Rules for Expressions in ML

The Why?

When we learn the sematics we can build upon them to reason with more complex ideas and expressions

Expressions

Every expression has the following

- Syntax
- Type Checking rules
    - Produces a type or fails
- Evaluation rules (only on things that type checks)
    - Produces a **value** or **exception** or **infinite loop**

There is a constant *true* just like there is a constant *34*. Conditional expressions will evaluate to *true/false*.

There are three questions I need to ask when looking at an expression

1. What is the *syntax*? (ie how do you write it down)
2. What are the type checking rules. (what will cause it to fail to type check)
3. What are the evaluation rules assuming it does type check. (How does it perform its computation in order to produce a result).
    1. Expression dont always have to produce a result, sometimes they raise an exception or loop infinitely.

Variables

- Syntax: any sequence of digits or letters or underscore (_). **Cant start with a digit**
- Type-Checking: Applies only when were using a variable not defining it: We look up the type in the **static** environment. **Fails if its not there**
- Evaluation: Look up the value in the **dynamic** environment.

Only type-checked variables are evaluated in ML

Addition

- Syntax: Any *expression* where you have other *expressions* with a *plus* symbol in between them. ie: `e1 + e2` where `e1` and `e2` are expressions with `+` in between them
- Type-Checking: You have to type check both the sub expressions
    - Fails: If either variable doesn't type check. Has a type other than int or dont both have the same type. if `e1` and `e2` have type `int` then **`e1 + e2`** has type **`int`**
- Evaluation: Evaluates both expression. Sum the evaluated expressions.
    - `e1` and `e2` evaluates to *v1* and *v2* and `e1+e2` will evaluate to the *sum* of *v1* and *v2*

Values

The result of evaluating something is a value, every value is an expression. ie 34 is an expression that represent itself (think of 0s and 1s) Not all expressions are values. Every value evaluates to itself in "zero steps". **For each of these types, there are a certain set of values**.

- 34,17,42 have a type *int*
- true, false have type *bool*
- () have type *unit*

They're the answers that we get when we have an expression of that type.

Conditional Expressions

- Syntax: A conditional expression contains the keywords *if then* and *else* ie: `if e1 then e2 else e3` Where `e1 e2 e3` are subexpressions
- Type-Checking:
    1. The expression after the *if* must have a type of **boolean**,
    2. The expressions after the *then* and the *else* must have the **same type**. type *t*
    3. The **entire** conditional is the type of the expression after *then* and *else*. type *t* In `if e1 then e2 else e3`
        - `e1` must be type bool
        - `e2 e3` must have the same type
        - `if e1 then e2 else e3` is type `e2 e3`
- Evaluation:
    1. the expression after the *if* is evaluated, since its a bool it will be either *true or false*
    2. if true then the entire expression evaluates to the evaluation of the expression after the *then*
    3. if false then the entire expression evaluates to the evaluation of the expression after the *else*

Less than comparison

- Syntax: Two expressions seperated by `<`.
    - ie `e1 < e2`
- Type-Check: you have to type check both sub expressions
    - Fails: If either variables doesnt type check, or has a type other than `int` and arent the same type
    - if `e1` and `e2` has both type `int` then `e1 < e2` has a type `bool`
- Evaluation: Evaluate the two expressions to two values, if v1 is less then v2 then the entire expression evaluates to true, evaluates to false otherwise.