A Theory of Parsers

Christoph-Simon Senjak

Lehr- und Forschungseinheit für Theoretische Informatik
Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstr.67, 80538 München

Leeds Logic Colloquium 05.08.2016

Objective

- Creating simple, verified, non-interactive parsers for streams of data (In our case: Data compressed with Deflate/GZip) $\Rightarrow$ preferrably no sophisticated library.
- Having multiple ways of parsing the data, which have different drawbacks; possible need to replace parts of the parser with other algorithms later $\Rightarrow$ approach should be modular.
- Simple axiomatic specification, complexity moved into the proofs $\Rightarrow$ use relational specifications of the data format rather than just combined functions.

Relations

- We use relations to specify the format we want to parse.
- The relations have the type `output -> input -> Prop`.
- Often in practice: `input` is `list bool` or `list ascii` (simplicity, performance)
- A very simple example that parses exactly one bit:

```
Inductive OneBit : bool → list bool → Prop :=
| oneBit : forall b, OneBit b [b].
```

- We can first define such simple relations and create more complicated relations using combinators.

Combinators - 1

- Parsing relations can be combined in a monadic fashion
- A combinator can be defined that first applies the first relation, and then the second relation:

$$Q \gg\!=_c R :=$$
$$\mu_\xi(\forall_{b_q b_r a_q a_r}. Q\ b_q\ a_q \to R\ b_q\ b_r\ a_r \to \xi\ (c\ b_q\ b_r)\ (a_q +\!\!+ a_r))$$

- From this, more complicated combinators can be built, like `nTimes`, which applies a relation a given number of times.

Combinators - 2

A parser to parse a given number of bits into a natural number can be realized with:

```
Definition readBitsLSB (length : nat) : nat → LB → Prop :=
AppCombine (nTimesCons length OneBit) ListToNat.
```

where `nTimesCons` applies the parser for `OneBit` for length times and returns a bit list, and `AppCombine` maps that bit list using `ListToNat` to a natural number.

Strong Uniqueness

- Relations must satisfy **Strong Uniqueness**: for a given string, there is at most one initial segment that can be parsed:

$$\begin{aligned} \mathrm{StrongUnique}\, R \quad :\Leftrightarrow \forall_{a,b,l_a,l'_a,l_b,l'_b}. \quad & l_a +\!\!+ l'_a = l_b +\!\!+ l'_b \rightarrow \\ & R\, a\, l_a \rightarrow R\, b\, l_b \rightarrow \\ & a = b \wedge l_a = l_b \end{aligned}$$

- This is equivalent to the two conditions
  - $\forall_{a,b,l}. R\, a\, l \rightarrow R\, b\, l \rightarrow a = b$ (left-uniqueness)
  - $\forall_{a,b,l,l'}. R\, a\, l \rightarrow R\, b\, (l +\!\!+ l') \rightarrow l' = []$ (a parsed segment of the input list cannot be extended).
- Strong Uniqueness is inherited through the former combinators.

Strong Decidability

- The most important property is **Strong Decidability**, meaning that whether an initial segment exists is decidable.

$$\mathrm{StrongDec}\, R :\Leftrightarrow \forall_l. \quad \begin{aligned} &(\exists_{a,x,y}. l = x +\!\!+ y \wedge R\, a\, x) \vee \\ &(\neg(\exists_{a,x,y}. l = x +\!\!+ y \wedge R\, a\, x) \wedge \quad E) \end{aligned}$$

  where E is some error type (for us, it is a string giving an error message).
- This resembles an exception monad.
- A constructive proof of strong decidability yields a parser.
- Strong decidability is inherited through the former combinators.

A Theory of Parsers

- Advantages of our theory:
  - It is simple, it needs no sophisticated library.
  - It yields algorithms that can be extracted directly from proofs.
- Disadvantages:
  - The input must be total – otherwise, some relations would not be decidable
  - It does not fully allow for lazy-i/o, and combining exception monads consumes stack space.

Conclusions, Future Work

- We presented a way of specifying parsers. We use this theory in our verified implementation of Deflate (GZip).
- It should be easy to port it from Coq to other proof checkers.
- Some of the disadvantages of our theory should be solvable by using *Iteratees*, but this is future work (see `http://okmij.org/ftp/Haskell/Iteratee/describe.pdf`)