Typechecking proceeds as in standard ML, together with the following
constraints:

1. Types containing full variables may never be substituted for
   weak variables (any types can be substituted for full
   variables).

2. For "e e'" to be well-typed, e must have type ty'\to ty, e'
   must have type ty' and all weak variables in ty' must occur
   in the types of enclosing \textbackslash-bound varstructures.

3. The ith argument position of a type operator \textit{op} defined by
   "abs[rec]type (xl,\ldots,xn)op = \ldots" is defined to be weak if
   xi is weak, otherwise it is full.

4. In a type "(ty1,\ldots,tyn)op", if the ith argument position of
   \textit{op} is weak then tyi must not contain any full type variables.

To help motivate the first two constraints consider:

\[
\text{let } f = \lambda x . \text{let } r = \text{newref } x \text{ in } (\lambda z . r = z, \lambda () . \text{cont } r) \text{ in }
\]

\[
f \text{ will be ascribed type: } * \to (\to ) \notin (\to \to ) \text{ now for each }
\]

application "f e" not occurring inside a \textbackslash, e must have a

\textit{definite monotype}; thus evaluating "let store, fetch = f([[]])" at top level is

\textit{prohibited}. However both "let store, fetch = f([[]]:form)" and

"let store, fetch = f([[]]:thm)" would be allowed.

The third and fourth constraint govern the creation and use of new

\textit{type operators}, for example consider:

\[
\text{abstype \textit{array} = (\textit{list} \# \textit{int} \# \textit{int} \text{ with newarray}(l, n, n, 2) = }
\]

\[
\text{if length } l = n-2+1 \text{ then absarray(map newref } l, n, 2) \text{ else failwith 'newarray'}
\]

\[
\text{and select a n = let l, n, 2 = reparray a in if n<n or n>n2 then failwith 'select'
\]

\[
\text{else el(l-1)} 1 \text{ where } \text{rec } el n 1 = n-1 \Rightarrow \text{hd l } | \text{ el(l-1)}(11)
\]

This defines a new unary type operator "\textit{array}" whose only argument is

\textit{weak}, and whose primitives have generic types:

\[
\text{newarray : \textit{list} \# \textit{int} \# \textit{int} \to \textit{array}
\]

\[
\text{select : \textit{array} \to \textit{int} \to \textit{int} \to \textit{int}}
\]

To create a new array with lower bound n1, upper bound n2, and initial

\textit{contents} v1,\ldots,vn \text{ (where } n=n2-n1+1 \text{) one evaluates

"newarray(v1;\ldots;vn), n1,n2"}. To get the value stored at the ith

\textit{component of a} one evaluates "cont(select a i)"; to change the value to

\textit{v} one evaluates "select a i := v".

Note suggesting research topic for Damas, circa 1980?