1 Syntax of MLB

For MLB there are further reserved words, identifier classes and derived forms. There are no further special constants; comments and lexical analysis are as for the Core and Modules. The derived forms appear in Appendix A.

1.1 Reserved Words

The following are the additional reserved words used in MLB.

\texttt{bas} basis

Note that many of the reserved words from the Core and Modules are not used by the grammar of MLB. However, as the grammar includes identifiers from the grammars of the Core and Modules, it is useful to consider the reserved words from the Core and Modules to be reserved in MLB as well.

1.2 Identifiers

The additional identifier class for MLB are \texttt{BasId} (basis identifiers). Basis identifiers must be alphanumeric, not starting with a prime. The class of each identifier occurrence is determined by the grammatical rules which follow. Henceforth, therefore, we consider all identifier classes to be disjoint.

1.3 Infixed operators

The grammar of MLB does not directly admit fixity directives. However, the static and dynamic semantics for MLB will import source files that must be parsed in the scope of fixity directives and that may introduce additional fixity directives into scope. Figure 1 formalizes the Definition’s notion of \textit{infix status} as a \textit{fixity environment}.

\[
\text{InfixStatus} = \{\text{nonfix}\} \cup \bigcup_{d \in \{0, \ldots, 9\}} \{\text{infix } d, \text{infixr } d\}
\]

\[
FE \in \text{FixEnv} = \text{VId} \xrightarrow{\text{fn}} \text{InfixStatus}
\]

Figure 1: Fixity Environment
1.4 Grammar for MLB

The phrase classes for MLB are shown in Figure 2. We use the variable base\textsl{exp} to range over BasExp, etc.

- BasExp \hspace{1em} basis expressions
- BasDec \hspace{1em} basis-level declaration
- BasBind \hspace{1em} basis bindings
- BStrBind \hspace{1em} (basis) structure bindings
- BSigBind \hspace{1em} (basis) signature bindings
- BFunBind \hspace{1em} (basis) functor bindings

Figure 2: MLB Phrase Classes

The conventions adopted in presenting the grammatical rules for MLB are the same as for the Core and Modules. The grammatical rules are shown in Figure 3.

Figure 3: Grammar: Basis Expressions, Declarations, and Bindings

1.5 Syntactic Restrictions

- No binding basbind may bind the same identifier twice.
- No binding bstrbind, bsigbind or bfunbind may bind the same identifier twice.
- MLB may not be cyclic; i.e., successively replacing path.mlb with its parsed BasDec must terminate.

1.6 Parsing

The static and dynamic semantics for MLB will interpret path.sml as a parsed TopDec and path.mlb as a parsed BasDec. Parsing a TopDec takes a fixity environment as input and returns a fixity environment as output; the output fixity environment corresponds to fixity directives introduced by and whose scope is not limited by the parsed TopDec.

Paths and parsers are given in Figure 4. A (fixed) Parser $\mathcal{P}$ provides the interpretation of path.sml and path.mlb imports. For a file extension .ext, path.ext denotes either an absolute path or a relative path
\[
\text{path.sml} \in \text{SourcePath} \\
\text{path.mlb} \in \text{MLBasisPath}
\]
\[
\mathcal{P} \in \text{Parser} = ((\text{FixEnv} \times \text{SourcePath}) \xrightarrow{\text{fin}} (\text{FixEnv} \times \text{TopDec})) \times (\text{MLBasisPath} \xrightarrow{\text{fin}} \text{BasDec})
\]

Figure 4: Parser

(relative to the BasDec being parsed) to a file in the underlying file system. Paths that denote the same file in the underlying file system are considered equal, though they may have distinct textual representations. An implementation may allow additional extensions (e.g., `.ML`, `.fun`, `.sig`) in elements of SourcePath. An implementation may additionally allow path variables to appear in paths. Parser could be refined by a current working directory, to formally specify the interpretation of relative paths, and an path map, to formally specify the interpretation of path variables, but the above suffices for the development in the following sections.

# 2 Static Semantics for MLB

## 2.1 Semantic Objects

The simple objects for the MLB static semantics are exactly as for Modules. The compound objects are those for Modules, augmented by those in Figure 5. The operations of projection, injection and modification

\[
\begin{align*}
BE & \in \text{BasEnv} = \text{BasId} \xrightarrow{\text{fin}} \text{MBasis} \\
M \lor FE, BE, B & \in \text{MBasis} = \text{FixEnv} \times \text{BasEnv} \times \text{Basis} \\
\Psi & \in \text{BasCache} = \text{MLBasisPath} \xrightarrow{\text{fin}} \text{MBasis}
\end{align*}
\]

Figure 5: Compound Semantic Objects

are as for Modules.

## 2.2 Inference Rules

As for the Core and for Modules, the rules for MLB static semantics allow sentences of the form

\[
A \vdash \text{phrase} \rightarrow A'
\]

to be inferred. Some hypotheses in rules are not of this form; they are called side-conditions. The convention for options is as in the Core and Modules semantics.

### Basis Expressions

\[
\begin{align*}
M, \Psi \vdash \text{basdec} & \rightarrow M', \Psi' \\
M, \Psi \vdash \text{bas} \text{ basdec end} & \rightarrow M', \Psi' \\
M(basid) & = M' \\
M, \Psi \vdash \text{basid} & \rightarrow M', \Psi \\
M, \Psi \vdash \text{let basdec in basexp end} & \rightarrow M_2, \Psi_2
\end{align*}
\]

(1)
Comments:
(3) The use of $\oplus$, here and elsewhere, ensures that the type names generated by the first sub-phrase are distinct from the names generated by the second sub-phrase.

Basis-level Declarations

\[
M, \Psi \vdash \text{basbind} \rightarrow BE', \Psi'
\]

\[
M, \Psi \vdash \text{basdec} \rightarrow BE' \text{ in } \text{MBasis, } \Psi'
\]

\[
M, \Psi \vdash \text{basdec}_1 \rightarrow M_1, \Psi_1 \quad M \oplus M_1, \Psi_1 \vdash \text{basdec}_2 \rightarrow M_2, \Psi_2
\]

\[
M, \Psi \vdash \text{local} \, \text{basdec}_1 \text{ in } \text{basdec}_2 \text{ end} \rightarrow M_2, \Psi_2
\]

\[
M, \Psi \vdash \text{open} \, \text{basid}_1 \cdots \text{basid}_n \rightarrow M_1 \oplus \cdots \oplus M_n, \Psi
\]

(4) \[
M, \Psi \vdash \text{basdec}_1 \rightarrow M_1, \Psi_1 \quad M \oplus M_1, \Psi_1 \vdash \text{basdec}_2 \rightarrow M_2, \Psi_2
\]

\[
M, \Psi \vdash \text{local} \, \text{basdec}_1 \text{ in } \text{basdec}_2 \text{ end} \rightarrow M_2, \Psi_2
\]

(5) \[
M, \Psi \vdash \text{basdec}_1 \rightarrow M_1, \Psi_1 \quad M \oplus M_1, \Psi_1 \vdash \text{basdec}_2 \rightarrow M_2, \Psi_2
\]

\[
M, \Psi \vdash \text{basdec}_1 \langle ; \rangle \text{basdec}_2 \rightarrow M_1 \oplus M_2, \Psi
\]

(6) \[
M, \Psi \vdash \text{open} \, \text{basid}_1 \cdots \text{basid}_n \rightarrow M_1 \oplus \cdots \oplus M_n, \Psi
\]

Comments:
(12) Note the use of the Definition’s $B \vdash \text{topdec} \Rightarrow B'$.

Basis Bindings

\[
M, \Psi \vdash \text{basbind} \rightarrow BE', \Psi'
\]

(15) \[
M, \Psi \vdash \text{baseexp} \rightarrow M', \Psi' \quad (M + \text{tynames} M', \Psi' \vdash \text{basbind} \rightarrow BE'', \Psi'')
\]

(16) \[
B(\text{strid}_2) = E \quad (B + \text{tynames} E \vdash \text{bstrbind} \rightarrow SE)
\]

Basis Bindings

\[
B \vdash \text{bstrbind} \rightarrow SE
\]
Comments:

(16) Note that $bstrbind \subset strbind$. Hence, this rule can be derived from the Definition’s $B \vdash strbind \Rightarrow SE$.

(Basis) Signature Bindings

$$B \vdash bsigbind \rightarrow G$$

$$B(sigid_2) = \Sigma \cdot \Sigma = (T)E \cdot T \cap (T \ of \ B) = \emptyset$$

$$T = \text{tynames } E \setminus (T \ of \ B) \cdot \langle B \vdash bsigbind \rightarrow G \rangle$$

$$B \vdash sigid_1 = sigid_2 \ (\text{and } bsigbind) \rightarrow \{sigid_1 \mapsto \Sigma\}?$$

(17)

Comments:

(17) Note that $bsigbind \subset sigbind$. Hence, this rule can be derived from the Definition’s $B \vdash sigbind \Rightarrow G$.

As such, the following comment from the Definition applies:

The bound names of $B(sigid_2)$ can always be renamed to satisfy $T \cap (T \ of \ B) = \emptyset$, if necessary.

(Basis) Functor Bindings

$$B \vdash bfunbind \rightarrow F$$

$$B(funid_2) = \Phi \cdot \Phi = (T)(E, (T')E') \cdot T \cap (T \ of \ B) = \emptyset$$

$$T' = \text{tynames } E' \setminus ((T \ of \ B) \cup T) \cdot \langle B \vdash bfunbind \rightarrow F \rangle$$

$$B \vdash funid_1 = funid_2 \ (\text{and } bfunbind) \rightarrow \{funid_1 \mapsto \Phi\}?$$

(18)

3 Dynamic Semantics for MLB

3.1 Reduced Syntax

The syntax of MLB is unchanged for the purposes of the dynamic semantics for MLB. However, the Parser $P$ returns a $\text{topdec}$ in the reduced syntax of Modules.

3.2 Compound Objects

The compound objects for the MLB dynamic semantics, extra to those for the Modules dynamic semantics, are shown in Figure 6.

$$BE \in \text{BasEnv} = \text{BasId} \rightarrow \text{MBasis}$$

$$M \ or \ FE, BE, B \in \text{MBasis} = \text{FixEnv} \times \text{BasEnv} \times \text{Basis}$$

$$\Psi \in \text{BasCache} = \text{MLBasisPath} \rightarrow \text{MBasis}$$

Figure 6: Compound Semantic Objects

3.3 Inference Rules

The semantic rules allow sentences of the form

$$s, A \vdash phrase \rightarrow A', s'$$

to be inferred, where $s, s'$ are the states before and after the evaluation represented by the sentence. Some hypotheses in rules are not of this form; they are called $\text{side-conditions}$. The convention for options is as in the Core and Modules semantics.
The state and exception conventions are adopted as in the Core and Modules dynamic semantics. However, it can be shown that the only MLB phrases whose evaluation may cause a side-effect or generate an exception packet are of the form \( \text{basexp}, \text{basdec} \) or \( \text{basbind} \).

### Basis Expressions

\[
M, \Psi \vdash \text{basdec} \rightarrow M', \Psi'/p
\]

\[
M, \Psi \vdash \text{bas} \text{basdec} \rightarrow M', \Psi
\]

\[
M(\text{basid}) = M'
\]

\[
M, \Psi \vdash \text{basdec} \rightarrow M_1, \Psi_1 \quad M + M_1, \Psi_1 \vdash \text{basexp} \rightarrow M_2, \Psi_2
\]

\[
M, \Psi \vdash \text{let basdec in basexp end} \rightarrow M_2, \Psi_2
\]

### Basis-level Declarations

\[
M, \Psi \vdash \text{basdec} \rightarrow BE', \Psi'
\]

\[
M, \Psi \vdash \text{basis basdec} \rightarrow BE' \text{ in MBasis, } \Psi'
\]

\[
M, \Psi \vdash \text{local basdec in basdec end} \rightarrow M_2, \Psi_2
\]

\[
M, \Psi \vdash \text{open basid}_1 \cdots \text{basid}_n \rightarrow M_1 + \cdots + M_n, \Psi
\]

\[
\text{Inter } (B \text{ of } M) \vdash \text{bsigbind} \rightarrow G
\]

\[
M, \Psi \vdash \text{signature bsigbind} \rightarrow G \text{ in MBasis, } \Psi
\]

\[
M, \Psi \vdash \text{functor bfunbind} \rightarrow F \text{ in MBasis, } \Psi
\]

\[
M, \Psi \vdash \{ \} \rightarrow \{} \text{ in MBasis, } \Psi
\]

\[
M, \Psi \vdash \text{basdec}_1 \rightarrow M_1, \Psi_1 \quad M + M_1, \Psi_1 \vdash \text{basdec}_2 \rightarrow M_2, \Psi_2
\]

\[
\mathcal{P}(\text{FE of } M, \text{path.sml}) = (\text{FE}', \text{topdec}) \quad B \text{ of } M \vdash \text{topdec \Rightarrow B'}
\]

\[
\Psi(\text{path.ml}) = M'
\]

\[
M, \Psi \vdash \text{path.ml} \rightarrow M', \Psi'
\]

\[
\text{path.ml} \notin \text{Dom } \Psi \quad \mathcal{P}(\text{path.ml}) = \text{basdec} \{ \} \text{ in MBasis, } \Psi \vdash \text{basdec} \rightarrow M', \Psi'
\]

\[
M, \Psi \vdash \text{path.ml} \rightarrow M', \Psi' + \{ \text{path.ml} \mapsto M' \}
\]
Comments:

(30) Note the use of the Definition’s $B \vdash \text{topdec} \Rightarrow B'$.

**Basis Bindings**

\[
M, \Psi \vdash \text{basexp} \to M', \Psi' \quad \text{(M,} \Psi \vdash \text{basbind} \to BE'', \Psi'')
\]

\[
M, \Psi \vdash \text{basid} = \text{basexp (and basbind)} \to \{\text{basid} \Rightarrow M'\}(+BE'', \Psi')
\]

(33)

**(Basis) Structure Bindings**

\[
B \vdash \text{bstrbind} \to SE
\]

\[
B(\text{strid}_2) = E \quad \langle B \vdash \text{bstrbind} \to SE \rangle
\]

\[
B \vdash \text{strid}_1 = \text{strid}_2 (\text{and bstrbind}) \to \{\text{strid}_1 \Rightarrow E\}(+SE)
\]

(34)

**Comments:**

(34) Note that $\text{bstrbind} \subset \text{strbind}$. Hence, this rule can be derived from the Definition’s $B \vdash \text{strbind} \Rightarrow SE/p$, noting that the derivation may neither cause a side-effect nor generate an exception packet.

**(Basis) Signature Bindings**

\[
\text{IB} \vdash \text{bsigbind} \to G
\]

\[
\text{IB}(\text{sigid}_2) = I \quad \langle \text{IB} \vdash \text{bsigbind} \to G \rangle
\]

\[
\text{IB} \vdash \text{sigid}_1 = \text{sigid}_2 (\text{and bsigbind}) \to \{\text{sigid}_1 \Rightarrow I\}(+G)
\]

(35)

**Comments:**

(35) Note that $\text{bsigbind} \subset \text{sigbind}$. Hence, this rule can be derived from the Definition’s $\text{IB} \vdash \text{sigbind} \Rightarrow G$, noting that the derivation may neither cause a side-effect nor generate an exception packet.

**(Basis) Functor Bindings**

\[
B \vdash \text{bfunbind} \to F
\]

\[
B(\text{funid}_2) = (\text{strid : I, strexp, B}) \quad \langle B \vdash \text{bfunbind} \to F \rangle
\]

\[
B \vdash \text{funid}_1 = \text{funid}_2 (\text{and bfunbind}) \to \{\text{funid}_1 \Rightarrow (\text{strid : I, strexp, B})\}(+F)
\]

(36)

**A Derived Forms**

Figure 7 shows derived forms for structure, signature, and functor bindings in MLB. These derived forms are a useful shorthand for specifying import and export filters.

**References**

### Derived Form Equivalent Form

<table>
<thead>
<tr>
<th>(Basis) Structure Bindings $bstbind$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$strid \langle \text{and } bstbind \rangle$</td>
<td>$strid = strid \langle \text{and } bstbind \rangle$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(Basis) Signature Bindings $bsigbind$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$sigid \langle \text{and } bsigbind \rangle$</td>
<td>$sigid = sigid \langle \text{and } bsigbind \rangle$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(Basis) Functor Bindings $bfunbind$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$funid \langle \text{and } bfunbind \rangle$</td>
<td>$funid = funid \langle \text{and } bfunbind \rangle$</td>
</tr>
</tbody>
</table>

Figure 7: Derived forms of (Basis) Structure, Signature, and Functor Bindings