1 Syntax

$$
\begin{aligned}
S = {} & \textbf{pure } E \\
\mid\ & x = S_1; S_2 \\
\mid\ & f\ E^* \\
& \\
\mid\ & \textbf{fail} \\
\mid\ & S_1 \mathbin{[\!]} S_2 \\
\mid\ & S_1 \lhd S_2 \\
& \\
\mid\ & \textbf{get}E \\
\mid\ & \textbf{peek} \\
& \\
\mid\ & \textbf{parse } S\ E \\
\mid\ & \textbf{case } E \textbf{ of } (P \to S)^+ \\
& \\
E = {} & \ldots \text{language of expressions} \ldots
\end{aligned}
$$

Figure 1: The Core language

2 Semantics

In this section we present a few different formulations of the semantics of the Core language. Throughout, we use the following notation:

I	The type of streams of tokens (i.e., strings)
X, Y, Z	Refer to elements of I
$+\!\!+$	Concatenates members of I
V	The type of semantic values
u, v	Refer to elements of V

The dynamic environments are implicit, except in the relational specification. The notation $[\![\_]\!]^{x=v}$ indicates the dynamic environment is extended with a new binding of variable x to v.

We won't specify the details of expression language is as it is somewhat orthogonal to the semantics of parsers:

$$[\![E]\!] : V$$

2.1 Semantics as a State Transformer

One way to give semantics is to model them as a state transformer:

$$[\![S]\!] : I \to \{(V, I)\}$$

$$
\begin{aligned}
[\![\mathbf{pure}\ E]\!]\ X &= \{([\![E]\!], X)\} \\
[\![x = S_1; S_2]\!]\ X &= \{(v, Z) \mid (u, Y) \leftarrow [\![S_1]\!]\ X, (v, Z) \leftarrow [\![S_2]\!]^{x=u}\ Y\} \\
[\![\mathbf{fail}]\!]\ X &= \{\} \\
[\![S_1 \mathbin{[\!]} S_2]\!]\ X &= [\![S_1]\!]\ X \cup [\![S_2]\!]\ X \\
[\![S_1 \lhd S_2]\!]\ X &= \begin{cases} [\![S_1]\!]\ X & \text{if } [\![S_1]\!]\ X \neq \emptyset \\ [\![S_2]\!]\ X & \text{otherwise} \end{cases} \\
[\![\mathbf{get} E]\!]\ X &= \begin{cases} \{(Y, Z)\} & \text{if } |Y| = [\![E]\!] \wedge X = Y +\!\!+ Z \\ \emptyset & \text{otherwise} \end{cases} \\
[\![\mathbf{peek}]\!]\ X &= \{(X, X)\} \\
[\![\mathbf{parse}\ S\ E]\!]\ X &= \{(v, X) \mid (v, Y) \leftarrow [\![S]\!]\ [\![E]\!]\} \\
[\![\mathbf{case}\ E\ \mathbf{of}\ A]\!]\ X &= [\![\text{select}\ [\![E]\!]\ A]\!]\ X
\end{aligned}
$$

Figure 2: State transformer semantics of Core parsers.

2.2 Set Semantics

An alternative to state transformers is to describe a parser is a set of triples:

$$[\![S]\!] : \{(V, I, I)\}$$

If (v, X, Y) is in the semantics of S, then when applied to input $X +\!\!+ Y$, S will consume X and produce result v. This formulation allows us to talk about parsers in context. A parser *accepts* an input if it doesn't fail on it:

$$\text{accepts}\ S\ (X +\!\!+ Y) = \exists v.(v, X, Y) \in [\![S]\!]$$

$$
\begin{aligned}
[\![\mathbf{pure}\ E]\!] &= \{([\![E]\!], [], X)\} \\
[\![x = S_1; S_2]\!] &= \{(v, X +\!\!+ Y, Z) \mid (u, X, Y +\!\!+ Z) \leftarrow [\![S_1]\!], (v, Y, Z) \leftarrow [\![S_2]\!]^{x=u}\} \\
[\![\mathbf{fail}]\!] &= \{\} \\
[\![S_1 \mathbin{[\!]} S_2]\!] &= [\![S_1]\!] \cup [\![S_2]\!] \\
[\![S_1 \lhd S_2]\!] &= [\![S_1]\!] \cup \{(v, X, Y) \mid (v, X, Y) \leftarrow [\![S_2]\!], \text{not (accepts } S_1\ (X +\!\!+ Y))\} \\
[\![\mathbf{get} E]\!] &= \{(X, X, Y) \mid |X| = [\![E]\!]\} \\
[\![\mathbf{peek}]\!] &= \{(X, [], X)\} \\
[\![\mathbf{parse}\ S\ E]\!] &= \{(v, [], Z) \mid (v, X, Y) \leftarrow [\![S]\!], [\![E]\!] = X +\!\!+ Y\} \\
[\![\mathbf{case}\ E\ \mathbf{of}\ A]\!] &= [\![\text{select}\ [\![E]\!]\ A]\!]
\end{aligned}
$$

Figure 3: Set semantics of Core parsers.

Example of a parser that depends on context:

$$S = x = \textbf{peek}; \textbf{case}\ x\ \textbf{of}\ \{[] \rightarrow \textbf{fail}; \_ \rightarrow \textbf{pure}\ ()\}$$

This parser accepts the empty string, but only if is not at the end of the input.

2.3 Semantics as a Relation

This is an alternative presentation of the set semantics.

Pure
$$\frac{\Gamma \vdash E \ \rightarrow \ v}{\Gamma \vdash \textbf{pure}\ E \ \rightarrow \ v \rhd [] \cdot X}$$

Advance
$$\frac{\Gamma \vdash E \ \rightarrow \ |X|}{\Gamma \vdash \textbf{get}E \ \rightarrow \ X \rhd X \cdot Y}$$

Look-Ahead
$$\frac{}{\Gamma \vdash \textbf{peek} \ \rightarrow \ X \rhd [] \cdot X}$$

Sequnce
$$\frac{\Gamma \vdash S_1 \ \rightarrow \ u \rhd X \cdot Y +\!\!+ Z \qquad \Gamma, x = u \vdash S_2 \ \rightarrow \ v \rhd Y \cdot Z}{\Gamma \vdash x = S_1; S_2 \ \rightarrow \ v \rhd X +\!\!+ Y \cdot Z}$$

Unbiased-Choice-Left
$$\frac{\Gamma \vdash S_1 \ \rightarrow \ v \rhd X \cdot Y}{\Gamma \vdash S_1 \ [\!]\ S_2 \ \rightarrow \ v \rhd X \cdot Y}$$

Unbiased-Choice-Right
$$\frac{\Gamma \vdash S_2 \ \rightarrow \ v \rhd X \cdot Y}{\Gamma \vdash S_1 \ [\!]\ S_2 \ \rightarrow \ v \rhd X \cdot Y}$$

Biased-Choice-Left
$$\frac{\Gamma \vdash S_1 \ \rightarrow \ v \rhd X \cdot Y}{\Gamma \vdash S_1 \lhd S_2 \ \rightarrow \ v \rhd X \cdot Y}$$

Biased-Choice-Right
$$\frac{\Gamma \vdash S_2 \ \rightarrow \ v \rhd X \cdot Y \qquad \Gamma \vdash (X +\!\!+ Y) \notin S_1}{\Gamma \vdash S_1 \lhd S_2 \ \rightarrow \ v \rhd X \cdot Y}$$

Nested-Parser
$$\frac{\Gamma \vdash E \ \rightarrow \ X +\!\!+ Y \qquad \Gamma \vdash S \ \rightarrow \ v \rhd X \cdot Y}{\Gamma \vdash \textbf{parse}\ S\ E \ \rightarrow \ v \rhd [] \cdot Z}$$

Case
$$\frac{\Gamma \vdash E \ \rightarrow \ u \qquad \Gamma \vdash \text{select}\ u\ A \ \rightarrow \ v \rhd X \cdot Y}{\Gamma \vdash \textbf{case}\ E\ \textbf{of}\ A \ \rightarrow \ v \rhd X \cdot Y}$$

Figure 4: $\Gamma \vdash S \ \rightarrow \ v \rhd X \cdot Y$ describes the behavior or parser S in dynamic environment Γ. When applied to the input $X +\!\!+ Y$, S will consume X and produce semantic value v.

$$\frac{}{\Gamma \vdash X \notin \textbf{fail}}\ \textsc{Empty} \qquad \frac{\Gamma \vdash E \ \rightarrow\ v \qquad |X| < v}{\Gamma \vdash X \notin \textbf{get} E}\ \textsc{Too-Short}$$

$$\frac{\Gamma \vdash X \notin S_1 \qquad \Gamma \vdash X \notin S_2}{\Gamma \vdash X \notin S_1 \mathbin{[\!]} S_2}\ \textsc{Unbiased-Mismatch} \qquad \frac{\Gamma \vdash X \notin S_1 \qquad \Gamma \vdash X \notin S_2}{\Gamma \vdash X \notin S_1 \lhd S_2}\ \textsc{Biased-Mismatch}$$

$$\frac{\Gamma \vdash X \notin S_1}{\Gamma \vdash X \notin x = S_1; S_2}\ \textsc{Not-Front} \qquad \frac{\Gamma \vdash S_1 \ \rightarrow\ v \rhd X \cdot Y \qquad \Gamma, x = v \vdash Y \notin S_2}{\Gamma \vdash (X \mathbin{+\!\!+} Y) \notin x = S_1; S_2}\ \textsc{Not-Back}$$

$$\frac{\Gamma \vdash E \ \rightarrow\ v \qquad \Gamma \vdash v \notin P}{\Gamma \vdash X \notin \textbf{parse}\ P\ E}\ \textsc{Not-Nested} \qquad \frac{\Gamma \vdash E \ \rightarrow\ v \qquad \Gamma \vdash X \notin \text{select}\ v\ A}{\Gamma \vdash X \notin \textbf{case}\ E\ \textbf{of}\ A}\ \textsc{No-Case}$$

Figure 5: $\Gamma \vdash X \notin S$ asserts that X is not accepted by S in the sense described before.

3 Set vs. State Transformer Semantics

$$(v, X, Y) \in [\![S]\!]^{\text{set}} \iff (v, Y) \in [\![S]\!]^{\text{fun}}\ (X \mathbin{+\!\!+} Y)$$

Using state transformers is a more powerful abstraction than what is expressible in Core. In particular, consider an extension of Core that allows for direct stream manipulation, **setStream** E, which returns no interesting semantic value, but modifies the stream that we are parsing. Such a construct is readily expressible using the state transformers semantics:

$$[\![\textbf{setStream}\ E]\!]^{\text{fun}} X = \{((), [\![E]\!])\}$$

We cannot, however, express such a parser using the set-based semantics, because in this formalism, parsers declare constraints on a global stream, but they cannot change the actual stream. One attempt to define the semantics of such a construct could be:

$$[\![\textbf{setStream}\ E]\!]^{\text{set}} = \{((), [], [\![E]\!])\}$$

This, however, is incorrect because instead of changing the stream, we are making a look-ahead assertion about what the stream should be. Thus, we'll reject any inputs that do not match E, which is quite different than the intended semantics, which is a parser that never fails but modifies the input. As a concrete example, consider **setStream** "`a`": the input "`b`" is accepted by the transformer interpreation but not by the (incorrect) set interpreation.