

# Lec 7

## Semantic Analysis

Recall IR tree & type judgement

Type judgement:  $\Gamma \vdash e : \tau$   
 context  $\Gamma ::= \bullet \mid \Gamma, x : \tau$

Rules  $\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$        $\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}}$

### # Type Checking

assign(x, x+5) requires:  $x : \text{int}$  (& x initialised) init checking can be indep from type checking

Type of statement — have to consider side effect & returns  
 write  $\Gamma \vdash s : [\tau]$   
statement      return type of func  
 "In context  $\Gamma$ , s is well-typed and all args of returns have type  $\tau$ "

return(s) requires: the func returns int

```
{
  return(s);
  return(true); ← bad, even if already returned
}
```

$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{return}(e) : [\tau]}$        $\frac{\Gamma \vdash s_1 : [\tau] \quad \Gamma \vdash s_2 : [\tau]}{\Gamma \vdash \text{seq}(s_1, s_2) : [\tau]}$  ← no decl to share btwn  $s_1$  &  $s_2$

$\frac{\Gamma \vdash e : \tau' \quad \Gamma(x) : \tau'}{\Gamma \vdash \text{assign}(x, e) : [\tau]}$        $\frac{\Gamma, x : \tau' \vdash s : [\tau]}{\Gamma \vdash \text{decl}(x, \tau', s) : [\tau]}$

$\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash s : [\tau]}{\Gamma \vdash \text{while}(e, s) : [\tau]}$        $\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash s_1 : [\tau] \quad \Gamma \vdash s_2 : [\tau]}{\Gamma \vdash \text{if}(e, s_1, s_2) : [\tau]}$

### # Implementing These — Modes of Judgements

chk\_exp: context → exp → type → bool? Nope, since we might not have concrete type as input yet

syn\_exp: context → exp → type

chk\_stmt: context → stmt → type → unit  
if not type check just reuse exprs

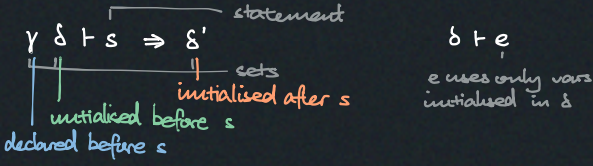
### # Initialisation Checking

Option 1: Use liveness, using these judgements  
given in language specs

- use(e, x)      x used in e
- def(s, x)      s defined in s
- live(s, x)      x live in s
- init(s)      all vars in s initialised

$\frac{\neg \text{live}(s, x) \quad \text{init}(s)}{\text{init}(\text{decl}(x, \tau, s))}$

Option 2: Alt judgements



$\frac{\delta \vdash e}{\forall \delta \vdash \text{assign}(x, e) \Rightarrow \delta \cup \{x\}}$

$\frac{\forall \delta \vdash s_1 \Rightarrow \delta_1 \quad \forall \delta \vdash s_2 \Rightarrow \delta_2}{\forall \delta \vdash \text{seq}(s_1, s_2) \Rightarrow \delta_2}$

$\frac{\delta \vdash e}{\forall \delta \vdash \text{return}(e) \Rightarrow \delta}$

### # Lexing — first pass to identify tokens

In: string source code  
 Out: token stream

```
int x;          TYPE(int)
// init        VAR(x)
// init        SEMI
x = n;         VAR(x)
              EQ
              VAR(n)
              SEMI
```

Lexer spec

1.  $\Sigma = \{0, \dots, 9, a, \dots, z, (, ), \dots\}$
2. Mapping  $\Sigma^* \rightarrow \text{TokenType}$   
usually by regex

Problem: mapping overlap, eg. if → IF or Ident (if)  
 → Longest match first  
 → If same len, take first one in table

Execution: generate finite automata, run it, backtrack to last accept state