5118014 Principles of Programming Languages

Lecture 5. Identifiers

Shin Hong

Variable

- A variable relates a name to a value in a program
  - use the value by writing (calling) the related name
- An identifier is a name related to a certain entity in a program
  - variable names
  - function names
  - parameter names
  - class/method name
  - type/field name

Identifiers: Binding, Bound and Free

```
fn f(x: i32) -> i32 {
  let y = 2 ;
  x + y
}

fn main () {
  let x = f(z) ;
  println!(“{}”, x) ;
}
```

- binding occurrence
  - the identifier occurs to be defined
  - a binding occurrence relates the identifier to a particular entity
  - every binding occurrence has a scope
- bound occurrence
  - the identifier occurs to use the related entity
- free identifiers
  - neither binding nor bound

Scope

- a scope is a condition where the identifier is defined by the binding occurrence
  - static scoping: the condition is defined as a code region
  - dynamic scoping: the condition is defined as a period in an execution
- shadowing: innermost/last binding of an identifier shadows the outer/previous binding occurrences of the same identifier.
- Example

```
fn f (x: i32) -> i32 {
    let y = 1 ;
    if x < 0 {
        let x = 0 ;
        return x + y ;
    } else {
        let y = x ;
        return x + y ;
    }
}
```

VAE: Arithmetic Expr. with Immutable Variable

- add variables to AE
  - ex. `3 + 4`

```
     1 + (val x=1 in (val y=x+1 in (x + y)))
```

- update syntax

```
- <expr> ::= ⋯ | “val” <id> “=“ <expr> “in” <expr>
               | <id>
- <id> : r”[a-zA-Z][a-zA-Z0-9]*”
```

VAE: Semantics (1/3)

- an environment is a map (partial function) from identifiers to values
  - $Env = Id \rightharpoonup\!\!\mapsto \mathbb{Z}$
  - $\sigma \in Env$

- add environment as a factor of semantics function
  - $\Rightarrow \ \in Env \times E \rightarrow \mathbb{Z}$
  - $\Rightarrow \subseteq Env \times E \times \mathbb{Z}$
  - $(\sigma, e, n) \in \Rightarrow$ if and only if e evaluates to n under σ (i.e., $\sigma \vdash e \Rightarrow n$)

VAE: Semantics (2/3)

AE

$$n \Rightarrow n \quad [\textsc{Num}]$$

$$\frac{e_1 \Rightarrow n_1 \qquad e_2 \Rightarrow n_2}{e_1 + e_2 \Rightarrow n_1 +_{\mathbb{Z}} n_2} \quad [\textsc{Add}]$$

$$\frac{e_1 \Rightarrow n_1 \qquad e_2 \Rightarrow n_2}{e_1 - e_2 \Rightarrow n_1 -_{\mathbb{Z}} n_2} \quad [\textsc{Sub}]$$

VAE

$$\sigma \vdash n \Rightarrow n \quad [\textsc{Num}]$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 + e_2 \Rightarrow n_1 + n_2} \quad [\textsc{Add}]$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 - e_2 \Rightarrow n_1 - n_2} \quad [\textsc{Sub}]$$

VAE: Semantics (3/3)

```
<expr> ::= ”val” <id> “=“
           <expr> “in” <expr>
```

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma[x \mapsto n_1] \vdash e_2 \Rightarrow n_2}{\sigma \vdash \text{val } x{=}e_1 \text{ In } e_2 \Rightarrow n_2}$$

$$\sigma[x \mapsto n](x') = \begin{cases} n & \text{if } x = x' \\ \sigma(x') & \text{if } x \neq x' \end{cases}$$

```
<expr> ::= <id>
```

$$\frac{x \in Domain(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

Example

$$\dfrac{\emptyset \vdash 1 \Rightarrow 1 \qquad \dfrac{\dfrac{x \in \mathit{Domain}([x \mapsto 1])}{[x \mapsto 1] \vdash x \Rightarrow 1} \qquad \dfrac{x \in \mathit{Domain}([x \mapsto 1])}{[x \mapsto 1] \vdash x \Rightarrow 1}}{[x \mapsto 1] \vdash x + x \Rightarrow 2}}{\emptyset \vdash \text{val } x{=}1 \text{ in } x + x \Rightarrow 2}$$

VAE: Interpreter

```
std::collections::BtreeMap
https://doc.rust-lang.org/std/collections/struct.BTreeMap.html
```

```
use std::collections::BTreeMap ;
…
fn interp (e: Box<Expr>, env: &BTreeMap::<String, i32>) -> i32 {
    match *e {
        Op(l, Add, r) => interp(l, env) + interp(r, env),
        Op(l, Sub, r) => interp(l, env) - interp(r, env),
        Num(n) => n,
        Ref(id) => *env.get(&id).unwrap(),
        Val(id, v, e) => {
            let mut nenv = env.clone() ;
            nenv.insert(id, interp(v, env)) ;
            interp(e, &nenv)
        }
    }
}
```