Formal Specification of the ML Basis system

Henry Cejtin, Matthew Fluet, Suresh Jagannathan, and Stephen Weeks

January 28, 2005

This document formally specifies the ML Basis system (MLB) in MLton used to program in the large. The system has been designed to be a natural extension of Standard ML, and the specification is given in the style of The Definition of Standard ML [MTHM97] (henceforeth, the Definition). This section adopts (often silently) abbreviations, conventions, definitions, and notation from the Definition.

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.

`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 BasId (basis identifiers). Basis identifiers must be alphanumeric, not starting with a prime. The class of each identifier occurence 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 *infix status* as a *fixity environment*.

$$\begin{array}{rcl} & & \mathrm{InfixStatus} = \{\texttt{nonfix}\} \cup \bigcup_{d \in \{0,\ldots,9\}} \{\texttt{infix}\ d, \texttt{infixr}\ d\} \\ FE & \in & \mathrm{FixEnv} = \mathrm{VId} \xrightarrow{\mathrm{fin}} \mathrm{InfixStatus} \end{array}$$

Figure 1: Fixity Environment

BasExp	basis expressions
BasDec	basis-level declaration
BasBind	basis bindings
BStrBind	(basis) structure bindings
BSigBind	(basis) signature bindings
BFunBind	(basis) functor bindings

Figure 2: MLB Phrase Classes

1.4 Grammar for MLB

The phrase classes for MLB are shown in Figure 2. We use the variable *basexp* to range over BasExp, etc. The conventions adopted in presenting the grammatical rulse for MLB are the same as for the Core and Modules. The grammatical rules are showin in Figure 3.

basexp	::=	`bas` *basdec* `end`	basic
		basid	basis identifier
		`let` *basdec* `in` *basexp* `end`	local declaration
basdec	::=	`basis` *basbind*	basis
		`local` $basdec_1$ `in` $basdec_2$ `end`	local
		`open` $basid_1 \cdots basid_n$	open ($n \geq 1$)
		`structure` *bstrbind*	(basis) structure binding
		`signature` *bsigbind*	(basis) signature binding
		`functor` *bfunbind*	(basis) functor binding
			empty
		$basdec_1 \langle ; \rangle\ basdec_2$	sequential
		path.mlb	import ML basis
		path.sml	import source
basbind	::=	$basid = basexp$ ⟨`and` *basbind*⟩	
bstrbind	::=	$strid_1 = strid_2$ ⟨`and` *bstrbind*⟩	
bsigbind	::=	$sigid_1 = sigid_2$ ⟨`and` *bsigbind*⟩	
bfunbind	::=	$funid_1 = funid_2$ ⟨`and` *bfunbind*⟩	

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 it's 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 (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.

$$\begin{array}{rcl} & \textsf{path.sml} & \in \ \text{SourcePath} \\ & \textsf{path.mlb} & \in \ \text{MLBasisPath} \\ \mathcal{P} & \in & \text{Parser} = ((\text{FixEnv} \times \text{SourcePath}) \xrightarrow{\text{fin}} (\text{FixEnv} \times \text{TopDec})) \times (\text{MLBasisPath} \xrightarrow{\text{fin}} \text{BasDec}) \end{array}$$

Figure 4: Parser

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

$$\begin{array}{rcl} BE & \in & \text{BasEnv} = \text{BasId} \xrightarrow{\text{fin}} \text{MBasis} \\ M \text{ or } FE, BE, B & \in & \text{MBasis} = \text{FixEnv} \times \text{BasEnv} \times \text{Basis} \\ \Psi & \in & \text{BasCache} = \text{MLBasisPath} \xrightarrow{\text{fin}} \text{MBasis} \end{array}$$

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

$$A \vdash \mathit{phrase} \longrightarrow A'$$

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 $\boxed{M, \Psi \vdash \mathit{basexp} \longrightarrow M', \Psi'}$

$$\frac{M, \Psi \vdash \mathit{basdec} \longrightarrow M', \Psi'}{M, \Psi \vdash \texttt{bas}\ \mathit{basdec}\ \texttt{end} \longrightarrow M', \Psi'} \tag{1}$$

$$\frac{M(\mathit{basid}) = M'}{M, \Psi \vdash \mathit{basid} \longrightarrow M', \Psi} \tag{2}$$

$$\frac{M, \Psi \vdash \mathit{basdec} \longrightarrow M_1, \Psi_1 \quad M \oplus M_1, \Psi_1 \vdash \mathit{basexp} \longrightarrow M_2, \Psi_2}{M, \Psi \vdash \texttt{let}\ \mathit{basdec}\ \texttt{in}\ \mathit{basexp}\ \texttt{end} \longrightarrow M_2, \Psi_2} \tag{3}$$

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

$$\boxed{M, \Psi \vdash basdec \longrightarrow M', \Psi'}$$

$$\frac{M, \Psi \vdash basbind \longrightarrow BE', \Psi'}{M, \Psi \vdash \textsf{basis}\ basbind \longrightarrow BE' \text{ in MBasis}, \Psi'} \tag{4}$$

$$\frac{M, \Psi \vdash basdec_1 \longrightarrow M_1, \Psi_1 \quad M \oplus M_1, \Psi_1 \vdash basdec_2 \longrightarrow M_2, \Psi_2}{M, \Psi \vdash \texttt{local}\ basdec_1\ \texttt{in}\ basdec_2\ \texttt{end} \longrightarrow M_2, \Psi_2} \tag{5}$$

$$\frac{M(basid_1) = M_1 \quad \cdots \quad M(basid_n) = M_n}{M, \Psi \vdash \texttt{open}\ basid_1 \cdots basid_n \longrightarrow M_1 \oplus \cdots \oplus M_n, \Psi} \tag{6}$$

$$\frac{B \text{ of } M \vdash bstrbind \longrightarrow SE}{M, \Psi \vdash \texttt{structure}\ bstrbind \longrightarrow SE \text{ in MBasis}, \Psi} \tag{7}$$

$$\frac{B \text{ of } M \vdash bsigbind \longrightarrow G}{M, \Psi \vdash \texttt{signature}\ bsigbind \longrightarrow G \text{ in MBasis}, \Psi} \tag{8}$$

$$\frac{B \text{ of } M \vdash bfunbind \longrightarrow F}{M, \Psi \vdash \texttt{functor}\ bfunbind \longrightarrow F \text{ in MBasis}, \Psi} \tag{9}$$

$$\frac{}{M, \Psi \vdash \quad \longrightarrow \{\} \text{ in MBasis}, \Psi} \tag{10}$$

$$\frac{M, \Psi \vdash basdec_1 \longrightarrow M_1, \Psi_1 \quad M \oplus M_1, \Psi_1 \vdash basdec_2 \longrightarrow M_2, \Psi_2}{M, \Psi \vdash basdec_1\ \langle ; \rangle\ basdec_2 \longrightarrow M_1 \oplus M_2, \Psi_2} \tag{11}$$

$$\frac{\mathcal{P}(FE \text{ of } M, \textsf{path.sml}) = (FE', topdec) \quad B \text{ of } M \vdash topdec \Rightarrow B'}{M, \Psi \vdash \textsf{path.sml} \longrightarrow (FE', \{\}, B'), \Psi} \tag{12}$$

$$\frac{\Psi(\textsf{path.mlb}) = M'}{M, \Psi \vdash \textsf{path.mlb} \longrightarrow M', \Psi} \tag{13}$$

$$\frac{\textsf{path.mlb} \notin \text{Dom } \Psi \quad \mathcal{P}(\textsf{path.mlb}) = basdec \quad \{\} \text{ in MBasis}, \Psi \vdash basdec \longrightarrow M', \Psi'}{M, \Psi \vdash \textsf{path.mlb} \longrightarrow M', \Psi' + \{\textsf{path.mlb} \mapsto M'\}} \tag{14}$$

Comments:

(12) Note the use of the Definition's $B \vdash topdec \Rightarrow B'$.

Basis Bindings

$$\boxed{M, \Psi \vdash basbind \longrightarrow BE', \Psi'}$$

$$\frac{M, \Psi \vdash basexp \longrightarrow M', \Psi' \quad \langle M + \text{tynames } M', \Psi' \vdash basbind \longrightarrow BE'', \Psi'' \rangle}{M, \Psi \vdash basid = basexp\ \langle \texttt{and}\ basbind \rangle \longrightarrow \{basid \mapsto M'\} \langle + BE'' \rangle, \Psi' \langle ' \rangle} \tag{15}$$

(Basis) Structure Bindings

$$\boxed{B \vdash bstrbind \longrightarrow SE}$$

$$\frac{B(strid_2) = E \quad \langle B + \text{tynames } E \vdash bstrbind \longrightarrow SE \rangle}{B \vdash strid_1 = strid_2\ \langle \texttt{and}\ bstrbind \rangle \longrightarrow \{strid_1 \mapsto E\} \langle + SE \rangle} \tag{16}$$

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 $\boxed{B \vdash bsigbind \longrightarrow G}$

$$\frac{\begin{array}{c} B(sigid_2) = \Sigma \quad \Sigma = (T)E \quad T \cap (T \text{ of } B) = \emptyset \\ T = \text{tynames } E \setminus (T \text{ of } B) \quad \langle B \vdash bsigbind \longrightarrow G \rangle \end{array}}{B \vdash sigid_1 = sigid_2 \ \langle \texttt{and}\ bsigbind \rangle \longrightarrow \{sigid_1 \mapsto \Sigma\} \langle + G \rangle} \tag{17}$$

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 $T \cap (T \text{ of } B) = \emptyset$, if necessary.

(Basis) Functor Bindings $\boxed{B \vdash bfunbind \longrightarrow F}$

$$\frac{\begin{array}{c} B(funid_2) = \Phi \quad \Phi = (T)(E, (T')E') \quad T \cap (T \text{ of } B) = \emptyset \\ T' = \text{tynames } E' \setminus ((T \text{ of } B) \cup T) \quad \langle B \vdash bfunbind \longrightarrow F \rangle \end{array}}{B \vdash funid_1 = funid_2 \ \langle \texttt{and}\ bfunbind \rangle \longrightarrow \{funid_1 \mapsto \Phi\} \langle + F \rangle} \tag{18}$$

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 $\mathcal{P}$ returns a *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.

$$\begin{array}{rcl} BE & \in & \text{BasEnv} = \text{BasId} \xrightarrow{\text{fin}} \text{MBasis} \\ M \text{ or } FE, BE, B & \in & \text{MBasis} = \text{FixEnv} \times \text{BasEnv} \times \text{Basis} \\ \Psi & \in & \text{BasCache} = \text{MLBasisPath} \xrightarrow{\text{fin}} \text{MBasis} \end{array}$$

Figure 6: Compound Semantic Objects

3.3 Inference Rules

The semantic rules allow sentences of the form

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

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 *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 *basexp*, *basdec* or *basbind*.

Basis Expressions

$$\boxed{M, \Psi \vdash \mathit{basexp} \longrightarrow M', \Psi'/p}$$

$$\frac{M, \Psi \vdash \mathit{basdec} \longrightarrow M', \Psi'}{M, \Psi \vdash \texttt{bas}\ \mathit{basdec}\ \texttt{end} \longrightarrow M', \Psi'} \tag{19}$$

$$\frac{M(\mathit{basid}) = M'}{M, \Psi \vdash \mathit{basid} \longrightarrow M', \Psi} \tag{20}$$

$$\frac{M, \Psi \vdash \mathit{basdec} \longrightarrow M_1, \Psi_1 \quad M \oplus M_1, \Psi_1 \vdash \mathit{basexp} \longrightarrow M_2, \Psi_2}{M, \Psi \vdash \texttt{let}\ \mathit{basdec}\ \texttt{in}\ \mathit{basexp}\ \texttt{end} \longrightarrow M_2, \Psi_2} \tag{21}$$

Basis-level Declarations

$$\boxed{M, \Psi \vdash \mathit{basdec} \longrightarrow M', \Psi'/p}$$

$$\frac{M, \Psi \vdash \mathit{basbind} \longrightarrow BE', \Psi'}{M, \Psi \vdash \mathsf{basis}\ \mathit{basbind} \longrightarrow BE' \text{ in MBasis}, \Psi'} \tag{22}$$

$$\frac{M, \Psi \vdash \mathit{basdec}_1 \longrightarrow M_1, \Psi_1 \quad M + M_1, \Psi_1 \vdash \mathit{basdec}_2 \longrightarrow M_2, \Psi_2}{M, \Psi \vdash \texttt{local}\ \mathit{basdec}_1\ \texttt{in}\ \mathit{basdec}_2\ \texttt{end} \longrightarrow M_2, \Psi_2} \tag{23}$$

$$\frac{M(\mathit{basid}_1) = M_1 \quad \cdots \quad M(\mathit{basid}_n) = M_n}{M, \Psi \vdash \texttt{open}\ \mathit{basid}_1 \cdots \mathit{basid}_n \longrightarrow M_1 + \cdots + M_n, \Psi} \tag{24}$$

$$\frac{B \text{ of } M \vdash \mathit{bstrbind} \longrightarrow SE}{M, \Psi \vdash \texttt{structure}\ \mathit{bstrbind} \longrightarrow SE \text{ in MBasis}, \Psi} \tag{25}$$

$$\frac{\text{Inter } (B \text{ of } M) \vdash \mathit{bsigbind} \longrightarrow G}{M, \Psi \vdash \texttt{signature}\ \mathit{bsigbind} \longrightarrow G \text{ in MBasis}, \Psi} \tag{26}$$

$$\frac{B \text{ of } M \vdash \mathit{bfunbind} \longrightarrow F}{M, \Psi \vdash \texttt{functor}\ \mathit{bfunbind} \longrightarrow F \text{ in MBasis}, \Psi} \tag{27}$$

$$\frac{}{M, \Psi \vdash \quad \longrightarrow \{\} \text{ in MBasis}, \Psi} \tag{28}$$

$$\frac{M, \Psi \vdash \mathit{basdec}_1 \longrightarrow M_1, \Psi_1 \quad M + M_1, \Psi_1 \vdash \mathit{basdec}_2 \longrightarrow M_2, \Psi_2}{M, \Psi \vdash \mathit{basdec}_1\ \langle;\rangle\ \mathit{basdec}_2 \longrightarrow M_1 \oplus M_2, \Psi_2} \tag{29}$$

$$\frac{\mathcal{P}(FE \text{ of } M, \mathsf{path.sml}) = (FE', \mathit{topdec}) \quad B \text{ of } M \vdash \mathit{topdec} \Rightarrow B'}{M, \Psi \vdash \mathsf{path.sml} \longrightarrow (FE', \{\}, B'), \Psi} \tag{30}$$

$$\frac{\Psi(\mathsf{path.mlb}) = M'}{M, \Psi \vdash \mathsf{path.mlb} \longrightarrow M', \Psi} \tag{31}$$

$$\frac{\mathsf{path.mlb} \notin \text{Dom } \Psi \quad \mathcal{P}(\mathsf{path.mlb}) = \mathit{basdec} \quad \{\} \text{ in MBasis}, \Psi \vdash \mathit{basdec} \longrightarrow M', \Psi'}{M, \Psi \vdash \mathsf{path.mlb} \longrightarrow M', \Psi' + \{\mathsf{path.mlb} \mapsto M'\}} \tag{32}$$

Comments:

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

Basis Bindings

$$\boxed{M, \Psi \vdash \mathit{basbind} \longrightarrow BE', \Psi'/p}$$

$$\frac{M, \Psi \vdash \mathit{basexp} \longrightarrow M', \Psi' \quad \langle M, \Psi' \vdash \mathit{basbind} \longrightarrow BE'', \Psi''\rangle}{M, \Psi \vdash \mathit{basid} = \mathit{basexp}\ \langle \texttt{and}\ \mathit{basbind}\rangle \longrightarrow \{\mathit{basid} \mapsto M'\}\langle +BE''\rangle, \Psi'\langle '\rangle} \tag{33}$$

(Basis) Structure Bindings

$$\boxed{B \vdash bstrbind \longrightarrow SE}$$

$$\frac{B(strid_2) = E \quad \langle B \vdash bstrbind \longrightarrow SE \rangle}{B \vdash strid_1 \ = \ strid_2 \ \langle \texttt{and} \ bstrbind \rangle \longrightarrow \{strid_1 \mapsto E\} \langle + SE \rangle} \tag{34}$$

Comments:

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

(Basis) Signature Bindings

$$\boxed{IB \vdash bsigbind \longrightarrow G}$$

$$\frac{IB(sigid_2) = I \quad \langle IB \vdash bsigbind \longrightarrow G \rangle}{IB \vdash sigid_1 \ = \ sigid_2 \ \langle \texttt{and} \ bsigbind \rangle \longrightarrow \{sigid_1 \mapsto I\} \langle + G \rangle} \tag{35}$$

Comments:

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

(Basis) Functor Bindings

$$\boxed{B \vdash bfunbind \longrightarrow F}$$

$$\frac{B(funid_2) = (strid : I, strexp, B) \quad \langle B \vdash bfunbind \longrightarrow F \rangle}{B \vdash funid_1 \ = \ funid_2 \ \langle \texttt{and} \ bfunbind \rangle \longrightarrow \{funid_1 \mapsto (strid : I, strexp, B)\} \langle + F \rangle} \tag{36}$$

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.

Derived Form	Equivalent Form
(Basis) Structure Bindings *bstrbind*	
$strid \ \langle \texttt{and} \ bstrbind \rangle$	$strid \ = \ strid \ \langle \texttt{and} \ bstrbind \rangle$
(Basis) Signature Bindings *bsigbind*	
$sigid \ \langle \texttt{and} \ bsigbind \rangle$	$sigid \ = \ sigid \ \langle \texttt{and} \ bsigbind \rangle$
(Basis) Functor Bindings *bfunbind*	
$funid \ \langle \texttt{and} \ bfunbind \rangle$	$funid \ = \ funid \ \langle \texttt{and} \ bfunbind \rangle$

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

References

[MTHM97] Robin Milner, Mads Tofte, Robert Harper, and David B. MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.