1 Pratt Parsing

1.1 Language:

$$
\begin{array}{lcll}
t & ::= & a & \text{(atom)} \\
 & | & \odot & \text{(operator)} \\
 & & & \\
\odot & ::= & {}_i\odot_j & \text{(infix operator)} \\
 & | & \odot_i & \text{(prefix operator)} \\
 & | & {}_i\odot & \text{(suffix operator)} \\
 & & & \\
\mathit{state} & ::= & \overline{t} \lceil \overline{\odot} \rceil \overline{t} & \text{(parse expression)} \\
 & & & \\
 & | & \overline{t} \lfloor \overline{\odot} \rfloor \overline{t} & \text{(parse suffix)}
\end{array}
$$

$${}_?\odot_j \quad \textit{is shorthand for} \quad {}_i\odot_j \text{ or } \odot_j$$
$${}_i\odot_? \quad \textit{is shorthand for} \quad {}_i\odot_j \text{ or } {}_i\odot$$

1.2 Evaluation:

- The inital state for a token stream $\overline{t}$ is $\lceil \rceil \overline{t}$.
- The final state is $\overline{t'} \lfloor \rfloor$, where $\overline{t'}$ is the original tokens rearranged in RPN.

1. $\overline{t'} \lceil \overline{\odot} \rceil a \ \overline{t} \quad \rightarrow \quad \overline{t'} \ a \lfloor \overline{\odot} \rfloor \overline{t}$
2. $\overline{t'} \lceil \overline{\odot} \rceil \odot_i \ \overline{t} \quad \rightarrow \quad \overline{t'} \lceil \overline{\odot} \ \odot_i \rceil \overline{t}$
3. $\overline{t'} \lfloor \overline{\odot} \ {}_?\odot_i \rfloor \ {}_j\odot_? \ \overline{t} \quad \rightarrow \quad \overline{t'} \ {}_?\odot_i \lfloor \overline{\odot} \rfloor \ {}_j\odot_? \ \overline{t}$ if $i < j$
4. $\overline{t'} \lfloor \overline{\odot} \ {}_?\odot_i \rfloor \ {}_j\odot_k \ \overline{t} \quad \rightarrow \quad \overline{t'} \lceil \overline{\odot} \ {}_?\odot_i \ {}_j\odot_k \rceil \overline{t}$ if $i > j$
5. $\overline{t'} \lfloor \overline{\odot} \ {}_?\odot_i \rfloor \ {}_j\odot \ \overline{t} \quad \rightarrow \quad \overline{t'} \ {}_j\odot \lfloor \overline{\odot} \ {}_?\odot_i \rfloor \overline{t}$ if $i > j$
6. $\overline{t'} \lfloor \rfloor \ {}_i\odot_j \ \overline{t} \quad \rightarrow \quad \overline{t'} \lceil {}_i\odot_j \rceil \overline{t}$
7. $\overline{t'} \lfloor \rfloor \ {}_i\odot \ \overline{t} \quad \rightarrow \quad \overline{t'} \ {}_i\odot \lfloor \rfloor \overline{t}$
8. $\overline{t'} \lfloor \overline{\odot} \ \odot \rfloor \quad \rightarrow \quad \overline{t'} \ \odot \lfloor \overline{\odot} \rfloor$

1.3 Derivations of some of the rules:

Rule 2. A prefix should act as if it were an atom followed by an infix operator with maximally low (tight) left precedence. Thus:

$$
\begin{array}{ll}
 & \overline{t'}\ \lceil\ \overline{\odot}\ \rceil\ \odot_i\ \overline{t} \\
\approx & \overline{t'}\ \lceil\ \overline{\odot}\ \rceil\ a\ {}_0\odot_i\ \overline{t} \\
\to_1 & \overline{t'}\ a\ \lfloor\ \overline{\odot}\ \rfloor\ {}_0\odot_i\ \overline{t} \\
\to_4 & \overline{t'}\ a\ \lfloor\ \overline{\odot}\ {}_0\odot_i\ \rfloor\ \overline{t} \\
\approx & \overline{t'}\ \lfloor\ \overline{\odot}\ \odot_i\ \rfloor\ \overline{t}
\end{array}
$$

Rule 5. A suffix should act as if it were an infix operator with maximally low (tight) right precedence, followed by an atom. Thus:

$$
\begin{array}{ll}
 & \overline{t'}\ \lfloor\ \overline{\odot}\ {}_?\odot_i\ \rfloor\ {}_j\odot\ \overline{t} \\
\approx & \overline{t'}\ \lfloor\ \overline{\odot}\ {}_?\odot_i\ \rfloor\ {}_j\odot_0\ a\ \overline{t} \\
\to_4 & \overline{t'}\ \lceil\ \overline{\odot}\ {}_?\odot_i\ {}_j\odot_0\ \rceil\ a\ \overline{t} \\
\to_1 & \overline{t'}\ a\ \lfloor\ \overline{\odot}\ {}_?\odot_i\ {}_j\odot_0\ \rfloor\ \overline{t} \\
\to_3 & \overline{t'}\ a\ {}_j\odot_0\ \lfloor\ \overline{\odot}\ {}_?\odot_i\ \rfloor\ \overline{t} \\
\approx & \overline{t'}\ {}_j\odot\ \lfloor\ \overline{\odot}\ {}_?\odot_i\ \rfloor\ \overline{t}
\end{array}
$$

Rule 6. The bottom of the operator stack should act as if it contains a maximally high precedence (weakly binding) operator. Thus:

$$\overline{t'}\ \lfloor\ \rfloor\ {}_i\odot_j\ \overline{t}\ \approx\ \overline{t'}\ \lfloor\ \odot_\infty\ \rfloor\ {}_i\odot_j\ \overline{t}\ \to_4\ \overline{t'}\ \lceil\ \odot_\infty\ {}_i\odot_j\ \rceil\ \overline{t}\ \approx\ \overline{t'}\ \lceil\ {}_i\odot_j\ \rceil\ \overline{t}$$

Rule 7. Similar to the previous rule.

$$\overline{t'}\ \lfloor\ \rfloor\ {}_i\odot\ \overline{t}\ \approx\ \overline{t'}\ \lfloor\ \odot_\infty\ \rfloor\ {}_i\odot\ \overline{t}\ \to_5\ \overline{t'}\ {}_i\odot\ \lfloor\ \odot_\infty\ \rfloor\ \overline{t}\ \approx\ \overline{t'}\ {}_i\odot\ \lfloor\ \rfloor\ \overline{t}$$

Rule 8. The end of the token stream should act as if it contains a maximally high precedence (weakly binding) operator. Thus:

$$\overline{t'}\ \lfloor\ \overline{\odot}\ \odot\ \rfloor\ \approx\ \overline{t'}\ \lfloor\ \overline{\odot}\ \odot\ \rfloor\ {}_\infty\odot\ \to_3\ \overline{t'}\ \odot\ \lfloor\ \overline{\odot}\ \rfloor\ {}_\infty\odot\ \approx\ \overline{t'}\ \odot\ \lfloor\ \overline{\odot}\ \rfloor$$

1.4 Error Cases

To handle potentially malformed inputs gracefully, introduce a special atom called M (for "missing"), and a special operator J (for "juxtaposition"). Insert M and J as required to make the expression well-formed. For example, 1+ would turn into $1 + M$, and 1 2 would turn into $1\ J\ 2$.

Using these special tokens, we can "fill out" the rest of the parsing cases, so that *every* expression parses.

$$
\begin{array}{lllll}
9. & \overline{t'}\ \lceil\ \overline{\odot}\ \rceil\ {}_i\odot_?\ \overline{t} & \to & \overline{t'}\ \lceil\ \overline{\odot}\ \rceil\ M\ {}_i\odot_?\ \overline{t} \\
10. & \overline{t'}\ \lceil\ \overline{\odot}\ \rceil & \to & \overline{t'}\ \lfloor\ \overline{\odot}\ \rfloor\ M \\
11. & \overline{t'}\ \lfloor\ \overline{\odot}\ \rfloor\ a\ \overline{t} & \to & \overline{t'}\ \lfloor\ \overline{\odot}\ \rfloor\ J\ a\ \overline{t} \\
12. & \overline{t'}\ \lfloor\ \overline{\odot}\ \rfloor\ \odot_i\ \overline{t} & \to & \overline{t'}\ \lfloor\ \overline{\odot}\ \rfloor\ J\ \odot_i\ \overline{t}
\end{array}
$$

(You can check that rules 1-12 now cover all cases; parsing never "gets stuck".)