COS320 types - I

- Midterm due Friday
- HW3 due next week (Tuesday

halfway point thru class

Semantic Analysis, typechecking

<u>Semantic analysis phases</u> :

- Connects symbol occurance to definitions
  (i.e. scoping rules)
- AST well-typed checking
- E.g. break must appear inside for, while, switch.

- Report warnings (potential problems)
  errors (severe problems that must be resolved in order to compile).
- Semantic analysis may not be a separate phase (may be incorporated into IR translation).

- Main data structure: symbol table

  T : symbol ⟶ info about symbols

Semantic analysis may also decorate AST.

type checking: Catching some errors at compile-time.
Eliminates a class of mistakes that would otherwise lead to runtime errors.
type information sometimes <u>necessary</u> for compiling.

Typing

Intrinsic view (Church-style)

- a type is syntactically part of term
- typesystem part of syntax
- types do not have inherent meaning, just used to define syntax of program (more general than CFGs)

Extrinsic (Curry-style)

- type is <u>property</u> of term.
- ~~can~~ term have multiple or no types.

Dynamic typed languages: only Extrinsic view makes sense,
e.g. Javascript (Valid JS in which var doesn't have fixed type throughout compilation).

- A type system is system of judgement + Inference rules.

Extrinsic: Judgement is a claim

ex:

$\vdash 3 : int \Longrightarrow$ 3 has type integer (maybe true).

$\vdash (1+2) : bool \longrightarrow$ (1+2) has type bool (false)

Inference rules.

Add

$$\frac{\vdash e_1 : int \qquad \vdash e_2 : int}{e_1 + e_2 : int.}$$

if $e_1 \wedge e_2$ are ints $\Longrightarrow e_1 + e_2 : int.$

May involve different kinds of judgements

- Well typed expressions, types, statements.

Inference rules

$$\frac{\text{Premises } J_1 \ldots J_n}{\text{Conclusion } J} \quad \text{(side-condition: additional premise but not judgement)}$$

top-down: If premise and given side-condition then conclusion.

bottom-up: to prove J valid $\longleftarrow$ $J_1 \ldots J_n$ are valid.

ex

$\langle exp \rangle \longrightarrow$ var | int | exp + exp
| exp * exp
| $\wedge$
| $\vee$
| $\leq$
| =
| if exp then exp else exp

- 3 + (2 $\wedge$ 0) syntatically well-formed but not well typed
- Is x+1 welld-typed?

- type environment : Symbol table

Γ: symbol $\longrightarrow$ type.

$[x \mapsto int$
$y \mapsto bool$
$z \mapsto int]$ | x, z ints
y, bool

Notation: type env is Γ
Notation:

$\Gamma\{x \mapsto t\}$ is functional update.
change $\Gamma[x] \to t$.

$$\Gamma\{x \mapsto t\}(y) = \begin{cases} t & \text{if } x = y \\ \Gamma(y) & \text{otherwise.} \end{cases}$$

type judgement: $\Gamma \vdash e : t$

"Under type env Γ, expression e has type t".

Inference rules

Int :

$$\frac{}{\Gamma \vdash n : int} \quad n \in \mathbb{Z}$$

Side-condition
not judgement itself
but something we need
to hold.

Var :

$$\frac{}{\Gamma \vdash x : t} \quad \Gamma(x) = t$$

"given that
in Γ, $x \mapsto t$"

"x has type t"

Add :

$$\frac{\Gamma \vdash e_1 : Int \quad \Gamma \vdash e_2 : Int}{\Gamma \vdash e_1 + e_2 : Int}$$

And :

$$\frac{\Gamma \vdash e_1 : bool \quad \Gamma \vdash e_2 : bool}{\Gamma \vdash e_1 \wedge e_2 : bool}$$

Leq :

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 \leq e_2 : bool}$$

If :

$$\frac{\Gamma \vdash e_1 : bool \quad \Gamma \vdash e_2 : t \quad \Gamma \vdash e_3 : t}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : t}$$

$\boxed{\vdash : \text{Turnstile, Syntactic entailment}}$

Derivations / Proof trees

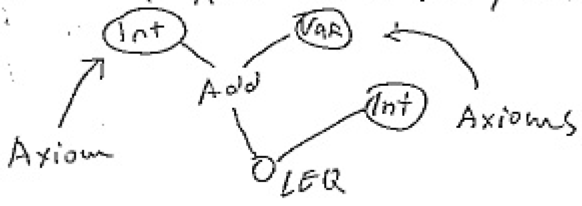
- tree where each node labelled by judgement, edges connect premises to conclusion according to some inference rule.
- leaves of tree are axioms
  → Inf rules w/o premises.

Ex) $\boxed{x : \text{int} \vdash 2 + x \le 10 : \text{bool}.}$

Pf. :

$$\dfrac{\dfrac{\text{Int}\ \dfrac{}{x:\text{Int}\vdash 2:\text{int}} \qquad \text{Var}\ \dfrac{}{x:\text{Int}\vdash x:\text{Int}}}{x:\text{Int}\vdash 2+x:\text{int}}\,\text{Add} \qquad \text{Int}\ \dfrac{}{x:\text{Int}\vdash 10:\text{int}}}{x:\text{Int}\vdash (2+x)\le 10:\text{bool}}\,\text{LEQ}$$

LEQ ⟶ Add ⟶ Int , Var



trees are bottom-up.

the leaves are axioms, $\boxed{\text{QED}}$

$\boxed{\vdash : \text{Turnstile, Syntactic entailment}}$

Another ex:

$$x : \text{int} \vdash \text{if } x \le 0 \text{ then } x \text{ else } -1 * x : \text{int}$$

...

$$\dfrac{\dfrac{}{x : \text{int} \vdash x \le 0 : \text{bool}}\,\text{Leq} \qquad \dfrac{}{}\,\text{Var} \qquad \dfrac{}{}\,\text{Mul}}{x : \text{int} \vdash \text{if } x \le 0 \text{ then } x \text{ else } -1 * x : \text{int}}\,\text{if}$$

Goal: given context Γ, expression e, type τ, determine if $\Gamma \vdash e : \tau$ exists.

Method: recurse on structure of AST, apply inference rules starting at the root of AST.
1-1 correspondence btw rules & cases in typechecker.

* produce constraint system $\longrightarrow$ then solve the constraints.

Scope logic (bindings)

Ocaml-like example

New rules:

LET:

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma\{x \mapsto t_1\} \vdash e_2 : t}{\Gamma \vdash (\text{let } x = e_1 \text{ in } e_2) : t}$$

(rule for "let" not "let rec")

FUN

$$\frac{\Gamma\{x \mapsto t_1\} \vdash e : t_2}{\Gamma \vdash \text{fun } (x : t_1) \to e : t_2 \to t_1}$$

APP

$$\frac{\Gamma \vdash e_1 : t_1 \to t_2 \quad \Gamma \vdash e_2 : t_1}{\Gamma \vdash e_1\ e_2 : t_2}$$

☆ rules are not syntax-directed, premises are not necessarily included in conclusion:

e.g. in LET, $(\Gamma \vdash e_1 : t_1)$ is not used in conclusion

type inference ☆

given Γ, e, determin $\exists\, t$ for which there is derivation for $\Gamma \vdash e : t$.

☆ requires backtracking sometimes).

☆ Recurse on structure of AST to produce constraint system $\longrightarrow$ then solve the constraints.

Type Soundness

Milner: "Well-typed programs do not go wrong"

Formally: if $\vdash e : t$ is derivable, then evaluating e either fails or returns type t.

Well-formed types

Need additional rules to define.

Judgements take form $H \vdash t$.

- H set of type names
- t a type
- $H \vdash t$ — "if H names well-formed types t is well-formed type."

$$\text{INT}\ \frac{}{H \vdash \text{Int}} \qquad \text{BOOL}\ \frac{}{H \vdash \text{bool}}$$

$$\text{Arrow}\ \frac{H \vdash t_1 \quad H \vdash t_2}{H \vdash t_1 \rightarrow t_2}$$

$$\text{NAMED}\ \frac{}{H \vdash s} \quad s \in H$$

Now modify existing type rules:

$$\text{FUN}\ \frac{H \vdash t_1 \quad H, \Gamma\{x \mapsto t_1\} \vdash e : t_2}{H, \Gamma \vdash \text{fun}(\ldots) \rightarrow \ldots}$$

Additional rules for well-formed statements

EX) Judgements take form $D; \Gamma; rt \vdash s$

D : type name $\longrightarrow$ definition

Γ : typeenv vars $\longrightarrow$ types.

rt : type

$D; \Gamma; rt \vdash s$

"w/ typedef D, assume typeenv Γ, s is valid statement within ctxt of a function tat returns type value rt.

ASSIGN : $$\frac{\Gamma \vdash e : \Gamma(x)}{D; \Gamma; rt \vdash x := e}$$

RETURN : $$\frac{\Gamma \vdash e : rt}{D; \Gamma; rt \vdash \text{return } e.}$$

DECL (skipped).

Additional topics

- In Ocaml, can have vars, types w/ same name
  - Multiple namespaces → Multiple environments, Symbol tables.
- Parametric polymorphism
  e.g. fun x -> x has 'a → 'a as type.
  (Hinley - Milner type inference)
  * finite representation of ∞ - many typings

- Subtyping (OO - languages), next time
  - casting, coersion