Abordaje Funcional a EDSLs

Alberto Pardo Marcos Viera

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

ECI 2024

Parsers aplicativos

Combinadores de parsing

Los combinadores de parsing forman un EDSL que es implementado usando un shallow embedding.

Están formados por dos grupos de funciones:

- Funciones básicas que sirven para reconocer determinados strings de entrada
- Un grupo de combinadores que permiten construir nuevos parsers a partir de otros.

Parsers elementales

La mayoría de las bibliotecas de parsing están formadas por los siguientes 4 combinadores básicos:

string vacío	*pSucceed*
terminales	*pSym s*
alternativa	*p* <|> *q*
composición	*p* <*> *q*

El tipo de un parser

- Un parser puede ser entendido como una función que toma un string de entrada y retorna algo de tipo a:

$$String \rightarrow a$$

El tipo de un parser

- Un parser puede ser entendido como una función que toma un string de entrada y retorna algo de tipo a:

$$String \rightarrow a$$

- Un parser podría ser ambiguo y retornar varios posibles valores, significando que puede haber varias formas de reconocer la entrada.

$$String \rightarrow [a]$$

El tipo de un parser

- Un parser puede ser entendido como una función que toma un string de entrada y retorna algo de tipo a:

$$String \rightarrow a$$

- Un parser podría ser ambiguo y retornar varios posibles valores, significando que puede haber varias formas de reconocer la entrada.

$$String \rightarrow [a]$$

- Un parser podría no consumir toda la entrada y retornar además la parte de la entrada no consumida.

$$String \rightarrow [(a, String)]$$

El tipo de un parser

En resumen,

$$\texttt{type}\ \mathit{Parser}\ a = \mathit{String} \to [(a, \mathit{String})]$$

El tipo de un parser

En resumen,

$$\texttt{type}\ \mathit{Parser}\ a = \mathit{String} \to [(a, \mathit{String})]$$

Podemos abstraer el tipo *String*:

$$\texttt{type}\ \mathit{Parser}\ s\ a = \mathit{Eq}\ s \Rightarrow [s] \to [(a, [s])]$$

- En su lugar ponemos una lista de valores de tipo s.
- A los valores de tipo s les vamos a requerir que sean comparables por igualdad (instancia de la clase *Eq*).

Combinadores básicos

$$
\begin{aligned}
\mathit{pFail} &:: && && \mathit{Parser}\ s\ a \\
\mathit{pSucceed} &:: a && && \to \mathit{Parser}\ s\ a \\
\mathit{pSym} &:: \mathit{Eq}\ s \Rightarrow s && && \to \mathit{Parser}\ s\ s \\
\texttt{<|>} &:: \mathit{Parser}\ s\ a && \to \mathit{Parser}\ s\ a && \to \mathit{Parser}\ s\ a \\
\texttt{<*>} &:: \mathit{Parser}\ s\ (a \to b) && \to \mathit{Parser}\ s\ a && \to \mathit{Parser}\ s\ b
\end{aligned}
$$

Combinadores básicos

$$pFail :: Parser\ s\ a$$
$$pFail = \lambda cs \rightarrow [\,]$$

Combinadores básicos

$$pFail :: Parser\ s\ a$$
$$pFail = \lambda cs \rightarrow [\,]$$

$$pSucceed :: a \rightarrow Parser\ s\ a$$
$$pSucceed\ a = \lambda cs \rightarrow [(a, cs)]$$

Combinadores básicos

```
pFail :: Parser s a
pFail = λcs → []
```

```
pSucceed    :: a → Parser s a
pSucceed a = λcs → [(a, cs)]
```

```
pSym    :: Eq s ⇒ s → Parser s s
pSym s = λcs → case cs of
                 []       → []
                 (c : cs′) → if c == s
                               then [(c, cs′)]
                               else []
```

Combinadores básicos

$$(<|>) \quad :: \mathit{Parser}\ s\ a \to \mathit{Parser}\ s\ a \to \mathit{Parser}\ s\ a$$
$$p\ <|>\ q = \lambda cs \to p\ cs +\!\!+ q\ cs$$

Combinadores básicos

$$
\begin{aligned}
&(<|>) \quad :: \mathit{Parser}\ s\ a \to \mathit{Parser}\ s\ a \to \mathit{Parser}\ s\ a \\
&p\ <|>\ q = \lambda cs \to p\ cs +\!\!+ q\ cs
\end{aligned}
$$

$$
\begin{aligned}
&(<*>) \quad :: \mathit{Parser}\ s\ (a \to b) \to \mathit{Parser}\ s\ a \to \mathit{Parser}\ s\ b \\
&(p <*> q)\ cs = [(f\ a, cs'') \mid (f, cs') \leftarrow p\ cs \\
&\qquad\qquad\qquad\qquad\qquad\qquad , (a, cs'') \leftarrow q\ cs']
\end{aligned}
$$

Ejemplos de parsers (1)

- Reconocer una 'A' y retornar una 'B':

Ejemplos de parsers (1)

- Reconocer una 'A' y retornar una 'B':

$$pA2B = pSucceed\ (\lambda\_ \rightarrow \texttt{'B'})\ \texttt{<*>}\ pSym\ \texttt{'A'}$$

Ejemplos de parsers (1)

- Reconocer una 'A' y retornar una 'B':

$$pA2B = pSucceed\ (\lambda\_ \rightarrow \texttt{'B'})\ \texttt{<*>}\ pSym\ \texttt{'A'}$$

- Reconocer una 'A', seguida de una 'B', y retornar ambos caracteres en un par.

Ejemplos de parsers (1)

- Reconocer una 'A' y retornar una 'B':

$$pA2B = pSucceed\ (\lambda\_ \rightarrow \texttt{'B'})\ \texttt{<*>}\ pSym\ \texttt{'A'}$$

- Reconocer una 'A', seguida de una 'B', y retornar ambos caracteres en un par.

$$pAB = pSucceed\ (,)\ \texttt{<*>}\ pSym\ \texttt{'A'}\ \texttt{<*>}\ pSym\ \texttt{'B'}$$

Ejemplos de parsers (2)

- Parser que retorna una lista de valores de tipo a. Toma como parámetro un parser que retorna un a.

Ejemplos de parsers (2)

- Parser que retorna una lista de valores de tipo a. Toma como parámetro un parser que retorna un a.

$$
\begin{aligned}
&pList :: Parser\ s\ a \rightarrow Parser\ s\ [a] \\
&pList\ p = pSucceed\ (:)\ \texttt{<*>}\ p\ \texttt{<*>}\ pList\ p \\
&\qquad\qquad \texttt{<|>} \\
&\qquad\qquad pSucceed\ [\,]
\end{aligned}
$$

Ejemplos de parsers (2)

- Parser que retorna una lista de valores de tipo a. Toma como parámetro un parser que retorna un a.

$$
\begin{aligned}
&pList :: Parser\ s\ a \rightarrow Parser\ s\ [a] \\
&pList\ p = pSucceed\ (:)\ \texttt{<*>}\ p\ \texttt{<*>}\ pList\ p \\
&\qquad\qquad \texttt{<|>} \\
&\qquad\qquad pSucceed\ [\,]
\end{aligned}
$$

- Parser que reconoce un string de la forma $(AB)^*$.

Ejemplos de parsers (2)

- Parser que retorna una lista de valores de tipo a. Toma como parámetro un parser que retorna un a.

$$
\begin{aligned}
&pList :: Parser\ s\ a \rightarrow Parser\ s\ [a] \\
&pList\ p = pSucceed\ (:)\ \texttt{<*>}\ p\ \texttt{<*>}\ pList\ p \\
&\qquad\qquad \texttt{<|>} \\
&\qquad\qquad pSucceed\ [\,]
\end{aligned}
$$

- Parser que reconoce un string de la forma $(AB)^*$.

$$pListAB = pList\ pAB$$

Otros combinadores útiles

```
(<$>)  :: (a → b) → Parser s a → Parser s b
f <$> p = pSucceed f <*> p
```

```
opt       :: Parser s a → a → Parser s a
p `opt` a = p <|> pSucceed a
```

```
pSat   :: (s → Bool) → Parser s s
pSat p = λcs → case cs of
                 []       → []
                 (c : cs') → if p c
                               then [(c, cs')]
                               else []
```

Ejemplos

- Definición de *pAB* usando <$>:

$$pAB = (,) \texttt{<\$>} pSym\ \texttt{'A'} \texttt{<*>} pSym\ \texttt{'B'}$$

Ejemplos

- Definición de *pAB* usando <$>:

$$pAB = (,) \texttt{<\$>} pSym\ \texttt{'A'} \texttt{<*>} pSym\ \texttt{'B'}$$

- Definición de *pSym* usando *pSat*:

$$pSym\ a = pSat\ (== a)$$

Ejemplos

- Definición de *pAB* usando <$>:

$$pAB = (,) \texttt{<\$>} pSym\ \texttt{'A'} \texttt{<*>} pSym\ \texttt{'B'}$$

- Definición de *pSym* usando *pSat*:

$$pSym\ a = pSat\ (== a)$$

- Reconocer un dígito:

$$pDigit = pSat\ isDigit$$
$$\texttt{where}$$
$$isDigit\ c = (c \geqslant \texttt{'0'}) \wedge (c \leqslant \texttt{'9'})$$

Ejemplos

- Definición de *pAB* usando <$>:

$$pAB = (,) \texttt{<\$>} pSym\ \texttt{'A'} \texttt{<*>} pSym\ \texttt{'B'}$$

- Definición de *pSym* usando *pSat*:

$$pSym\ a = pSat\ (== a)$$

- Reconocer un dígito:

$$pDigit = pSat\ isDigit$$
$$\texttt{where}$$
$$isDigit\ c = (c \geqslant \texttt{'0'}) \wedge (c \leqslant \texttt{'9'})$$

- Definición de *pList* usando <$> y *opt*:

$$pList\ p = (:) \texttt{<\$>} p \texttt{<*>} pList\ p\ `opt`\ []$$

Selección de resultados de parsers

```
(<*)  :: Parser s a → Parser s b → Parser s a
p <* q = (λx _ → x) <$> p <*> q
```

```
(*>)  :: Parser s a → Parser s b → Parser s b
p *> q = (λ_ y → y) <$> p <*> q
```

```
(<$)  :: a → Parser s b → Parser s a
a <$ q = pSucceed a <* q
```

Selección de resultados de parsers

$$(<*) \quad :: \mathit{Parser}\ s\ a \to \mathit{Parser}\ s\ b \to \mathit{Parser}\ s\ a$$
$$p <* q = (\lambda x\ \_ \to x) <\$> p <*> q$$

$$(*>) \quad :: \mathit{Parser}\ s\ a \to \mathit{Parser}\ s\ b \to \mathit{Parser}\ s\ b$$
$$p *> q = (\lambda \_\ y \to y) <\$> p <*> q$$

$$(<\$) \quad :: a \to \mathit{Parser}\ s\ b \to \mathit{Parser}\ s\ a$$
$$a <\$ q = \mathit{pSucceed}\ a <* q$$

Ejemplo. Reconocer algo entre paréntesis.

$$\mathit{pParens}\ p = \mathit{pSym}\ '(' *> p <* \mathit{pSym}\ ')'$$