Section 2

Plan for today

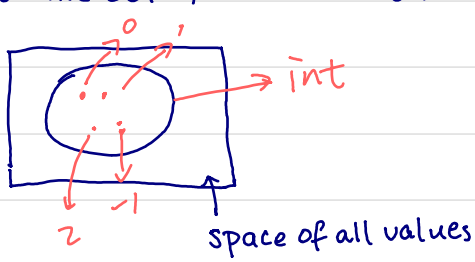
- Datatypes
- Recursion

Datatypes

What are types, anyway?

1. A type is the set of values that inhabit it.

   Example:



2. A type T is a contract that stipulates

   ① how to construct a value of type T

   ② given a value of type T, how to consume/destruct that value

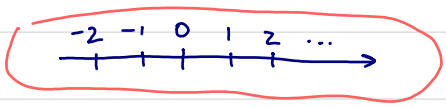
   ① ⇒ datatype constructors

   ② ⇒ pattern match on data type

Product Types : Package multiple things into one big thing.

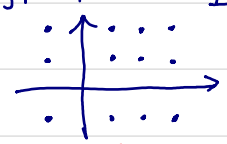
Example:

type point = int



values: 0, 1, -1, 2, -2, ...

type point = int * int



values: (0,0), (1,-2), (100,7)

→ In general, given type 'a and type 'b, the product, written as 'a * 'b, is a type that "bundles together" 'a and 'b.

Tip: Useful for returning multiple values in a function!

① How to contruct a value of type 'a * 'b, from a value of type 'a and a value of type 'b?

② How to destruct a value of type 'a * 'b, to get back a value of type 'a and a value of type 'b?

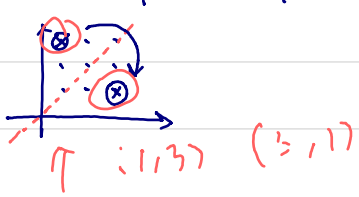
In general
type t = (a * b * ...)
Degenerate case unit

→ Exercise.

① Define a type, called `point`, that represents points in the 2D integer plane.

② Define a value of type `point` that represents the origin.

③ Write a function `reflect : point → point` that reflects a point along the 45° degree line.



Sum Types: OCaml's glorified 'enum'

Example:

Common Scenario:

A method may either succeed with a return value, or fail with no return value.

<Python>

```
def f() :
    ⋮
  if success :
    return (some int)
  else :
    return None
```

<C>

```
int f() {
    ⋮
  if (success) {
    return (some int);
  } else {
    return -1;
  }
```

Library

client

```
res = f()
if res is not None :
    ...
else :
    ...
```

```
int res = f();
if (res < 0) {
    ...
} else { ... }
```

Introducing option:

A type that expresses two possibilities:

either it is a success with some value

or it is a failure with none.

In general
type T = A | B | C ...

Degenerate case:
void
(no constructor)

① How to construct a value of type 'int option'?

② How to destruct a value of type 'int option'?

→ Exercise Write a function 'funny_add : (int option * int option) → int option' that returns the sum of the input options if both are not None, or returns None otherwise.

Product type : "~~ and ~~ and ~~"

Sum type : "either ~~, or ~~, or ~~"

Datatypes = sum of product + recursion

* A lot of common data structures can be defined this way!

Examples:

- A date is int (year) * int (month) * int (day)

- An (integer) linked list is
  - / (empty) — Nil : int list
  - int (head) * int list (tail) — Cons : (int * int list) → int list

- An (integer) binary tree is
  - / (leaf) — Leaf : int tree
  - int * tree * tree (non-leaf) — Node : (int * int tree * int tree) → int tree

- A regular expression is

literal match →	|	string			
concat →	|	regex	*	regex	r1 • r2
alternative →	|	regex	*	regex	r1 | r2
kleene star →	|	regex			

Exercise

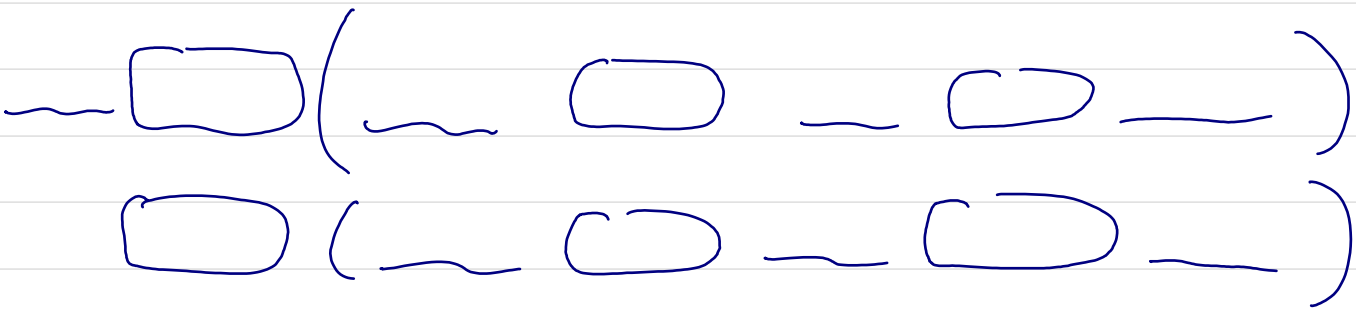
Translate those into Ocaml.

Exercise :

Define red-black trees (integer) using data types.

① Informally,

An integer red-black tree is :



② Translate the informal description to OCaml.

```
type rb_tree = ...
             | ...
```

Recursion

Recipe: ① define your base case(s)

→ ② define your recursive case by

(i) Split the input into sub-cases → usually the hardest, but in CS162 this is often trivial.

(ii) call the same fn on ↓

(iii) assemble solutions to ↓ into solution to overall problem.

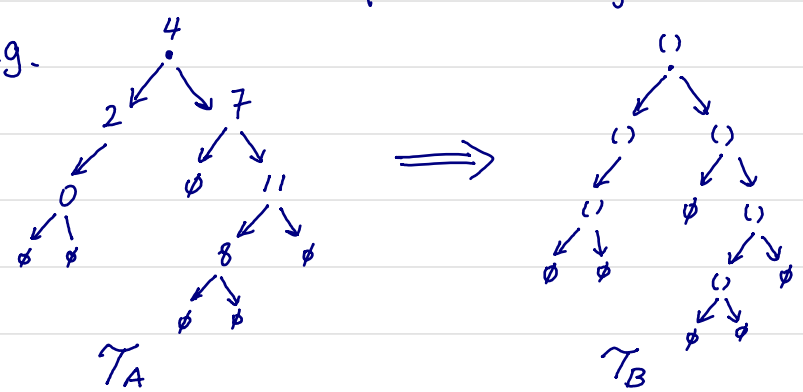
→ Question: What if the input to the recursive function has a recursive type, e.g. list, binary tree, etc?

Good news: You get (i) for free*, using pattern match.

Exercises. ① Define size: 'a binary_tree → int that computes the number of inner nodes of a binary tree.

② Define shape: 'a binary_tree → unit binary_tree that returns the "shape" of a binary tree.

E.g.



③ Define longest_path: 'a binary_tree → 'a list that returns the longest path from the root to a leaf. E.g. T_A ⇒ [4; 7; 11; 8]

*: most of the time, at least in CS162.