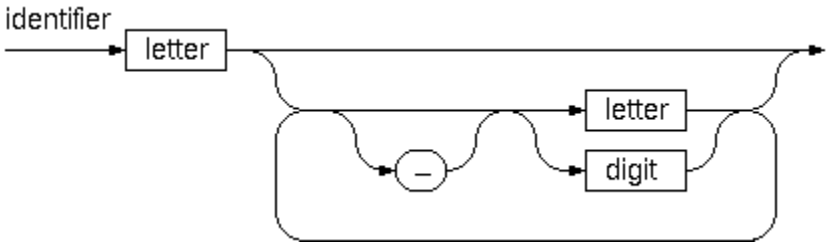
# 99 questions/Solutions/96

## From HaskellWiki

< 99 questions | Solutions

(**) Syntax checker

In a certain programming language (Ada) identifiers are defined by the syntax diagram below.




Transform the syntax diagram into a system of syntax diagrams which do not contain loops; i.e. which are purely recursive. Using these modified diagrams, write a predicate identifier/1 that can check whether or not a given string is a legal identifier.

```
import Data.Char
syntax_check :: String -> Bool
syntax_check []     = False
syntax_check (x:xs) = isLetter x && loop xs
    where loop [] = True
          loop (y:ys) | y == '-'     = (not . null) ys && isAlphaNum (head ys) && loop (tail
                      | isAlphaNum y = loop ys
                      | otherwise    = False
```

Simple functional transcription of the diagram.

Another direct transcription of the diagram:

```
identifier :: String -> Bool
identifier (c:cs) = isLetter c && hyphen cs
  where hyphen [] = True
        hyphen ('-':cs) = alphas cs
        hyphen cs = alphas cs
        alphas [] = False
        alphas (c:cs) = isAlphaNum c && hyphen cs
```

The functions `hyphen` and `alphas` correspond to states in the automaton at the start of the loop and before a compulsory alphanumeric, respectively.

This solution explicitly describes a finite state machine for the syntax:

```
identifier :: String -> Bool
identifier s = identifier' s 'l'
   where
      identifier' []    t = t == 'e'
      identifier' (c:s) t = (match c t) && or [identifier' s b | (a, b) <- fsm, a == t]
      fsm = [('l', 'e'), ('l', 'l'), ('l', '-'), ('l', 'd'), ('-', 'l'),
          ('-', 'd'), ('d', 'e'), ('d', '-'), ('d', 'l'), ('d', 'd')]
      match c t | t == '-' = c == '-'
            | t == 'd' = '0' <= c && c <= '9'
            | t == 'l' = 'a' <= c && c <= 'z' || 'A' <= c && c <= 'Z'
```

Here is a solution that parses the identifier using Parsec, a parser library that is commonly used in Haskell code:

```
identifier x = either (const False) (const True) $ parse parser "" x where
   parser = letter >> many (optional (char '-') >> alphaNum)
```

Or we can use regular expression ( in this case Text.RegexPR ):

```
import Text.RegexPR
import Data.Maybe

identifier = isJust . matchRegexPR "^[a-zA-Z](-?[a-zA-Z0-9])*$"
```



---