Práctica 1: Parser - Soluciones

1. Solución Ejercicio 5:
Escribir un parser para listas heterogéneas de enteros y caracteres por extensión usando el formato de Haskell. Defina un tipo de datos adecuado para representar estas listas parseadas. Por ejemplo, una cadena a parsear es la siguiente:

$[1, \texttt{'a'}, \texttt{'b'}, 2, 3, \texttt{'c'}]$

Solución:

```
nc :: Parser (Either Int Char)
nc =
  do n ← int
     return (Left n)
   ⟨|⟩ (do symbol "'"
           c ← letter
           symbol "'"
           return (Right c)
           )
ncl :: Parser [Either Int Char]
ncl = do token (char '[')
         xs ← sepBy (token nc) (char ',')
         token (char ']')
         return xs
```

2. Solución Ejercicio 8:
Transformar la gramática para eliminar la recursión izquierda e implementar el parser para la gramática transformada.

$$
\begin{aligned}
expr &\rightarrow expr\ (\texttt{'+'}\ term \mid \texttt{'-'}\ term) \mid term \\
term &\rightarrow term\ (\texttt{'*'}\ factor \mid \texttt{'/'}\ factor) \mid factor \\
factor &\rightarrow digit \mid \texttt{'('}\ expr\ \texttt{')'} \\
digit &\rightarrow \texttt{'0'} \mid \texttt{'1'} \mid \texttt{'2'} \mid ... \mid \texttt{'9'}
\end{aligned}
$$

Solución

Gramática sin recursión izquierda:

$$
\begin{aligned}
expr &\rightarrow term\ expr' \\
expr' &\rightarrow \varepsilon \mid (\texttt{'+'}\ term \mid \texttt{'-'}\ term)\ expr' \\
term &\rightarrow factor\ term' \\
term' &\rightarrow \varepsilon \mid (\texttt{'*'}\ factor \mid \texttt{'/'}\ factor)\ term' \\
factor &\rightarrow digit \mid \texttt{'('}\ expr\ \texttt{')'} \\
digit &\rightarrow \texttt{'0'} \mid \texttt{'1'} \mid \texttt{'2'} \mid ... \mid \texttt{'9'}
\end{aligned}
$$

```
expr_ :: Parser Int
expr_ = do x ← term_
           f ← expr_'
           return (f x)
```

```
expr_' :: Parser (Int → Int)
expr_' = do token (char '+')
            t ← term_
            f ← expr_'
            return (f ∘ (+t))
          ⟨|⟩ do token (char '-')
                 t ← term_
                 f ← expr_'
                 return (f ∘ (λx → x − t))
          ⟨|⟩ return id

term_ :: Parser Int
term_ = do x ← factor
           f ← term_'
           return (f x)

term_' :: Parser (Int → Int)
term_' = do token (char '*')
            f ← factor
            t ← term_'
            return (t ∘ (∗f))
          ⟨|⟩ do token (char '/')
                 f ← factor
                 t ← term_'
                 return (t ∘ (`div`f))
               ⟨|⟩ return id
```