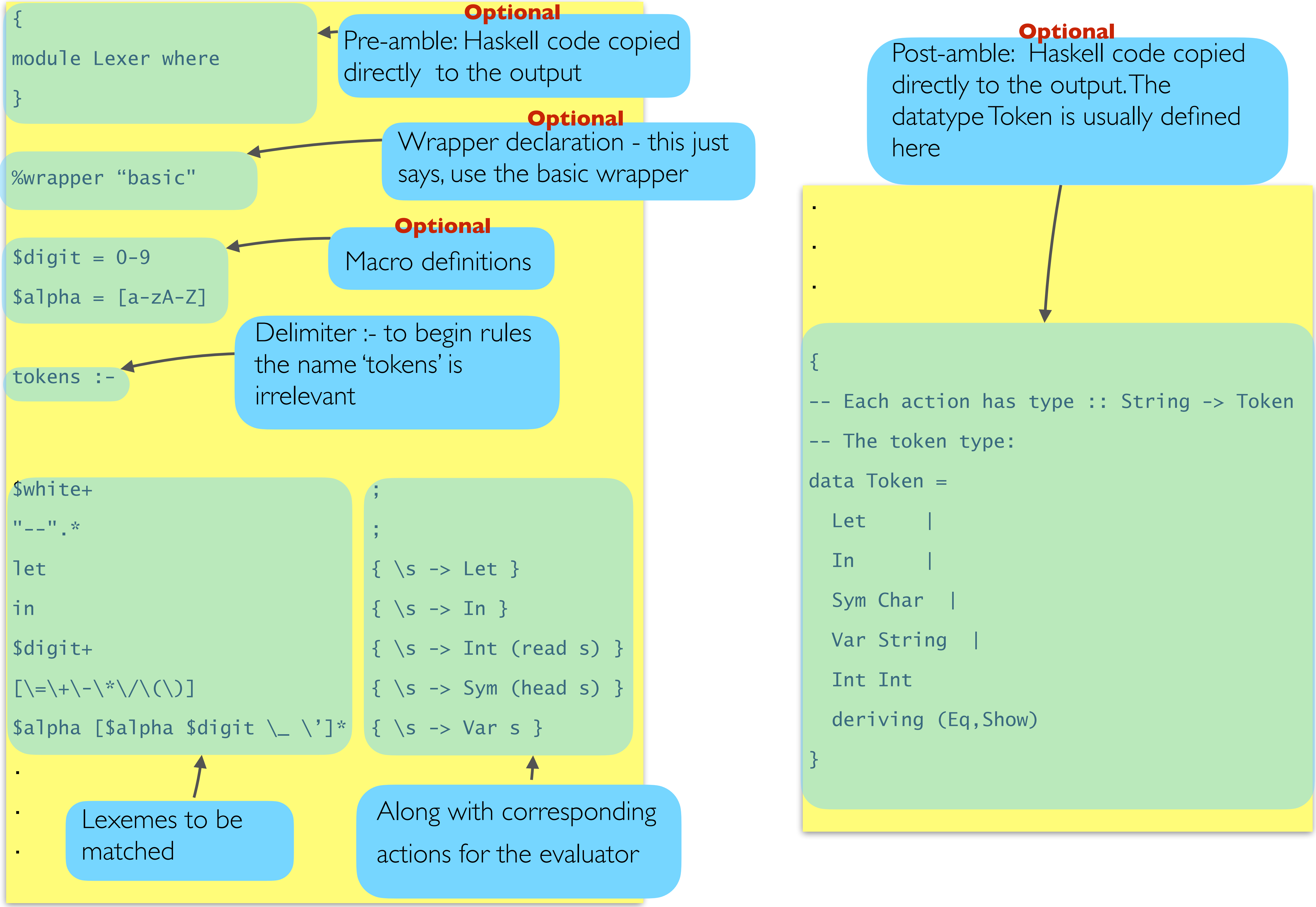
- This is a simple way of getting a function String → [Token]

This is all coded for you

```haskell
type AlexInput = (Char,  [Byte], String)
-- previous char
-- rest of the bytes for the current char
-- rest of the input string

alexGetByte :: AlexInput -> Maybe (Byte,AlexInput)
alexGetByte (c,(b:bs),s) = Just (b,(c,bs,s))
alexGetByte (c,[],[])    = Nothing
alexGetByte (_,[],(c:s)) = case utf8Encode c of
                             (b:bs) -> Just (b, (c, bs, s))

alexInputPrevChar :: AlexInput -> Char
alexInputPrevChar (c,_,_) = c

alexScanTokens :: String -> [Token]
alexScanTokens str = go ('\n',[],str)
  where go inp@(_,_bs,str) =
          case alexScan inp 0 of
                AlexEOF -> []
                AlexError _ -> error "lexical error"
                AlexSkip  inp' len     -> go inp'
                AlexToken inp' len act -> act (take len str) : go inp'
```

Note the type of actions here : String → Token

ANATOMY OF AN ALEX FILE



```
{
module Lexer where
}

%wrapper “basic"

$digit = 0-9
$alpha = [a-zA-Z]

tokens :-

$white+                           ;
"--".*                            ;
let                               { \s -> Let }
in                                { \s -> In }
$digit+                           { \s -> Int (read s) }
[\=\+\-\*\/\(\)]                  { \s -> Sym (head s) }
$alpha [$alpha $digit \_ \’]*     { \s -> Var s }
.
.
.
```

Optional

Pre-amble: Haskell code copied directly to the output

Optional

Wrapper declaration - this just says, use the basic wrapper

Optional

Macro definitions

Delimiter :- to begin rules the name ‘tokens’ is irrelevant

Lexemes to be matched

Along with corresponding actions for the evaluator

Optional

Post-amble: Haskell code copied directly to the output. The datatype Token is usually defined here

```
.
.
.
{
-- Each action has type :: String -> Token
-- The token type:
data Token =
  Let      |
  In       |
  Sym Char   |
  Var String   |
  Int Int
  deriving (Eq,Show)
}
```

WRAPPERS

- There are other pre-defined wrappers available: posn, monad, monadUserState, and ByteString wrappers
  - You are unlikely to need any of these other than ‘posn’
- The posn wrapper keeps track of line and column numbers of tokens in the input text.

```
data AlexPosn = AlexPn !Int  -- absolute character offset
                       !Int  -- line number
                       !Int  -- column number

type AlexInput = (AlexPosn,     -- current position,
                  Char,         -- previous char
                  [Byte],       -- rest of the bytes for the current char
                  String)       -- current input string

alexScanTokens :: String -> [Token]
alexScanTokens str = go (alexStartPos,'\n',[],str)
  where go inp@(pos,_,_,str) =
     case alexScan inp 0 of
        AlexEOF -> []
        AlexError ((AlexPn _ line column),_,_,_) ->
            error $ "lexical error at " ++ (show line) ++ " line, " ++ (show column) ++ " column"
        AlexSkip  inp' len     -> go inp'
        AlexToken inp' len act -> act pos (take len str) : go inp'
```

Again, note the type of actions:

AlexPosn → String → Token