Type checking/inference for a sequential language

SPA Ch 2/3

1. define C-like language
2. define a type inference algorithm

```
program     P → F F F ... F
function    F → X ( X, ..., X) { var X, ..., var X;
                                  S; S; .... ; S;
                                  return E }

statement   S → X = E
                | if (E) { S; ... ; S } else { S; ... ; S }
                | while (E) { S; ... ; S }
                | *X = E

expressions E → I      integer
                | X
                | E + E | E - E | E * E
                | E == E | E > E
                | X (E, ..., E)
                | null
                | &X | * E | alloc E
```

E.g.

```
iterate (n) {
    var f;
    f=1;
    while (n>0) {
        f= f*n; n=n-1;
    }
    return f;
}
```

What should our type system catch?

arithmetic operations are over int

conditions of if/while are int

"main" should only return int

* only appears to pointers

arguments to a function are of correct type

$::=$

$:=$

$$
\begin{aligned}
T \rightarrow\ & \mathrm{int} \\
\mid\ & \& T \\
\mid\ & (T, \ldots, T) \rightarrow T
\end{aligned}
$$

$T \rightarrow T \rightarrow T \rightarrow \ldots T$

E.g. int

&int

&&int

(int, int) → &int

Constraints

$S = T$

$⟦\cdot⟧$ type variable

$[\ \ ]$ for simplicity

e.g. $[1 + foo] = [x]$

$x = 1 + foo$

I e.g. 10 $[I] = int$

$[10] = int$

$E_1 == E_2$ $[E_1] = [E_2]$ and $[E_1 == E_2] = int$

$E_1 \text{ op } E_2$ $[E_1 \text{ op } E_2] = [E_1] = [E_2] = int$

arithmetic expression

if .. {$x = 10$}
else {$x = \& y$}

$[x] = [10] = int$

$[x] = \&[y] = \& int$

$X = E$ $[X] = [E]$

if (E) {$S \ldots S$} else {$S \ldots S$} $[E] = int$

while (E) .... $[E] = int$

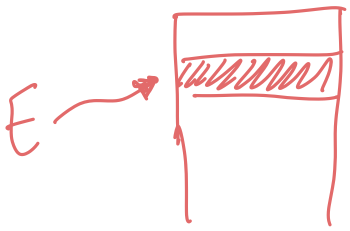
$X(X_1, \ldots, X_n)$ { .... return E } $[X] = ([X_1], \ldots [X_n]) \rightarrow [E]$

alloc E $[\text{alloc } E] = \& [E]$

$[\text{alloc } 10] = \& int$

$= \& [10]$

$[10] = int$



$\& X$ $[\& X] = \& [X]$

$*E$ $\underbrace{[E]} = \underbrace{\& [*E]}$

$\underbrace{[*E] = Y}$

equivalent

$$\boxed{\begin{array}{l}[E] = \& Y \\ [*E] = Y\end{array}}$$

→ E has to be an address

→ what is in this address

int?

Bool?

another address?

*X = E   [X] = &[E]

$$\boxed{\begin{array}{l} [X] = \& Y ; \\ [Y] = [E] \end{array}}$$

null   [null] = &Y

E.g.

```
Start(){
  var x,y,z;
  x=1;
  y=alloc x;
  *y = x;
  z = *y;
  return z;
}
```

Start : () → int

[Start] = () → [z]

[1] = int

[alloc x] = &[x]

[x] = [1]

[y] = [alloc x]

[y] = &[x]

[z] = [*y]

&[*y] = [y]

[x]=int

[y]=&int

[z]=int

E.g.

```
f(){
    var x;
    x = alloc 17;
    x = 42;
    return x+12;
}
```

[x] = & int

×

[x] = int

flow insensitive type system

e.g

```
baz() {
    var x;
    x = 1;
    return &x;
}
main() {
    var p
    p = baz()
    *p = 1
    return *p
}
```

baz : () → &int

[p] = &int

Rust

(lifetimes)