Changes to proposal for standard ML

1. **Identifiers.** Introduce "~" as a letter. Thus, an identifier is (as before) a letter followed by zero or more letters or digits followed by zero or more primes. (...or a sequence of symbols)

2. **Type variables.** A prime followed by an identifier of the first kind.

3. **Wildcard.** "_" takes the place of "any" (doesn't combine lexically with anything)

4. **Characters and Strings.** "Token" is abolished. "char" is a standard type; char constants are e.g. `\x27A`, `\x27` , and various representation of things like `ab`, end-of-line. "String" is an abbreviation for "char list"; "c1,c2...cn" abbreviates `[c1; ...; cn]`

5. **Predefined types**

(1) **type void == empty**

**Convention:** "empty" is represented by "( )"

(2) **type bool == true / false**

(3) **local rec type posint == one / succ of posint**

**in type int == zero / pos of posint / neg of posint**

**Convention:** zero, pos (succ k = one), neg (succ k = one) are represented by 0, k, ~k (k ≥ 1)

(4) **type char == c1 /.../ cn**

% &c1,...,cn is the character set.

**Convention:** c1 is represented by `\x27C1` for visible characters, and by e.g. `\x27R` for carriage return.
(5) \textit{\texttt{infix}} :: 30 \texttt{right} ;
\texttt{rec type }'a\texttt{ list} == \texttt{nil} | \texttt{op} :: \texttt{if }'a\texttt{ # }'a\texttt{ list}

6. The keyword \texttt{"op"} is to be used in place of \texttt{"infix"} to qualify non-infixed uses of infixes.

7. 

\underline{Exceptions}. Introduce the declaration \texttt{"packet id"} to declare an exception class. Add the expressions

\texttt{eject id exp} to eject an \texttt{id} packet loaded with the value of \texttt{exp}.

\texttt{exp hold id match} to catch an \texttt{"id"} packet and match the value with which it is loaded.

Then we assume a predeclaration

\texttt{packet string : string}.

Note that \texttt{"?"} only traps \texttt{string} packets, not arbitrary packets. This is to avoid a style of undisciplined use of ejections.

\texttt{can't fight "packet" switching.}

\texttt{type-checking: what about "open-ended" scopes of "packet" resulting from top-level declarations.}

\texttt{-- need weak-type variables? (treatment analogous to references)
8. **Evaluation of expressions**. Always left to right, i.e. `exp1` is evaluated before `exp2` in both `"exp1 exp2"` and `"exp1, exp2"`.

9. **Matches**. Variables are matched left-to-right. A compiler warning is issued in two cases:
   1. If a more specific varstruct follows a less specific one.
   2. If the collection of varstructs is not exhaustive.

In the latter case, an unmatchable packet is ejected. (Unfortunately, to avoid user exploitation of non-exhaustive matches)

10. **Value Bindings** (vb). This is the new name for "variable bindings". We use `"=="` in place of `"<-"`.

11. **Type Bindings** (tb). This is the new name for "data bindings". We use `"=="` in place of `"<-"`. Note (under 5 (v) above) that the keyword `"of"` must qualify an infixed constructor in a type binding.

12. **Abstract Bindings**. There are none now. But we have the abbreviation
   
   `{tyvar_seq? id <== ty} → {tyvar_seq? id == abs id of ty}

13. **Declarations**. The syntax is now

   ```
   dec ::- {rec? {val? vb | t6 % optional keyword "val"? | t6 | type t6 | abstype t6 with dec end | local dec1 in dec2 end | packet id | dec1 ; dec2
   ```
14 Standard expression attacuation.

\[
\begin{align*}
\text{escape } \text{exp} & \quad \rightarrow \quad \text{exec } \text{string } \text{exp} \\
\text{exp} \ \text{trap} \ \text{match} & \quad \rightarrow \quad \text{exp} \ \text{hold} \ \text{string} \ \text{match} \\
\text{exp} \ ? \ \text{exp} 2 & \quad \rightarrow \quad \text{exp} 1 \ \text{trap} \ - \ \text{exp} 2 \\
\text{case} \ \text{exp} \ \text{of} \ \text{match} & \quad \rightarrow \quad (\text{fun} \ \text{match}) \ \text{exp} 2 \\
\text{if } \text{exp} \ \text{then} \ \text{exp} 1 \ \text{else} \ \text{exp} 2 & \quad \rightarrow \quad \text{case} \ \text{exp} \ \text{of} \ (\text{true}, \ \text{exp} 1) | (\text{false}, \ \text{exp} 2) \\
\text{exp} 1 \ \text{or} \ \text{exp} 2 & \quad \rightarrow \quad \text{if} \ \text{exp} 1 \ \text{then} \ \text{true} \ \text{else} \ \text{exp} 2 \\
\text{exp} 1 \ \text{&} \ \text{exp} 2 & \quad \rightarrow \quad \text{if} \ \text{exp} 1 \ \text{then} \ \text{exp} 2 \ \text{else} \ \text{false} \\
\text{exp} \ \text{where} \ \text{dec} \ \text{end} & \quad \rightarrow \quad \text{let} \ \text{dec} \ \text{in} \ \text{exp} \ \text{end} \\
\text{fun} \ \text{v1} \ \ldots \ \text{vns} : t s \ \text{exp} & \quad \rightarrow \quad \text{fun} \ \text{v1}, \ \ldots, \ \text{fun} \ \text{on} \ \text{exp} : t s : (n \geq 1) \\
[\text{exp} 1, \ \ldots, \ \text{exp} n] & \quad \rightarrow \quad \text{exp} 1 :: \ldots :: \text{exp} n :: \text{nil} \quad (n \geq 0) \\
"c_1 \ldots c_n" & \quad \rightarrow \quad [\text{'c}_1 : \ldots : \text{'c}_n : ] \\
\text{exp} 1 \ ; \ \text{exp} 2 & \quad \rightarrow \quad \text{let} \ _==\ \text{exp} 1 \ \text{in} \ \text{exp} 2 \ \text{end} \\
\text{while} \ \text{exp} 1 \ \text{do} \ \text{exp} 2 & \quad \rightarrow \quad \text{let} \ f() = \text{if} \ \text{exp} 1 \ \text{then} \ \text{exp} 2 ; f() \ \text{else} () \ \\
& \quad \hspace{1cm} \text{in} \ f() \ \text{end} \quad \text{? \ type \ is \ void ! ?} \\
\text{quit} & \quad \rightarrow \quad \text{escape } \ "\text{quit}" 
\end{align*}
\]
15. Standard varstinct abbreviation

\[ [v_1; \ldots; v_n] \longrightarrow v_1; \ldots; v_n; \text{nil} \]

"c_1; \ldots; c_n" \longrightarrow [c_1; \ldots; c_n]

16. Standard binding abbreviation

\{ (\text{tyvar_seq}, \text{id}) \rightarrow \text{ty} \}

\id \rightarrow \text{id} = \text{abs} \text{id} \text{ of ty}

\id \cdot v_1 \ldots \cdot v_n \cdot \text{ty} \rightarrow \text{id} = \text{fun} v_1 \ldots \cdot (\text{ty} \cdot \text{exp}) \ (n \geq 1)

v_1 \cdot \id \cdot v_2 \cdot v_3 \ldots \cdot v_n \cdot \text{ty} \rightarrow \text{op} \ (v_1, v_2) \cdot v_3 \ldots \cdot v_n \cdot \text{ty} \rightarrow \text{exp}

\ (\text{when\ id\ is\ an\ infix}) \ (n \geq 2)

\id \cdot v_1 \cdot \text{ty} \rightarrow \text{id} = \text{fun} v_1 \cdot \text{exp} \ldots \text{on} \text{exp} \ (n \geq 2)

17. Standard declaration abbreviation

\text{exp} \rightarrow \text{val \ id} = \text{exp}

Note that "id" is just an ordinary variable. This abbreviation can be used anywhere, but is mainly for top-level use.

18. External ML files

The declaration "use "filename" " can occur anywhere except within a match, or within any abbreviation which expands to a match (this includes a whole expression!). The file may be any ML command sequence (which is equivalent to a single declaration), or may be a pre-amplified declaration.