Lec 7 Semantic Analysis

Recall IR tree & type judgement

Type judgement : $\Gamma \vdash e : \tau$

context $\Gamma ::= \cdot \mid \Gamma, x : \tau$

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

Type Checking

assign $(x, x+5)$ requires : $x : int$ (& x initialised)

init checking can be indep from type checking

Type of statement — have to conside side effect & returns

write $\Gamma \vdash \overset{\uparrow}{s} : [\overset{\uparrow}{\tau}]$

statement return type of func

"In context Γ, s is well-typed and all args of returns have type τ"

return(5) requires : the func returns int

```
{
   return(5)   ;
   return ( true ) ;   ← bad , even if already returned
}
```

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash return(e) : [\tau]} \qquad \frac{\Gamma \vdash s_1 : [\tau] \quad \Gamma \vdash s_2 : [\tau]}{\Gamma \vdash seq(s_1, s_2) : [\tau]}$$

← no decl to share btwn s_1 & s_2

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

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

Implementing These — Modes of Judgements

chk_exp : contex → 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 raise exns

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

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

Option 2 : Alt judgements

statement

$$\gamma\ \delta \vdash s \Rightarrow \delta'$$

sets

initialised after s

initialised before s

declared before s

$\delta \vdash e$

e uses only vars initialised in δ

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

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

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

Lexing — first pass to identify tkns

In : string source code
Out : token stream

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

Lexer spec

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

└ { TYPE, VAR, SEMI, … }

└ 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