Scheme Syntax in BNF

The following Backus-Naur Form (BNF) grammars describe the main constructs of Scheme[1] as well as some of the syntactic concepts needed in a rewriting "Substitution Model" for Scheme, namely, *syntactic values* and *control contexts*.

In these grammars, superscript "*" indicates zero or more occurrences of a grammatical phrase, and superscript "$^+$" indicates one or more occurrences.

1 The Functional Kernel

$$
\begin{aligned}
\langle\text{expression}\rangle &::= \langle\text{self-evaluating}\rangle \mid \langle\text{symbol}\rangle \mid \langle\text{nil}\rangle \mid \langle\text{variable}\rangle \\
&\quad\mid \langle\text{procedure}\rangle \mid \langle\text{let form}\rangle \mid \langle\text{if}\rangle \mid \langle\text{combination}\rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle\text{self-evaluating}\rangle &::= \langle\text{numeral}\rangle \mid \langle\text{boolean}\rangle \mid \langle\text{string}\rangle\ \ldots \\
\langle\text{numeral}\rangle &::= \texttt{0} \mid \texttt{-1} \mid \texttt{3.14159} \mid \ldots \\
\langle\text{boolean}\rangle &::= \texttt{\#t} \mid \texttt{\#f} \\
\langle\text{string}\rangle &::= \texttt{"hello there"} \mid \ldots
\end{aligned}
$$

$$
\begin{aligned}
\langle\text{symbol}\rangle &::= \texttt{(quote}\ \langle\text{identifier}\rangle\texttt{)} \\
\langle\text{identifier}\rangle &::= \text{identifiers which are not } \langle\text{self-evaluating}\rangle \\
\langle\text{nil}\rangle &::= \texttt{'()} \\
\langle\text{variable}\rangle &::= \text{identifiers which are neither } \langle\text{self-evaluating}\rangle, \\
&\qquad \langle\text{procedure identifier}\rangle,\ \text{nor } \langle\text{keyword}\rangle
\end{aligned}
$$

$$
\langle\text{keyword}\rangle ::= \texttt{quote} \mid \texttt{lambda} \mid \langle\text{let keyword}\rangle \mid \texttt{if}
$$

$$
\begin{aligned}
\langle\text{procedure}\rangle &::= \langle\text{procedure identifier}\rangle \mid \langle\text{lambda expression}\rangle \\
\langle\text{procedure identifier}\rangle &::= \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{=} \mid \texttt{<} \mid \texttt{atan} \mid \texttt{call/cc} \mid \ldots \\
&\quad\mid \texttt{number?} \mid \texttt{symbol?} \mid \texttt{procedure?} \mid \texttt{eq?} \mid \ldots \\
&\quad\mid \texttt{cons} \mid \texttt{car} \mid \texttt{cdr} \mid \texttt{list} \mid \texttt{apply} \mid \texttt{null?} \mid \texttt{pair?} \\
&\quad\text{(Note: No "side-effect" procedures such as } \texttt{display}, \texttt{set-car!}.) \\
\langle\text{lambda expression}\rangle &::= \texttt{(lambda (}\ \langle\text{formals}\rangle\ \texttt{)}\ \langle\text{expression}\rangle\texttt{)} \\
&\quad\mid \texttt{(lambda}\ \langle\text{variable}\rangle\ \langle\text{expression}\rangle\texttt{)} \\
\langle\text{formals}\rangle &::= \langle\text{variable}\rangle^* \qquad \text{(Note: all } \langle\text{variable}\rangle\text{'s must be distinct.)}
\end{aligned}
$$

[1]For the official, full Scheme grammar, see the Revised[5] Scheme Manual available on the web at:

`http://www-swiss.ai.mit.edu/~jaffer/r5rs_toc.html`

⟨let form⟩ ::= `(` ⟨let keyword⟩ `(` ⟨binding⟩* `)` ⟨expression⟩`)`
(Note: all `let`form-bound variables must be distinct.)
⟨let keyword⟩ ::= `letrec`
⟨binding⟩ ::= `(`⟨variable⟩ ⟨init⟩ `)`
⟨init⟩ ::= ⟨expression⟩

⟨if⟩ := `(if` ⟨test⟩ ⟨consequent⟩ ⟨alternative⟩ `)`
⟨test⟩ ::= ⟨expression⟩
⟨consequent⟩ ::= ⟨expression⟩
⟨alternative⟩ ::= ⟨expression⟩

⟨combination⟩ ::= `(` ⟨operator⟩ ⟨operand⟩* `)`
⟨operator⟩ ::= ⟨expression⟩
⟨operand⟩ ::= ⟨expression⟩

2 The Full Kernel

⟨define⟩ ::= `(define` ⟨variable⟩⟨expression⟩`)`

⟨expression⟩ ::= ... | ⟨begin⟩ | ⟨assignment⟩

⟨keyword⟩ ::= ... | `define` | `begin` | `set!`

⟨procedure identifier⟩ ::= ... | `set-car!` | `set-cdr!` | `display` | `newline` | ...

⟨begin⟩ ::= `(begin` ⟨expression⟩$^+$`)`

⟨assignment⟩ ::= `(set!` ⟨variable⟩ ⟨expression⟩`)`

3 Derived Syntax

$$
\begin{aligned}
\langle\text{define}\rangle &::= \ldots \mid \texttt{(define (}\langle\text{variable}\rangle\,\langle\text{formals}\rangle\texttt{)}\;\langle\text{body}\rangle\texttt{)} \\
\langle\text{body}\rangle &::= \langle\text{internal defines}\rangle\,\langle\text{expression}\rangle^{+} \\
\langle\text{internal defines}\rangle &::= \langle\text{define}\rangle^{*} \qquad \text{(Note: all defined variables must be distinct.)} \\
\\
\langle\text{expression}\rangle &::= \ldots \mid \langle\text{cond}\rangle \mid \langle\text{and}\rangle \mid \langle\text{or}\rangle \mid \langle\text{quoted}\rangle \\
\\
\langle\text{keyword}\rangle &::= \ \ldots \mid \texttt{cond} \mid \texttt{else} \mid \texttt{and} \mid \texttt{or} \mid \texttt{quasiquote} \\
\\
\langle\text{let form}\rangle &::= \ldots \mid \texttt{(}\,\langle\text{let keyword}\rangle\,\texttt{(}\,\langle\text{binding}\rangle^{*}\,\texttt{)}\;\langle\text{body}\rangle\texttt{)} \\
\langle\text{let keyword}\rangle &::= \ldots \mid \texttt{let} \mid \texttt{let*} \\
\\
\langle\text{lambda expression}\rangle &::= \ldots \mid \texttt{(lambda (}\langle\text{formals}\rangle\texttt{)}\;\langle\text{body}\rangle\texttt{)} \\
&\quad\ \mid \texttt{(lambda}\;\langle\text{variable}\rangle\,\langle\text{body}\rangle\texttt{)} \\
\\
\langle\text{cond}\rangle &::= \texttt{(cond}\;\langle\text{clause}\rangle\,\langle\text{clause}\rangle^{*}\,\texttt{)} \\
\langle\text{clause}\rangle &::= \texttt{(}\,\langle\text{test}\rangle\,\langle\text{expression}\rangle^{*}\texttt{)} \mid \texttt{(else}\;\langle\text{expression}\rangle^{+}\texttt{)} \\
\langle\text{and}\rangle &::= \texttt{(and}\;\langle\text{expression}\rangle^{*}\texttt{)} \\
\langle\text{or}\rangle &::= \texttt{(or}\;\langle\text{expression}\rangle^{*}\texttt{)} \\
\\
\langle\text{quoted}\rangle &::= \texttt{(quote}\;\langle\text{s-expression}\rangle\,\texttt{)} \mid \texttt{(quasiquote}\;\langle\text{s-expression}\rangle\,\texttt{)} \\
\langle\text{s-expression}\rangle &::= \langle\text{identifier}\rangle \mid \langle\text{self-evaluating}\rangle \mid \texttt{(}\,\langle\text{s-expression}\rangle^{*}\,\texttt{)}
\end{aligned}
$$

4 Syntactic Values

$$
\begin{aligned}
\langle\text{syntactic value}\rangle &::= \langle\text{immediate value}\rangle \mid \langle\text{pair value}\rangle \mid \langle\text{list value}\rangle \\
\langle\text{immediate value}\rangle &::= \langle\text{self-evaluating}\rangle \mid \langle\text{symbol}\rangle \mid \langle\text{procedure}\rangle \\
\langle\text{pair value}\rangle &::= \texttt{(cons}\;\langle\text{syntactic value}\rangle\,\langle\text{pair value}\rangle\texttt{)} \\
&\quad\ \mid \texttt{(cons}\;\langle\text{syntactic value}\rangle\,\langle\text{immediate value}\rangle\texttt{)} \\
\langle\text{list value}\rangle &::= \langle\text{nil}\rangle \mid \texttt{(list}\;\langle\text{syntactic value}\rangle^{+}\texttt{)}
\end{aligned}
$$

5 Control Contexts for Kernel Scheme

$$
\begin{array}{rcl}
\langle\text{control context}\rangle & ::= & \langle\text{hole}\rangle \\
 & | & \texttt{(if } \langle\text{control context}\rangle\ \langle\text{expression}\rangle\ \langle\text{expression}\rangle\texttt{)} \\
 & | & \texttt{(begin } \langle\text{control context}\rangle\ \langle\text{expression}\rangle^*\texttt{)} \\
 & | & \texttt{(set! } \langle\text{variable}\rangle\ \langle\text{control context}\rangle\texttt{)} \\
 & | & \texttt{(} \langle\text{let keyword}\rangle \\
 & & \quad \texttt{(} \langle\text{value binding}\rangle^*\ \texttt{(}\langle\text{variable}\rangle\ \langle\text{control context}\rangle\texttt{)}\ \langle\text{binding}\rangle^*\ \texttt{)} \\
 & & \qquad \langle\text{expression}\rangle\texttt{)} \\
 & | & \texttt{(}\langle\text{syntactic value}\rangle^*\ \langle\text{control context}\rangle\ \langle\text{expression}\rangle^*\texttt{)} \\
\langle\text{hole}\rangle & ::= & \texttt{< >} \\
\langle\text{value binding}\rangle & ::= & \texttt{(}\langle\text{variable}\rangle\ \langle\text{syntactic value}\rangle\texttt{)}
\end{array}
$$

6 Immediate Redexes for Kernel Scheme

$$
\begin{array}{rcl}
\langle\text{immediate redex}\rangle & ::= & \langle\text{variable}\rangle \\
 & | & \texttt{(if } \langle\text{syntactic value}\rangle\ \langle\text{expression}\rangle\ \langle\text{expression}\rangle\texttt{)} \\
 & | & \texttt{(begin } \langle\text{expression}\rangle\texttt{)} \\
 & | & \texttt{(begin } \langle\text{syntactic value}\rangle\ \langle\text{expression}\rangle^*\texttt{)} \\
 & | & \texttt{(set! } \langle\text{variable}\rangle\ \langle\text{syntactic value}\rangle\texttt{)} \\
 & | & \texttt{(} \langle\text{let keyword}\rangle\ \texttt{(} \langle\text{value binding}\rangle^*\ \texttt{)}\langle\text{expression}\rangle\texttt{)} \\
 & | & \texttt{(}\langle\text{syntactic value}\rangle^+\texttt{)} \\
 & | & \texttt{(define } \langle\text{variable}\rangle\ \langle\text{expression}\rangle\texttt{)}
\end{array}
$$

Lemma. (**Unique Matching**) If a Scheme expression, E, is not a syntactic value, then there is a unique control context, C, and a unique immediate redex, R, such that $E = C[\texttt{< >} := R]$. Moreover, if B is a sequence of value bindings (of distinct variables), then there is at most one Scheme Substitution Model rewrite rule whose pattern matches $\texttt{(letrec (}B\texttt{)}\ E\texttt{)}$, and if there is such a rewrite rule, its match is unique.