(*
Interlude: abstracting from the particular language
- to a theory of all abstract languages, and
- (all) possible interpreters (evaluators)
*)
(*
Signatures and Signed/Ranked Algebras *)
(*
Let us present a general framework for discussing Abstract Syntax

For simplicity, we confine our interest to a homogeneous algebraic framework: all expressions are of one *sort*. (The generalisation is called multi-sorted algebras.)
*)

(*
We first fix the symbols which any candidate (abstract) language will use:
Definition: Signature Σ - a set of symbols,
and for each symbol, its arity ≥ 0
*)

(* **Example**: Boolean expressions
`T` , `F` arity 0
`¬` arity 1
`\/`, `/\` arity 2
*)

(* **Example**: Integer expressions
(a denumerable set of) numerals arity 0
`+` , `*` arity 2
unary minus `-` arity 1
*)

(*
We now discuss the meaning of expressions in any language whose symbols have been specified using a signature Σ.

Semantics

Definition (Σ-algebra)
A — a carrier set, and
for each 0-ary symbol in Σ, associate some element of A
for each k-ary symbol in Σ, associate a total function in $A^k \to A$

Notes:
- Not all elements of A require having a 0-ary symbol in Σ associated with them. That is, the carrier set may contain values (perhaps infinitely many) for which there is no syntax. Irrational real numbers are a good examples — there may be

no way of writing a symbol for each irrational number. In fact, we may not even be able to write an expression for each irrational number.

- Different 0-ary symbols in $\sum$ *do not necessarily* have to be associated with *different* elements — two *different* symbols can be mapped to the same value. Similarly for k-ary symbols, two *different* symbols can be mapped to the same function.
- Two $\sum$-algebras can have the same carrier set A, but differ on the interpretation of the symbols in the signature $\sum$. These would be considered *different* $\sum$-algebras.

*)

(*

Note: Symbols do not *inherently* carry meaning. We can assign whatever meanings we choose to .... but once we assign meaning by choosing a particular $\sum$-algebra, we have fixed the meanings of the symbols.

In particular, we can give
Standard examples of $\sum$-algebras, where the symbols are given meanings that we all expect and agree upon, e.g., binary symbol + means addition on naturals, the 0-ary symbol numeral 0 means the natural number zero.

But we can also give
non-standard examples of $\sum$-algebras, where the meanings of symbols differ from what we normally expect, e.g., 0-ary symbol numeral 0 can be given the meaning "forty-two".
*)

(*
Notation:
If $\mathscr{A} = \langle A, \dots \rangle$ is a $\sum$-algebra, and $a \in A$ is associated with 0-ary symbol c in $\sum$, we write $c_{\mathscr{A}}$ to denote the element $a \in A$, highlighting that this is the meaning associated with symbol c in algebra $\mathscr{A}$.
Similarly, if f is a k-ary symbol in $\sum$, and in $\sum$-algebra $\mathscr{A} = \langle A, \dots \rangle$, f is associated with a total function $g : A^k \to A$, we write $f_{\mathscr{A}}$ to denote this function g.
*)

(* **Abstract Syntax, abstractly and formally**

Definition: (Carrier set $Tree_{\Sigma}$)
Suppose $\sum$ is a given signature.
The set $Tree_{\Sigma}$ is inductively defined as follows:

Base cases: for each 0-ary symbol c in $\sum$, a labelled node $\bullet c$ is in $Tree_{\Sigma}$
Induction cases: for *each* $k > 0$,
for each k-ary symbol f in $\sum$,
for *any* trees $t_1, \dots, t_k$ (already) in $Tree_{\Sigma}$,
the tree consisting of
a labelled node $\bullet f$ at the root
with the k trees $t_1, \dots, t_k$ ordered as subtrees below the root node
is in $Tree_{\Sigma}$.

Notes: The roots of $t_1, \ldots, t_k$ are the children, ordered 1...*k*, of the new root node •*f*
The t_i need not be distinct.

The Abstract Syntax Tree Algebra: *Syntax as Semantics*
Definition: **Tree**$_\Sigma$ is the Σ-algebra with carrier set $Tree_\Sigma$, where
- every 0-ary symbol *c* in Σ is interpreted as the labelled node •*c*, which is in $Tree_\Sigma$, and
- every *k*-ary symbol *f* in Σ is interpreted as the *tree-forming function* that takes *k* trees $t_1, \ldots, t_k$ and creates the tree

$$\begin{array}{c} \bullet f \\ / \quad \backslash \\ t_1 \ldots t_k \end{array}$$

in $Tree_\Sigma$ by sticking $t_1, \ldots, t_k$ as the subtrees below a new root node labelled •*f*.
*)
(*
Structure-preserving functions between Σ-algebras
Σ-homomorphisms are *structure-preserving* functions *between* Σ-algebras which respect the symbol association (imposed with respect to the symbols in the signature Σ) by the semantics of the source and target Σ-algebras.

Definition (Σ-homomorphisms) Suppose Σ is a given signature.
Suppose $\mathscr{A} = \langle A, \ldots \rangle$ and $\mathscr{B} = \langle B, \ldots \rangle$ are both Σ-algebras (for the same signature Σ).
A total function $h : A \rightarrow B$ between the carrier sets of these algebras is called a Σ-homomorphism if
- *for all* 0-ary symbols *c* in Σ: $h(c_{\mathscr{A}}) = c_{\mathscr{B}}$
- *for all* *k*-ary (*k* >0) symbols *f* in Σ,
  for all $a_1, \ldots a_k \in A$,
  $$h(\ f_{\mathscr{A}}(a_1, \ldots, a_k)\ ) \ = \ f_{\mathscr{B}}(\ h(a_1), \ldots, h(a_k)\ )$$
*)
(*
Initiality Theorem
For any Σ-algebra, $\mathscr{B} = \langle B, \ldots \rangle$, there is a unique Σ-homomorphism
$i_{\mathscr{B}} : Tree_\Sigma \rightarrow B$
from the Σ-algebra **Tree**$_\Sigma$ to the Σ-algebra $\mathscr{B}$.
Proof — by construction, define $i_{\mathscr{B}}$ to be a Σ-homomorphism.
Define $i_{\mathscr{B}} : Tree_\Sigma \rightarrow B$ as follows:
- for each 0-ary symbol *c* in Σ: $i_{\mathscr{B}}(\bullet c) = c_{\mathscr{B}}$
- for each *k*-ary (*k* >0) symbol *f* in Σ,

$$i_{\mathscr{B}} \left(\begin{array}{c} \bullet f \\ / \quad \backslash \\ t_1 \ldots t_k \end{array} \right) \ = \ f_{\mathscr{B}}(\ i_{\mathscr{B}}(t_1), \ldots,\ i_{\mathscr{B}}(t_k)\ )$$

Uniqueness of $i_{\mathscr{B}}$

Suppose $j : Tree_\Sigma \to B$ is also a Σ-homomorphism from $\mathbf{Tree}_\Sigma$ to $\mathscr{B}$.

Proof by induction on (`ht` t) that *for all* t in $Tree_\Sigma$, $i_{\mathscr{B}}(t) = j(t)$

Base cases (`ht` t) = 0.

t must be of the form $\bullet c$ for some 0-ary symbol in Σ.

But for each 0-ary symbol c in Σ: $i_{\mathscr{B}}(\bullet c) = c_{\mathscr{B}}$ // defn of $i_{\mathscr{B}}$

$= j(\bullet c)$ // j is a Σ-homomorphism // from $\mathbf{Tree}_\Sigma$ to $\mathscr{B}$

Induction Hypothesis:

Assume that for all t' in $Tree_\Sigma$ such that (`ht` t') ≤ n,

$i_{\mathscr{B}}(t') = j(t')$

Induction Step: Consider t in $Tree_\Sigma$ such that (`ht` t) = n+1

Therefore t is of the form

```
  •f
 /  \
t1...tk
```

with (`ht` t_i) ≤ n $(1 \le i \le k)$

Now, by definition of $i_{\mathscr{B}}$,

for each k-ary symbol (k >0) f in Σ,

```
iB(  •f  )
    /  \
   t1...tk
```

$= f_{\mathscr{B}}(i_{\mathscr{B}}(t_1),..., i_{\mathscr{B}}(t_k))$ // definition of $i_{\mathscr{B}}$

$= f_{\mathscr{B}}(j(t_1),..., j(t_k)).$ // by IH on each of $t_1 \ldots t_k$

```
j(  •f  )
   /  \
  t1...tk
```

// j is a Σ-homomorphism from $\mathbf{Tree}_\Sigma$ to $\mathscr{B}$

*)

(*

Variables

We now move from expressions to formulas, i.e., expressions that contain *variables*. As the name implies, a variable is a 0-ary symbol can be given different values, in contrast with constants in a signature Σ, whose values are fixed, once we fix a Σ-algebra $\mathscr{A}$. Assume a (denumerable) set $\mathscr{X}$ of variables, disjoint from symbols in Σ, i.e., $\mathscr{X} \cap \Sigma = \varnothing$ *)

(* Let us represent variables using the OCaml type `string` and extend the encoding of the abstract syntax for expressions in the data type `exp`:

*)

```
type exp = N of int
         | V of string
         | Plus of exp * exp
```

```
                  | Times of exp * exp;;
```

(* Note that we have a new family of base cases: variables, which are denoted using the parameterised constructor `V(_)` which takes arguments of type `string`.
*)

(* The corresponding Concrete Syntax can also be extended

E —> T
E —> T **+** E
T —> F
T —> F ***** T
F —> *n*
F —> *v*   *// a family of variables*
F —> **(** E **)**

*)

(*
We now extend the syntactic functions on tree-structured expressions *)

(* `size`, as before, returns the number of nodes in the tree, with a variable counted as a single node. *)

```
let rec size t = match t with
      N _ -> 1
    | V _ -> 1
    | Plus(t1, t2) -> (size t1) + (size t2) + 1
    | Times(t1, t2) -> (size t1) + (size t2) + 1
;;
```

(* `ht`, also as before, returns one less than the number of levels in the tree, with variables being treated as leaf nodes. *)

```
let rec ht t = match t with
      N _ -> 0
    | V _ -> 0
    | Plus(t1, t2) -> (max (ht t1) (ht t2)) + 1
    | Times(t1, t2) -> (max (ht t1) (ht t2)) + 1
;;
```

(* As mentioned before, both `size` and `ht` are good measures for performing induction on trees. *)

(* We now define a syntactic function on `exp` called `vars` which returns the *list* of variables in the tree. (We will later see that we really want a function returning the *set* of variables in a syntax tree. If we represent sets as lists, then the difference goes away; of course, we must avoid duplicates and not care about the relative order in which elements appear). *)

```
let rec vars t = match t with
      N _ ->   [ ]
    | V x ->   [x]
    | Plus(t1, t2) -> (vars t1) @ (vars t2)
```

```
    | Times(t1, t2) -> (vars t1) @ (vars t2)
;;
```

(* We now extend the *standard semantics* — the definitional interpreter of a language of expressions which include variables. The main question to answer is "what is the value of an expression that contains variables?" The way to approach this question is to pose a counter-question, asking what value each variable takes. So `eval` now takes an extra argument, which we call `rho`, that maps each variable to a value. For the most part, except the new base cases, argument `rho` is carried along as extra baggage in each call. But it cannot be left out. (Why not?) Note that `rho` is a function from `string` to `int` ! *)

```
let rec eval t rho = match t with
      N n ->  n
    | V x -> rho x
    | Plus(t1, t2) -> (eval t1 rho) + (eval t2 rho)
    | Times(t1, t2) ->  (eval t1 rho) * (eval t2 rho)
;;
```

(* We now extend the `compile` function. First we introduce an opcode `LOOKUP` to look up the value associated with a given variable.
*)

```
type opcode = LD of int
            | LOOKUP of string //look up variable's value
            | PLUS
            | TIMES
;;
```

(* The compiler as before, is a post-order traversal, i.e., a tree-recursive function, with a straightforward new base case. *)

```
let rec compile t = match t with
      N n ->  [LD (n)]
    | V x -> [ LOOKUP(x) ]
    | Plus(t1, t2) -> (compile t1) @ (compile t2) @ [PLUS]
    | Times(t1, t2) ->  (compile t1) @ (compile t2) @ [TIMES]
;;
```

(* The stack machine is again written as a tail recursive function driven by the first op-code and the state of the stack, and a new component `env` (environment), which maps variables to their corresponding answers.
*)

```
let rec stkmc s code env = match code with
      [ ] ->  s
    | (LD(n)) :: code' ->  stkmc (n :: s) code'  env
    | (LOOKUP(x))::code' -> let n=env x
                                in  stkmc (n::s)  code' env
    | PLUS :: code' -> (match s with
         n1 :: n2 :: s' ->  let n = n1 + n2
```

```
                                   in  stkmc (n :: s') code' env
      | _ -> raise Stuck)
  | TIMES :: code' -> (match s with
        n1 :: n2 :: s' ->  let n = n1 * n2
                           in  stkmc (n :: s') code' env
  | _ -> raise Stuck)
;;
```

(* Let us now abstract this treatment to arbitrary abstract syntax trees over arbitrary signatures Σ. We do so by defining a set $Tree_\Sigma(\mathscr{X})$.

Definition: Suppose Σ is a given signature.
Suppose $\mathscr{X}$ is a (denumerable) set of variables.
The set $Tree_\Sigma(\mathscr{X})$ is inductively defined as follows:

<u>Base cases</u>:
- for each 0-ary symbol c in Σ, a labelled node $\bullet c$ is in $Tree_\Sigma(\mathscr{X})$
- for each variable x in $\mathscr{X}$, *a labelled node* $\bullet x$ is in $Tree_\Sigma(\mathscr{X})$.

<u>Induction cases</u>: for each $k > 0$,
for each k-ary symbol f in Σ,
for *any* trees $t_1, \ldots, t_k$ in $Tree_\Sigma(\mathscr{X})$,
the tree consisting of a labelled node $\bullet f$ at the root
with k trees $t_1, \ldots, t_k$ ordered as subtrees below the root node
is in $Tree_\Sigma(\mathscr{X})$.

The algebra of abstract syntax trees
$\mathbf{Tree}_\Sigma(\mathscr{X})$ is the Σ-algebra with carrier set $Tree_\Sigma(\mathscr{X})$, where (as before)
- every 0-ary symbol c in Σ is interpreted as the labelled node $\bullet c$, in $Tree_\Sigma(\mathscr{X})$,
- every k-ary symbol ($k > 0$) f in Σ is interpreted as the tree-forming function that takes k trees $t_1, \ldots, t_k$ from $Tree_\Sigma(\mathscr{X})$, and creates the tree

```
    •f
   /  \
 t1...tk
```

by sticking $t_1, \ldots, t_k$ as the subtrees below a new root node labelled $\bullet f$. *)
(* Observe $\mathscr{X}$ is in 1-1 correspondence with a subset $\bullet\mathscr{X}$ of $Tree_\Sigma(\mathscr{X})$.
Note also that $Tree_\Sigma(\mathscr{X})$ contains $\bullet\mathscr{X} \cup Tree_\Sigma$.

Definition (B-valuation). A function $\rho : \mathscr{X} \to B$ that maps variables to values in a carrier set B is called a B-valuation.

Definition (Function extension)
A function h: $Tree_\Sigma(\mathscr{X}) \to B$ extends function $\rho : \mathscr{X} \to B$ if
for all $x \in \mathscr{X}$, $h(\bullet x) = \rho(x)$

Unique Homomorphic Extension Theorem
For any signature Σ, and
any Σ-algebra, $\mathscr{B} = \langle B, \ldots \rangle$,

and any B-valuation $\rho : \mathscr{X} \to B$,
there is a unique $\sum$-homomorphic extension of $\rho : \mathscr{X} \to B$, written

$\hat{\rho} : Tree_{\Sigma}(\mathscr{X}) \to B$

from the (syntactic) $\sum$-algebra $\mathbf{Tree}_{\Sigma}(\mathscr{X})$ to the $\sum$-algebra $\mathscr{B}$.

Proof (First) Existence of a $\sum$-homomorphic extension of $\rho : \mathscr{X} \to B$.
By construction: define $\hat{\rho} : Tree_{\Sigma}(\mathscr{X}) \to B$ to be a $\sum$-homomorphism as follows:
- For each 0-ary symbol c in $\sum$: $\hat{\rho}(\bullet c) = c_{\mathscr{B}}$
- For each $x \in \mathscr{X}$, $\hat{\rho}(\bullet x) = \rho(x)$
- For each k-ary symbol (k >0) f in $\sum$, and $t_1, \ldots, t_k$ in $Tree_{\Sigma}(\mathscr{X})$

$$\hat{\rho}\left(\begin{array}{c}\bullet f\\ / \quad \backslash \\ t_1 \ldots t_k\end{array}\right) = f_{\mathscr{B}}(\hat{\rho}(t_1),\ldots, \hat{\rho}(t_k))$$

<u>Uniqueness of $\hat{\rho}$</u>

Suppose j is an extension of ρ, and $j : Tree_{\Sigma}(\mathscr{X}) \to B$ is a $\sum$-homomorphism from $\sum$-algebra $\mathbf{Tree}_{\Sigma}(\mathscr{X})$ to the $\sum$-algebra $\mathscr{B}$.

Proof by induction on (`ht` t) that for all t in $Tree_{\Sigma}(\mathscr{X})$, $\hat{\rho}(t) = j(t)$
<u>Base cases</u> (`ht` t) = 0.
subcase t is of the form $\bullet c$
for each 0-ary symbol c in $\sum$: $\hat{\rho}(\bullet c) = c_{\mathscr{B}}$ // defn of $\hat{\rho}$
$= j(\bullet c)$ // j is a $\sum$-homomorphism
// from $\mathbf{Tree}_{\Sigma}(\mathscr{X})$ to $\mathscr{B}$
subcase t is of the form $\bullet x$ for some $x \in \mathscr{X}$
$\hat{\rho}(\bullet x) = \rho(x)$ // defn of $\hat{\rho}$
$= j(\bullet x)$ // j extends ρ

<u>Induction Hypothesis</u>:
Assume that for all t' in $Tree_{\Sigma}(\mathscr{X})$ such that (`ht` t) ≤ n,
$\hat{\rho}(t') = j(t')$

<u>Induction Step</u>: Consider t in $Tree_{\Sigma}(\mathscr{X})$ s.t. (`ht` t) = n+1
t must be of the form

$$\begin{array}{c}\bullet f\\ / \quad \backslash \\ t_1 \ldots t_k\end{array}$$

with (`ht` t_i) ≤ n $(1 \le i \le k)$
Now, by definition,
for each k-ary symbol (k >0) f in $\sum$, and $t_1, \ldots, t_k$ in $Tree_{\Sigma}(\mathscr{X})$ s.t. (`ht` t_i) ≤ n

$$\hat{\rho}\left(\begin{array}{c}\bullet f\\ / \quad \backslash \\ t_1 \ldots t_k\end{array}\right)$$

$= \ f_{\mathscr{B}}(\ \widehat{\rho}(t_1),..., \widehat{\rho}(t_k))$ // definition of $\widehat{\rho}$
$= \ f_{\mathscr{B}}(\ j(t_1),..., j(t_k))$ // by IH on each of $t_1 \ldots t_k$
$= \ j(\ \bullet f \ )$ // $j : Tree_\Sigma(\mathscr{X}) \to B$ is a $\sum$-homomorphism
$\quad / \ \backslash$ // from $\sum$-algebra $\mathbf{Tree}_\Sigma(\mathscr{X})$ to the $\sum$-algebra $\mathscr{B}$.
$\quad t_1 \ldots t_k$
*)

(* Only those variables that appear in a tree (term, expression) matter in its evaluation.
Lemma (Relevant Variables): Suppose $\sum$ is an *arbitrary* signature and $\mathscr{X}$ is the set of variables.
Let t in $Tree_\Sigma(\mathscr{X})$ be an arbitrary tree (term, expression).
Let V = `vars`(t).
Consider any $\sum$-algebra $\mathscr{B} = \langle B, \ldots \rangle$, and let $\rho_1, \rho_2 \in \mathscr{X} \to B$ be any two B-valuations such that *for all* variables $x \in V$, $\rho_1(x) = \rho_2(x)$.
[Note: $\rho_1(y)$ and $\rho_2(y)$ may be very different for $y \notin V$.]
Then $\widehat{\rho_1}(t) = \widehat{\rho_2}(t)$.

Proof by induction on `ht`(t).
Base cases (`ht`(t) = 0).
subcases: t is of the form $\bullet c$ for some 0-ary symbol c in $\sum$.
Therefore $\widehat{\rho_1}(\bullet c) \ = \ c_{\mathscr{B}} \ = \ \widehat{\rho_2}(\bullet c)$
subcases t is of the form $\bullet x$ for some x in $\mathscr{X}$. So $x \in V$.
Therefore $\widehat{\rho_1}(\bullet x) \ = \ \rho_1(x) \ = \ \rho_2(x) \ = \ \widehat{\rho_2}(\bullet x)$
Induction Hypothesis: Assume *for all* t' in $Tree_\Sigma(\mathscr{X})$ such that `ht`(t') ≤ n,
for all $\rho_1', \rho_2' \in \mathscr{X} \to B$ such that
for all variables $x \in V'$ = `vars`(t'), $\rho_1'(x) = \rho_2'(x)$,
that $\widehat{\rho_1'}(t') = \widehat{\rho_2'}(t')$.
Induction Step: Let t in $Tree_\Sigma(\mathscr{X})$ be such that `ht`(t) = n+1.
By analysis t is of the form $\bullet f$ for some symbol f of arity k in $\sum$
$\quad / \ \backslash$
$\quad t_1 \ldots t_k$
and trees $t_1, \ldots, t_k$ from $Tree_\Sigma(\mathscr{X})$, where `max` `ht`(t_i) = n. (1≤i≤k)
$\widehat{\rho_1}(\bullet f \ ) \ = \ f_{\mathscr{B}}(\ \widehat{\rho_1}(t_1),..., \widehat{\rho_1}(t_k))$
$\quad / \ \backslash$
$\quad t_1 \ldots t_k$
$= f_{\mathscr{B}}(\ \widehat{\rho_2}(t_1),..., \widehat{\rho_2}(t_k))$ // By IH on $t_1, \ldots, t_k$ with $\rho_1' = \rho_1$ and $\rho_2' = \rho_2$
// Note that V_i = `vars`(t_i) $\subseteq$ `vars`(t) = V
// so if *for all* $x \in V$, $\rho_1(x) = \rho_2(x)$,
// then *for all* $x \in V_i$, $\rho_1(x) = \rho_2(x)$ (1≤i≤k)
$= \widehat{\rho_2}(\bullet f \ )$
$\quad / \ \backslash$
$\quad t_1 \ldots t_k$
*)

(*
Proposition: For any t in $Tree_\Sigma(\mathcal{X})$, $V = \texttt{vars}(t)$ is finite.
Corollary: t in $Tree_\Sigma(\mathcal{X})$ can evaluate to at most $|B|^{|B|^{|V|}}$ values.
*)