Type Systems

type system: a set of rules which specify a type for each part of a program

$$T : \text{tenv} \to \text{expr} \to \text{checked}$$

$$T(\Gamma, e)$$

$e ::= \mathbb{Z} \mid \mathbb{B} \mid e + e \mid \text{after}(e) \mid \ldots$

$\tau ::= \text{int} \mid \text{bool} \mid \text{tuple}(\tau)$

$$T(\emptyset, \text{true}) = \text{bool} \to \text{true}$$

$$T(\Gamma, e_1 < e_2) = \text{bool} \to (<) \to (\text{int}, \text{int}) \to (c_1, c_2)$$

when $T(\Gamma, e_1) = \text{int} \to c_1$ and $T(\Gamma, e_2) = \text{int} \to c_2$

$$T(\Gamma, \text{tuple}(e_1, \ldots, e_n)) = \text{tuple}\ \tau \to \text{tuple} \to (\tau_1, \ldots, \tau_n) \to (c_1, \ldots, c_n)$$

when, for every k s.t. $1 \le k \le n$, $T(\Gamma, e_k) = \tau_k \to c_k$ and $\tau_1 = \tau_2 = \cdots = \tau_n = \tau$

$$T(\Gamma, e_1[e_2]) = \tau \to \text{tuple index} \to (\text{tuple}\ \tau, \text{int}) \to (c_1, c_2)$$

$$T(\Gamma, e_1) = \text{tuple}\ \tau \to c_1$$

$$T(\Gamma, e_2) = \text{int} \to c_2$$

Generally:

- Varying type and fixed size
- Fixed type and varying size

$\tau ::= \text{int} \mid \text{bool} \mid (\tau, \ldots, \tau)$

$$T(\Gamma, \text{tuple}(e_1, \ldots, e_n)) = (\tau_1, \ldots, \tau_n) \to \text{tuple} \to (\tau_1, \ldots, \tau_n) \to (c_1, \ldots, c_n)$$

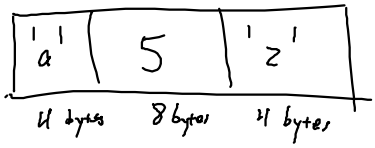
when, $\forall k \in \{1, \ldots, n\}$. $T(\Gamma, e_k) = \tau_k \to c_k$

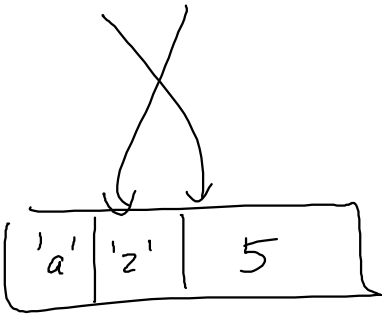
```
let (x, y, z) = t in
```

$e ::= \mathbb{A} \mid \mathbb{B} \mid \mathbb{C} \mid \cdots$

↑ 8 bytes ↑ 1 byte ↑ UTF-32 4 bytes

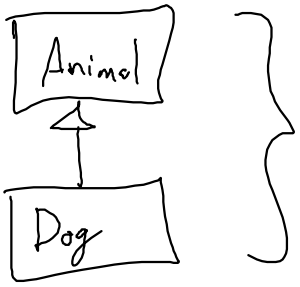
('a', 5, 'z') : (char, int, char)





Polymorphism

many form



object polymorphism:
I can use a Dog anywhere an Animal is required

```
let f n = (n, n);;
'a -> 'a * 'a
```

$\forall \alpha . \alpha \to (\alpha, \alpha)$

1. Templatization: create a template upon definition and fill it in when the item is used
2. Genericity: create a generic code structure to work with a consistent runtime representation