$$
\begin{array}{rcl}
e & := & atom \\
 & & \mid (\ e_0\ e_1\ \ldots\ e_n\ ) \\
 & & \mid \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \\
 & & \mid \texttt{let}\ v_1 = e_1\ \ldots\ v_n = e_n\ \texttt{in}\ e_0 \\
 & & \mid \lambda v_1 v_2 \ldots v_n . e_0 \\
 & & \mid\ <\ e_1\ \ldots\ e_n\ > \\
 & & \mid \texttt{define}\ f_1 = e_1\ \ldots\ f_m = e_m\ \texttt{in}\ e_0 \\
 & & \\
atom & := & num \mid string \mid var \mid bool \mid ops \\
 & & \\
num & := & digit \mid num\ digit \mid\ \sim num \\
digit & := & 0 \mid 1 \mid \ldots \mid 9 \\
 & & \\
string & := & `str' \\
str & := & char \mid str\ char \\
char & := & a \mid \ldots \mid z \mid \_ \mid 0 \mid \ldots \mid 9 \\
 & & \\
var & := & letter \mid letter\ str \\
letter & := & u \mid v \mid \ldots \mid z \\
 & & \\
bool & := & true \mid false \\
 & & \\
ops & := & num\_ops \mid rel\_ops \mid log\_ops \mid list\_ops \\
 & & \\
num\_ops & := & + \mid - \mid * \mid / \\
 & & \\
rel\_ops & := & gt \mid lt \mid eq \mid neq \mid \ldots \\
 & & \\
log\_ops & := & not \mid and \mid or \mid \ldots \\
 & & \\
list\_ops & := & first \mid rest \mid append \mid empty \mid \ldots
\end{array}
$$

EVAL

- definiert eine operationelle Semantik auf Termen | Ausdrücken von FL
  → semantische Äquivalenz | Bedeutungsgleichheit von Termen
  → nicht die Bedeutung selbst

- schreibt lediglich die zu beachtenden kausalen Abhängigkeiten bei der Auswertung von Termen, nicht aber eine spezielle sequentielle Reihenfolge vor
  → da syntaktisch vollkommen unabhängig voneinander, können die Teilterme einer Applikation in beliebiger Reihenfolge | müssen aber vor Auswertung der Applikation selbst ausgewertet werden

- wird rekursiv von aussen nach innen vor Teilterme getrieben | Auswertung erfolgt von innen nach aussen
  → überall dort wo EVALs zum Verschwinden gebracht werden können

- impliziert die Definition der auf Terme | Teilterme anzuwendenden Transformationsregeln (mehr oder weniger direkt)

Offene Probleme

- rekursive Konstruktion von Termen durch Ersetzen von Vorkommen von e in allen syntaktischen Positionen durch beliebige legitime (korrekt konstruierte) Terme
  $\rightarrow$ alle syntaktisch korrekten Terme können sowohl als Operatoren als auch als Operanden von Applikationen auftreten
  $\rightarrow$ insbesondere Variablen | Abstraktionen | let–Terme define–Terme in Operandenpositionen relativ zu Abstraktionen in Operatorpositionen

- die Anzahl der Operanden einer Applikation muss nicht mit der Stelligkeit der primitiven Funktion | Abstraktion in Operatorposition übereinstimmen

- bei Anwendungen von Abstraktionen ist die Substitution von Vorkommen formaler Parameter nicht an bestimmte Operandenterme gebunden
  $\rightarrow$ alle syntaktisch korrekt konstruierten Terme sind zugelassen

- Anwendungen primitiver Funktionen | Operatoren verlangen Operanden (Argumente), deren Werte im Definitionsbereich der Operatoren liegen
  $\rightarrow$ ansonsten sind Applikationen nicht auswertbar $\rightarrow$ konstante Terme