Type: \( t \text{ list} \), for any type \( t \).

Values: \([v_1, \ldots, v_n]\), with each \( v_i \) a value of type \( t \) and \( n \geq 0 \).
When \( n = 0 \), one really means the empty list, which appears in ML as \([]\) and is pronounced “nil”.

Expressions: The collection of expressions for this type includes all values of type \( t \text{ list} \).
It further includes all expressions of the form \( e::es \), with \( e : t \) and \( es : t \text{ list} \).
For example, a valid expression is \( 1::[2,3] \). This gives the value \([1,2,3]\) of type \( \text{int list} \).
The operator \( :: \) is right associative. It is pronounced “cons”.

Typing Rules:

\[
\begin{align*}
  [] & : t \text{ list} \\
  e::es & : t \text{ list} \quad \text{if } e : t \text{ and } es : t \text{ list}.
\end{align*}
\]

Evaluation: Evaluation of lists proceeds from left to right, until one obtains values. Formally:

\[
\begin{align*}
  [] & \text{ is a value} \\
  e::es & \Rightarrow e'::es \quad \text{if } e \Rightarrow e' \\
  v::es & \Rightarrow v::es' \quad \text{if } es \Rightarrow es' \text{ and } v \text{ is a value}.
\end{align*}
\]

Pattern Matching: One can pattern match on the structure of a list. For instance, in a \texttt{case} expression one might have a clause of the form

\[
| x::xs \Rightarrow e
\]

When type-checking \( e \), the compiler will use \( x : t \) and \( xs : t \text{ list} \) (assuming that is consistent with other patterns and the rest of the \texttt{case}).

At runtime, if the pattern \( x::xs \) is matched by a value of the form \( v::vs \), the evaluator will create local bindings \([v/x, vs/xs]\). The evaluator will evaluate \( e \) with these bindings in scope.
Let us write a short function that computes the length of a list. Then we will use structural induction to prove that the function is total.

(* length : int list -> int
  REQUIRES: true
  ENSURES: length(L) returns the number of integers in L.
  *)

fun length ([] : int list) : int = 0
  | length (x::xs) = 1 + length(xs)

Theorem:  length is total.

Proof: Establishing totality means proving that length(L) reduces to a value for all list values L of type int list.

We will prove that assertion by structural induction on L.

BASE CASE: L = [].

NEED TO SHOW: length [] reduces to a value.
SHOWING:

  length []
  ⇒  0  [first clause of length]

0 is a value, so we have established the base case.

INDUCTION STEP: L = x::xs, for some values x : int and xs : int list.

INDUCTION HYPOTHESIS: length(xs) ↦ v for some value v.

NEED TO SHOW: length(x::xs) ↦ v’ for some value v’.
SHOWING:

  length(x::xs)
  ⇒  1 + length(xs)  [step, second clause of length]
  ⇒  1 + v  [Inductive Hypothesis]
  ⇒  v’  [for some value v’, assuming SML addition is correct]

That establishes the induction step.

The base case and the induction step together establish the Theorem, by a principle of structural induction for int list.