Type Systems

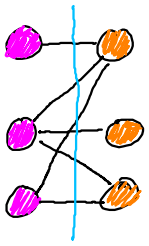
- Organizes types in a programming language
- Provides some guarantees/invariants (often used for "safety")
- Types dictate structure
- Types are sets of values*

Type	Values
bool	{true, false}
int	$\{0, 1, -1, 2, -2, \ldots, 2^{62}-1, -2^{62}\}$
bool → bool	$\{\{\text{true} \mapsto \text{true}, \text{false} \mapsto \text{false}\}, \{\text{true} \mapsto \text{true}, \text{false} \mapsto \text{true}\}, \ldots\}$

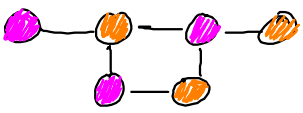
- A set is an unordered collection of unique values (e.g. $\{3\}, \{3,4\}, \{3,3\}, \{4, \frac{11}{2}\}$)

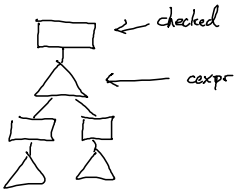
$\{3,4\} = \{4,3\}$ $\quad$ $\{3,3\} = \{3\}$

```
type expr =
    | EInt of int
    | EAfter of expr
    ....
type cexpr =
    | CInt of int
    | CAfter of checked
    .....
and checked =
    | Check of cexpr * typ
and typ =
    | TInt
    | TBool
```



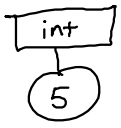
Bipartite graph

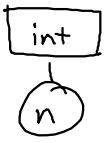


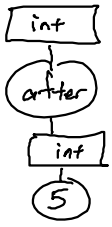


Type inference function:

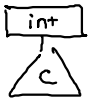
$T : \text{expr} \to \text{checked}$

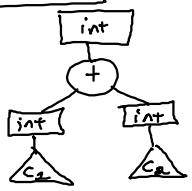
$T(5) =$ 

$T(n) =$ 

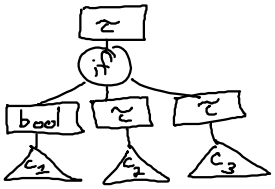
$T($after 5$) =$ 

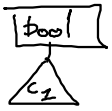
$T($after e$) =$ 

where $T(e) =$ 

$T($+ e_1 $e_2$$) =$ 

where $\forall i \in \{1,2\}.\ T(e_i) =$ 

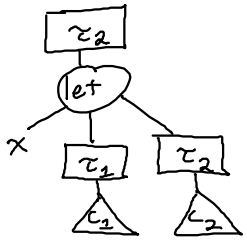
$T($if e_1 e_2 $e_3$$) =$ 

where $T(e_1) =$ 

and

$T(e_2) =$ 

and $T(e_3) =$ 

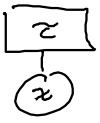
$T(\Gamma,$ let x e_1 $e_2$$) =$ 

where $T(\Gamma, e_1) =$ 

and

$T(\Gamma', e_2) =$ 

map from strings to typ

$T(\Gamma, x) =$ 

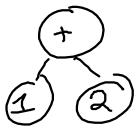
where $\Gamma(x) = \tau$

where $\Gamma' = \Gamma$ extended with $x \mapsto \tau_1$

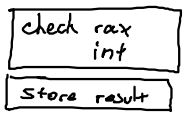
Why?

compiles to

NOT TO SCALE

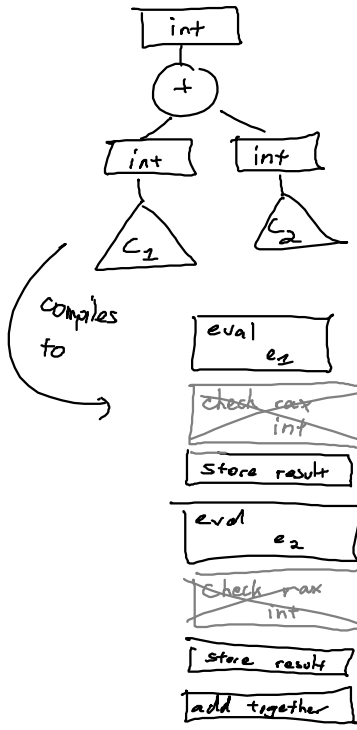


mov rax, 2



mov rax, 4

proof of soundness:
the $\boxed{\tau}$ on $\triangle c$ are correct



1. Help the programmer understand their code
2. Provide invariants for software like compilers