Robin Milner, undated, probably April/May 1983.
(appears to be changes relative to the first (TENTATIVE) draft of the Proposal for SML)

Changes to proposal for standard ML

no – use ' instead.

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 `A`, ` `, and various representations of things like tab, end-of-line. "string" is an abbreviation for "char list"; "$c_1c_2\cdots c_n$" abbreviates [`$c_1$`; ... ; `$c_n$`]

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

ω-many const.

CONVENTION: zero, pos(succ^{k-1}one), neg(succ^{k-1}one) are represented by 0, k, ~k (k ≥ 1)

(4) type char == c_1 | .... | c_n % {c_1, ..., c_n} is the character set.

CONVENTION: c_i is represented by `$c_i$` for visible characters, and by e.g. `\R` for carriage return.

(5) <u>infix</u> :: 30 right ;
<u>rec</u> <u>type</u> 'a list == nil | <u>op</u> :: <u>of</u> 'a # 'a list

6. The keyword "<u>op</u>" is to be used in place of "<u>infix</u>" to qualify non-infixed uses of infixes.

7. <u>Exceptions</u>. Introduce the declaration "<u>packet</u> id" to declare an exception class. Add the expressions

<u>eject</u> id exp	to eject an 'id' packet loaded with the value of exp.
exp <u>hold</u> id match	to catch an "id" packet and match the value with which it is loaded.

Then we assume a predeclaration

<u>packet</u> string : string. % "string" here is actually an unusable identifier %

and we have the abbreviations

$$\underline{\text{escape}}\ \text{exp} \longmapsto \underline{\text{eject}}\ \text{string exp}$$

$$\underline{\text{trap}}\ \text{match} \longmapsto \underline{\text{hold}}\ \text{string match}$$

$$\text{exp1} \mathbin{?} \text{exp2} \longmapsto \text{exp1}\ \underline{\text{trap}}\ \_ .\ \text{exp2}$$

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

can't fight "<u>packet</u>" switching.
type-checking: what about "open-ended" scopes of "packets" resulting from top-level declarations.
— need weak-type variables? (treatment analogous to references)

8. Evaluation of expressions. Always left to right, ie exp1 is evaluated before exp2 in both "exp1 exp2" and "exp1, exp2".

9. Matches. Varstructs are matched left-to-right. A compiler warning is issued in two cases:

    (1) If a more specific varstruct (& is ∴ subsumed) follows a less specific one.
    (2) If the collection of varstructs is not exhaustive.

In the latter case an untrappable packet is ejected. (Untrappable, to avoid user exploitation of nonexhaustive 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(5) 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                 % optional keyword "val" %
         {rec} type tb
         {rec} abstype tb with dec end
         local dec1 in dec2 end
         packet id
         dec1 ; dec2
```

14 Standard expression abbreviations.

escape exp $\longmapsto$ eject string exp

exp trap match $\longmapsto$ exp hold string match

exp1 ? exp2 $\longmapsto$ exp1 trap _ . exp2

case exp of match $\longmapsto$ (fun match) exp   % but with let style typechecking %

if exp then exp1 else exp2 $\longmapsto$ case exp of (true . exp1)|(false . exp2)

exp1 or exp2 $\longmapsto$ if exp1 then true else exp2

exp1 & exp2 $\longmapsto$ if exp1 then exp2 else false

exp where dec end $\longmapsto$ let dec in exp end

fun v1 ... vn{:ty} exp $\longmapsto$ fun v1. .... . fun vn. exp{:ty} $(n \geq 1)$

[exp1; ... ; expn] $\longmapsto$ exp1 :: ... :: expn :: nil $(n \geq 0)$

"c_1 .. c_n" $\longmapsto$ [\`$c_1$\`; ... ; \`$c_n$\`]

exp1 ; exp2 $\longmapsto$ let _ == exp1 in exp2 end

while exp1 do exp2 $\longmapsto$ let f() == if exp1 then exp2 ; f() else ()
in f() end   % type is void ! %

quit $\longmapsto$ escape "quit"

15. Standard varstruct abbreviations

$[v1; \dots ; vn] \longmapsto v1 :: \dots :: vn :: nil$

$"c_1 \dots c_n" \longmapsto [`c_1`; \dots ; `c_n`]$

16. Standard binding abbreviations

$\{tyvar\_seq\}\ id <=> ty \longmapsto \{tyvarseq\}\ id == abs\sim id\ \underline{of}\ ty$

$id\ v1 \dots vn \{:ty\} == exp \longmapsto id == \underline{fun}\ v1 \dots vn \{:ty\} . exp \quad (n \geq 1)$

$v1\ id\ v2\ v3 \dots vn \{:ty\} == exp \longmapsto \underline{op}\ id\ (v1, v2)\ v3 \dots vn \{:ty\} == exp$
(when id is an infix) $(n \geq 2)$

$id\ v1 \{:ty\} == exp1 | \dots | id\ vn \{:ty\} == expn \longmapsto$
$id == \underline{fun}\ v1.exp1 | \dots | vn.expn \quad (n \geq 2)$.

17. Standard declaration abbreviation

$exp \longmapsto \underline{val}\ it == exp$

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

18. External ML files

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