Total Parser Combinators

Nils Anders Danielsson (Nottingham)

MGS Christmas Seminars, Sheffield, 2009-12-15

Parser combinator example

$$
\begin{aligned}
\mathit{expr} &= \mathit{term} \cdot \mathit{tok}\ \texttt{"-"} \cdot \mathit{expr} \\
&\mid \mathit{term} \\
\mathit{term} &= \ldots
\end{aligned}
$$

Parser combinator propaganda

- Parser combinator libraries are great!
- Elegant code.
- Executable grammars.
- Easy to abstract out recurring patterns.
- Light-weight.
- Nowadays often fast enough.

Parser combinator example

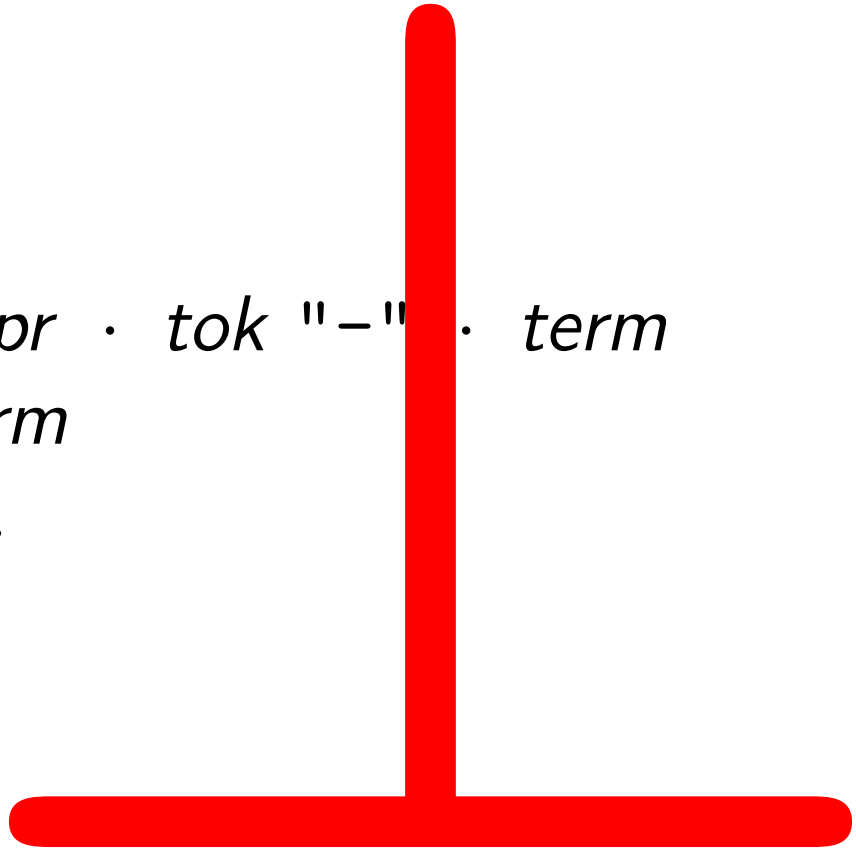
$$
\begin{aligned}
\mathit{expr} &= \mathit{term} \cdot \mathit{tok}\ \texttt{"-"} \cdot \mathit{expr} \\
&\mid \mathit{term} \\
\mathit{term} &= \ldots
\end{aligned}
$$

Left recursion

$$
\begin{aligned}
\mathit{expr} &= \mathit{expr} \cdot \mathit{tok}\ \texttt{"-"} \cdot \mathit{term} \\
&\mid \mathit{term} \\
\mathit{term} &= \ldots
\end{aligned}
$$

Left recursion

$$
\begin{array}{lcl}
\mathit{expr} & = & \mathit{expr} \cdot \mathit{tok}\ \texttt{"-"} \cdot \mathit{term} \\
& | & \mathit{term} \\
\mathit{term} & = & \ldots
\end{array}
$$



Infinite ambiguity

$\mathit{return}\ x$	The empty string.
fail	Nothing.
$\mathit{many}\ p$	Zero or more occurrences of p.
$p \gg\!= f$	First p, then f applied to result of p.

- $\mathit{many}\ (\mathit{return}\ x) \quad \mapsto \quad [], [x], [x, x], \ldots$
- $\mathit{many}\ (\mathit{return}\ x) \gg\!= \lambda\ \_\ \to\ \mathit{fail} \quad \mapsto \quad ?$

Infinite ambiguity

return x	The empty string.
fail	Nothing.
many p	Zero or more occurrences of p.
$p \gg\!= f$	First p, then f applied to result of p.

- $\mathit{many}\ (\mathit{return}\ x) \quad \mapsto \quad [], [x], [x, x], \ldots$
- $\mathit{many}\ (\mathit{return}\ x) \gg\!= \lambda\ \_\ \to\ \mathit{fail} \quad \mapsto \quad ?$

Techniques for explicit invariants

- Separation of induction and coinduction.
- Indexed types.

Induction and coinduction in Agda

Inductive types

$$
\begin{aligned}
&\textbf{data}\ \mathit{List}\ (A\ :\ \mathit{Set})\ :\ \mathit{Set}\ \textbf{where}\\
&\quad [] \quad\ :\ \mathit{List}\ A\\
&\quad \_::\_\ :\ A\ \to\ \mathit{List}\ A\ \to\ \mathit{List}\ A
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{finite}\ :\ \mathit{List}\ \mathbb{N}\\
&\mathit{finite}\ =\ 0 :: 0 :: 0 :: []
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{map}\ :\ \forall\ \{A\ B\}\ \to\ (A\ \to\ B)\ \to\ \mathit{List}\ A\ \to\ \mathit{List}\ B\\
&\mathit{map}\ f\ []\ \ \ \ \ \ \ \ \ \ \ \ =\ []\\
&\mathit{map}\ f\ (x :: \mathit{xs})\ =\ f\ x :: \mathit{map}\ f\ \mathit{xs}
\end{aligned}
$$

Coinductive types

$$\textbf{data}\ \mathit{Stream}\ (A\ :\ \mathit{Set})\ :\ \mathit{Set}\ \textbf{where}$$
$$\quad \_::\_\ :\ A \to \infty\ (\mathit{Stream}\ A) \to \mathit{Stream}\ A$$

- ∞ marks coinductive arguments.
- Can be seen as a suspension.
- Delay and force:

$$\sharp : \forall\, \{A\} \to A \to \infty\, A$$
$$\flat : \forall\, \{A\} \to \infty\, A \to A$$

Coinductive types

$$\textbf{data}\ \mathit{Stream}\ (A\ :\ \mathit{Set})\ :\ \mathit{Set}\ \textbf{where}$$
$$\quad \_::\_\ :\ A \to \infty\ (\mathit{Stream}\ A) \to \mathit{Stream}\ A$$

$$\mathit{infinite}\ :\ \mathit{Stream}\ \mathbb{N}$$
$$\mathit{infinite}\ =\ 0 :: \sharp\ \mathit{infinite}$$

$$\mathit{map}\ :\ \forall\ \{A\ B\} \to (A \to B) \to \mathit{Stream}\ A \to \mathit{Stream}\ B$$
$$\mathit{map}\ f\ (x :: xs)\ =\ f\ x :: \sharp\ (\mathit{map}\ f\ (\flat\ xs))$$

Fixpoint view

$$T = F\ (\infty\ T)\ T \quad \approx \quad T = \nu C.\mu I.\ F\ C\ I$$

$\mathit{Stream}\ A \approx \nu C.A \times C$:

```
data Stream (A : Set) : Set where
  _::_ : A → ∞ (Stream A) → Stream A
```

Mixed induction and coinduction

$$T = F\ (\infty\ T)\ T \quad \approx \quad T = \nu C.\mu I.\ F\ C\ I$$

$$SP\ A\ B \approx \nu C.\mu I.\ (A \to I) + B \times C:$$

$$\begin{array}{l}
\textbf{data}\ SP\ (A\ B\ :\ Set)\ :\ Set\ \textbf{where} \\
\quad \mathrm{get}\ :\ (A \to SP\ A\ B) \to SP\ A\ B \\
\quad \mathrm{put}\ :\ B \to \infty\ (SP\ A\ B) \to SP\ A\ B
\end{array}$$

Examples

- Not OK:

$$
\begin{aligned}
&\mathit{sink} \;:\; \forall\, \{A\ B\} \;\to\; \mathit{SP}\ A\ B \\
&\mathit{sink} \;=\; \mathsf{get}\ (\lambda\ \_ \;\to\; \mathit{sink})
\end{aligned}
$$

- OK:

$$
\begin{aligned}
&\mathit{const} \;:\; \forall\, \{A\ B\} \;\to\; B \;\to\; \mathit{SP}\ A\ B \\
&\mathit{const}\ x \;=\; \mathsf{put}\ x\ (\sharp\ (\mathit{const}\ x))
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{copy} \;:\; \forall\, \{A\} \;\to\; \mathit{SP}\ A\ A \\
&\mathit{copy} \;=\; \mathsf{get}\ (\lambda\ x \;\to\; \mathsf{put}\ x\ (\sharp\ \mathit{copy}))
\end{aligned}
$$

Examples

Lexicographic guarded corecursion and
higher-order structural recursion:

$$
\begin{aligned}
&[\![\_]\!] \;:\; \forall\,\{A\;B\} \to SP\;A\;B \to Stream\;A \to Stream\;B \\
&[\![\;\mathsf{get}\;f \quad\;]\!]\;(a :: as) \;=\; [\![\;f\;\;a\;]\!]\;(\flat\;as) \\
&[\![\;\mathsf{put}\;b\;sp\;]\!]\;as \qquad\quad\;\; = \;\; b :: \sharp\;([\![\;\flat\;sp\;]\!]\;as)
\end{aligned}
$$

Parser combinators

Interface (roughly)

Recognisers:

$$
\begin{aligned}
&P &&: \mathit{Set}\\
&\_\in\_ &&: \mathit{List}\ \mathit{Token} \to P \to \mathit{Set}\\
&\_\in?\_ &&: (s : \mathit{List}\ \mathit{Token})\ (p : P) \to \mathit{Dec}\ (s \in p)
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\ \mathit{Dec}\ (A : \mathit{Set}) : \mathit{Set}\ \textbf{where}\\
&\quad \mathsf{yes} : A \to \mathit{Dec}\ A\\
&\quad \mathsf{no} : \neg A \to \mathit{Dec}\ A
\end{aligned}
$$

Interface (roughly)

$$
\begin{array}{lll}
\emptyset & : & P \\
\varepsilon & : & P \\
\mathsf{sat} & : & (\mathit{Token} \to \mathit{Bool}) \to P \\
\_|\_ & : & P \to P \to P \\
\_\cdot\_ & : & P \to P \to P
\end{array}
$$

Some arguments should be coinductive.

Choice

Hard to decide infinite choice:

$$p = p \mid p'$$
$$p = p' \mid p$$

The arguments of $\_|\_$ will be inductive.

Sequencing

Problematic if p′ is nullable, otherwise OK:

$$p = p \cdot p'$$
$$p = p' \cdot p$$

Let us index parsers by their nullability:

- $P : \mathit{Bool} \to \mathit{Set}$.
- $p : P$ true if p accepts the empty string.
- $p : P$ false otherwise.

Conditional coinduction

$$
\begin{aligned}
&\textbf{data}\ \infty?\ (A\ :\ \mathit{Set})\ :\ \mathit{Bool}\ \to\ \mathit{Set}\ \textbf{where}\\
&\quad \langle\_\rangle\ \ :\ \ \ \ A\ \to\ \infty?\ A\ \mathsf{true}\\
&\quad \langle\!\langle\_\rangle\!\rangle\ :\ \infty\ A\ \to\ \infty?\ A\ \mathsf{false}
\end{aligned}
$$

$$
\begin{aligned}
&\sharp?\ :\ \forall\ \{b\ A\}\ \to\ A\ \to\ \infty?\ A\ b\\
&\sharp?\ \{\mathsf{true}\}\ x\ =\ \langle\ \ \ x\ \rangle\\
&\sharp?\ \{\mathsf{false}\}\ x\ =\ \langle\!\langle\ \sharp\ x\ \rangle\!\rangle
\end{aligned}
$$

$$
\begin{aligned}
&\flat?\ :\ \forall\ \{b\ A\}\ \to\ \infty?\ A\ b\ \to\ A\\
&\flat?\ \langle\ x\ \rangle\ \ =\ \ \ x\\
&\flat?\ \langle\!\langle\ x\ \rangle\!\rangle\ =\ \flat\ x
\end{aligned}
$$

Recognisers

$$
\begin{aligned}
&\textbf{data}\ P\ :\ \mathit{Bool}\ \to\ \mathit{Set}\ \textbf{where}\\
&\quad \emptyset\ \ :\ P\ \mathsf{false}\\
&\quad \varepsilon\ \ :\ P\ \mathsf{true}\\
&\quad \mathsf{sat}\ \ :\ (\mathit{Token}\ \to\ \mathit{Bool})\ \to\ P\ \mathsf{false}\\
&\quad \_|\_\ \ :\ \forall\ \{n_1\ n_2\}\ \to\ P\ n_1\ \to\ P\ n_2\ \to\ P\ (n_1 \vee n_2)\\
&\quad \_\cdot\_\ \ :\ \forall\ \{n_1\ n_2\}\ \to\ \infty?\ (P\ n_1)\ n_2\\
&\qquad\qquad\qquad\ \to\ \infty?\ (P\ n_2)\ n_1\ \ \to\ P\ (n_1 \wedge n_2)
\end{aligned}
$$

Example

Kleene star:

```
mutual
  _⋆ : P false → P true
  p ⋆ = ε ∣ p +
  _+ : P false → P false
  p + = ⟨ p ⟩ · ⟪ ♯ (p ⋆) ⟫
```

The argument must not accept the empty string:
$\varepsilon \star \approx$ *many* (*return* x) is not accepted.

Example

Left recursive Kleene star:

```
mutual
  _⋆ : P false → P true
  p ⋆ = ε | p +
  _+ : P false → P false
  p + = ⟪ ♯ (p ⋆) ⟫ · ⟨ p ⟩
```

Semantics

Inductive:

$$
\begin{aligned}
&\textbf{data}\ \_\in\_ : \mathit{List}\ \mathit{Token} \to P\ n \to \mathit{Set}\ \textbf{where}\\
&\quad \varepsilon \quad : [\,] \in \varepsilon\\
&\quad \mathsf{sat} \ : f\ t \equiv \mathsf{true} \to [\,t\,] \in \mathsf{sat}\ f\\
&\quad \mid^{\ell} \quad : s \in p_1 \to s \in p_1 \mid p_2\\
&\quad \mid^{\mathsf{r}} \quad : s \in p_2 \to s \in p_1 \mid p_2\\
&\quad \_\cdot\_ \ : s_1 \in \flat? \ p_1 \to s_2 \in \flat? \ p_2 \to\\
&\qquad\quad s_1 +\!\!+ s_2 \in p_1 \cdot p_2
\end{aligned}
$$

For $p : P\ n$:

$[\,] \in p$ iff $n \equiv \mathsf{true}$.

Implementation

- Uses a variant of Brzozowski's regular expression derivatives.
- $\partial : \forall\ \{n\}\ (p : P\ n)\ (t : \mathit{Token}) \rightarrow P\ (\partial n\ p\ t)$.
- $s \in \partial\ p\ t$ iff $t :: s \in p$.
- ∂ is used once per element in the input string.
- ∂ is productive.

∂

$$\partial : \forall \{n\}\ (p : P\ n)\ (t : \mathit{Token}) \to P\ (\partial n\ p\ t)$$
$$\partial\ \emptyset \qquad t = \emptyset$$
$$\partial\ \varepsilon \qquad t = \emptyset$$
$$\partial\ (\mathsf{sat}\ f)\ t\ \mathbf{with}\ f\ t$$
$$\partial\ (\mathsf{sat}\ f)\ t \mid \mathsf{true} = \varepsilon$$
$$\partial\ (\mathsf{sat}\ f)\ t \mid \mathsf{false} = \emptyset$$

∂

$$\partial\ (p_1 \mid p_2)\ t\ =\ \partial\ p_1\ t \mid \partial\ p_2\ t$$

$$\partial\ (\langle\ p_1\ \rangle\ \cdot \langle\ p_2\ \rangle)\ t\ =\ \langle\ \ \partial\ \ p_1\ \ t\ \ \rangle\ \cdot \sharp?\ p_2 \mid \partial\ p_2\ t$$
$$\partial\ (\langle\langle\ p_1\ \rangle\rangle \cdot \langle\ p_2\ \rangle)\ t\ =\ \langle\langle\ \sharp\ (\partial\ (\flat\ p_1)\ t)\ \rangle\rangle \cdot \sharp?\ p_2 \mid \partial\ p_2\ t$$

$$\partial\ (\langle\ p_1\ \rangle\ \cdot \langle\langle\ p_2\ \rangle\rangle)\ t\ =\ \langle\ \ \partial\ \ p_1\ \ t\ \ \rangle\ \cdot \sharp?\ (\flat\ p_2)$$
$$\partial\ (\langle\langle\ p_1\ \rangle\rangle \cdot \langle\langle\ p_2\ \rangle\rangle)\ t\ =\ \langle\langle\ \sharp\ (\partial\ (\flat\ p_1)\ t)\ \rangle\rangle \cdot \sharp?\ (\flat\ p_2)$$

Wrapping up

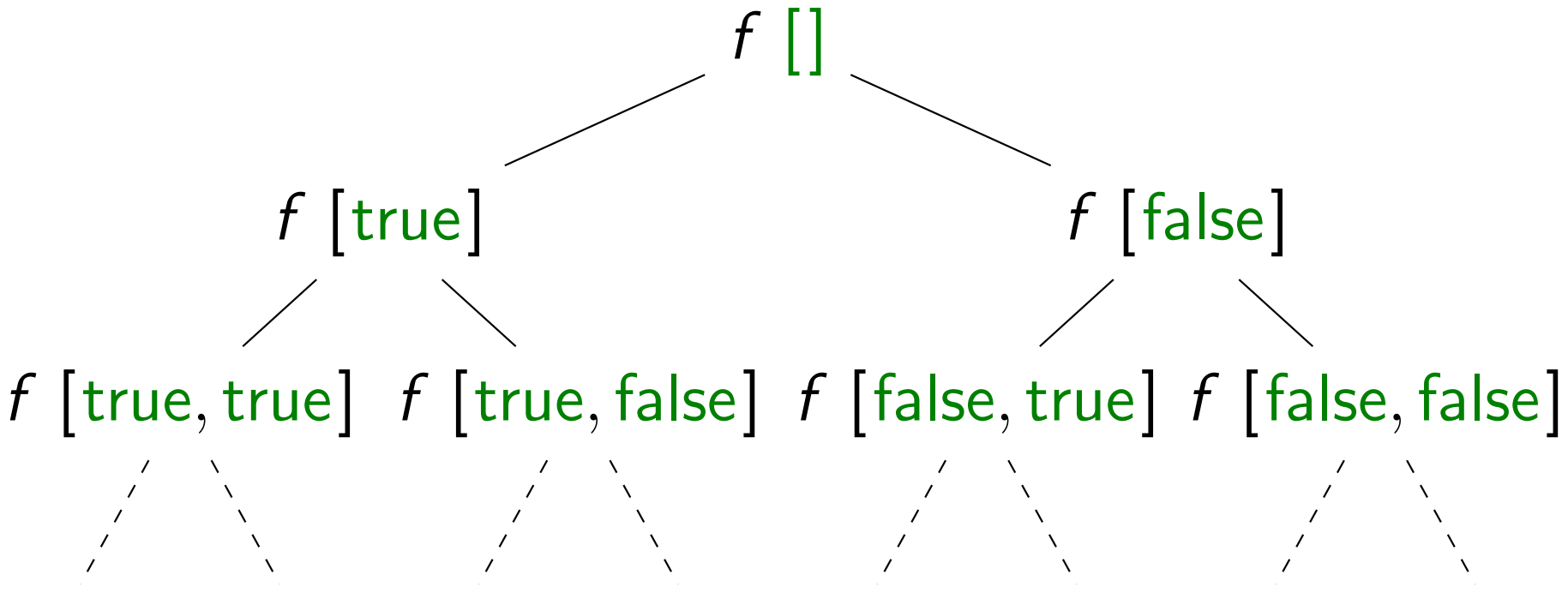
$$
\begin{aligned}
&\partial\text{-sound} &&: s \in \partial\ p\ t \to t :: s \in p\\
&\partial\text{-complete} &&: t :: s \in p \to s \in \partial\ p\ t\\
&\mathit{nullable?} &&: \forall\ \{n\}\ (p : P\ n) \to \mathit{Dec}\ ([\,] \in p)
\end{aligned}
$$

$$
\begin{aligned}
&\_\in?\_ : \forall\ \{n\}\ (s : \mathit{List}\ \mathit{Token})\ (p : P\ n) \to\\
&\qquad \mathit{Dec}\ (s \in p)\\
&[\,] \in?\ p = \mathit{nullable?}\ p\\
&t :: s \in?\ p\ \textbf{with}\ s \in?\ \partial\ p\ t\\
&t :: s \in?\ p \mid \text{yes}\ s{\in}\partial pt = \text{yes}\ (\partial\text{-sound}\ s{\in}\partial pt)\\
&t :: s \in?\ p \mid \text{no}\ \ s{\notin}\partial pt = \text{no}\ \ (s{\notin}\partial pt \circ \partial\text{-complete})
\end{aligned}
$$

Expressive strength

For finite alphabets the combinators are as expressive as possible:

$f \ : \ \mathit{List}\ \mathit{Bool} \ \rightarrow \ \mathit{Bool}$



Expressive strength

$$
\begin{aligned}
&\mathit{accept\text{-}if\text{-}true} : (b : \mathit{Bool}) \to P\ b \\
&\mathit{accept\text{-}if\text{-}true}\ \mathsf{true} = \varepsilon \\
&\mathit{accept\text{-}if\text{-}true}\ \mathsf{false} = \emptyset
\end{aligned}
$$

$$
\begin{aligned}
&p : (f : \mathit{List}\ \mathit{Bool} \to \mathit{Bool}) \to P\ (f\ [\,]) \\
&p\ f = \\
&\quad \langle\!\langle\ \sharp\ (p\ (\lambda xs \to f\ (xs +\!\!+ [\mathsf{true}\,]))) \rangle\!\rangle \cdot \sharp?\ (\mathsf{sat}\ \mathit{id}\ ) \\
&\quad \mid \langle\!\langle\ \sharp\ (p\ (\lambda xs \to f\ (xs +\!\!+ [\mathsf{false}]))) \rangle\!\rangle \cdot \sharp?\ (\mathsf{sat}\ \mathit{not}) \\
&\quad \mid \mathit{accept\text{-}if\text{-}true}\ (f\ [\,])
\end{aligned}
$$

Laws

- The combinators form a Kleene algebra.
- Need to generalise the Kleene star:

$$\_\bigstar : \forall\, \{n\} \;\to\; P\ n \;\to\; P\ \mathsf{true}$$
$$p \bigstar = (\mathit{nonempty}\ p) \star$$

- The *nonempty* combinator can be defined by structural recursion:

$$\mathit{nonempty} : \forall\, \{n\} \;\to\; P\ n \;\to\; P\ \mathsf{false}$$

Conclusions

- Mixed induction/coinduction:
  Precise control of size of data.
- I encourage you to add this technique
  to your toolbox.