CSCI 742 - Compiler Construction

Lecture 20

Name Analysis Implementation

Instructor: Hossein Hojjat

March 5, 2018

Recap: Name Analysis Goals

- For each declaration of identifier,
  identify where the identifier refers to
- Name analysis:
  - maps, partial functions (math)
  - environments (PL theory)
  - symbol table (implementation)
- Report some simple semantic errors
- We usually introduce symbols for things denoted by identifiers
- Symbol tables map identifiers to symbols

Notations for Maps

- Mathematical notation of map is a partial function $f : A \rightharpoonup B$ (that is a function from a subset of A to B)
  - $f \subseteq A \times B$
  - $\forall x.\forall y_1.\forall y_2.(x, y_1) \in f \wedge (x, y_2) \in f \to y_1 = y_2$

  We define $\mathit{dom}(f) = \{x \mid \exists y.(x, y) \in f\}$
- Sometimes we denote map $\{(k_1, v_1), \cdots, (k_n, v_n)\}$ by $\{k_1 \mapsto v_1, \cdots, k_n \mapsto v_n\}$
- The key operation is function update
  $f[k := v] = \{(x, y) \mid (x = k \wedge y = v) \vee (x \neq k \wedge (x, y) \in f)\}$
  If the value was defined before, now we redefine it
- A generalization of update is overriding one map by another
  $f \oplus g = \{(x, y) \mid (x, y) \in g \vee (x \notin \mathit{dom}(g) \wedge (x, y) \in f)\}$
- Is $f \oplus g = g \oplus f$?

Checking each variable is declared

- Environment (Symbol Table): $\Gamma = \{(x_1, T_1), \cdots, (x_n, T_n)\}$

identifier

symbol (type,...)

- $\Gamma \vdash e$: e uses only variables declared in Γ
- Example: if $\Gamma = \{(x, \texttt{int}), (y, \texttt{boolean}), (z, \texttt{int})\}$
  then

- $\Gamma \vdash (x + 5) - z$
- $\Gamma \vdash x = z + 1$ but
- $\Gamma \not\vdash x = w + 1$ as w is not declared in Γ

Checking each variable is declared

(Variable Use)
$$\frac{x \in \textit{dom}(\Gamma)}{\Gamma \vdash x}$$

$$\frac{x \in \textit{dom}(\Gamma) \qquad \Gamma \vdash e}{\Gamma \vdash x = e}$$

$$\frac{\Gamma \vdash e_1 \qquad \Gamma \vdash e_2}{\Gamma \vdash e_1 + e_2}$$

$$\frac{\Gamma \vdash e_1 \qquad \Gamma \vdash e_2}{\Gamma \vdash e_1 * e_2}$$

$$\frac{\Gamma \vdash s \qquad \Gamma \vdash \bar{s}}{\Gamma \vdash s; \bar{s}}$$

where s is statement
and $\bar{s}$ is a statement sequence

$$\frac{\Gamma[x := \texttt{int}] \vdash \bar{s}}{\Gamma \vdash (\texttt{int}\ \ x); \bar{s}}$$

Local block declarations change Γ

`int x = 0;`	$\Gamma = \{(\mathrm{x}, \mathrm{int})\}$
`{`	
`  int y = 0;`	
`  x = y - 1;`	$\Gamma = \{(\mathrm{x}, \mathrm{int}), (\mathrm{y}, \mathrm{int})\}$
`  {`	
`    boolean x = false;`	$\Gamma = \{(\mathrm{x}, \mathrm{boolean}), (\mathrm{y}, \mathrm{int})\}$
`    x = (y < 0);`	
`  }`	
`  x = x + 5;`	$\Gamma = \{(\mathrm{x}, \mathrm{int}), (\mathrm{y}, \mathrm{int})\}$
`}`	$\Gamma = \{(\mathrm{x}, \mathrm{int})\}$

Function definitions

$$\frac{\Gamma \oplus \{(x_1, T_1), \cdots, (x_n, T_n)\} \vdash \bar{s}}{\Gamma \vdash T\ m\ (T_1\ \ x_1, \cdots, T_n\ \ x_n)\{\bar{s}\}}$$

```
class World {
  int sum;
  int value;
  void add(int n) {
    sum = sum + n;
  }
}
```

Instantiating the inference rule:

$$\Gamma = \{(\texttt{sum}, \texttt{int})\ ,\ (\texttt{value}, \texttt{int})\}$$

$$\frac{\Gamma \oplus \{(\texttt{n}, \texttt{int})\} \vdash \texttt{sum = sum + n}}{\Gamma \vdash \texttt{void add(int n) }\{\texttt{sum = sum + n;}\}}$$

Symbol Table Γ Contents

What kind of information do we need to store for each identifier?

Variables (globals, fields, parameters, locals)

- Need to know types, positions - for error messages
- Later: memory layout
  - Example: To compile `x.f = y` into `memcopy(addr_y, addr_x+6, 4)`
  - 3rd field in an object should be stored at offset e.g. +6 from the address of the object
  - the size of data stored in x.f is 4 bytes
- Sometimes more information explicit: whether variable local or global

Classes, methods, functions

- Recursively have their own symbol tables

Implementation Approaches

- In Java, the standard model is a mutable graph of objects
- It seems natural to represent references to symbols using mutable fields (initially null, resolved during name analysis)
- Alternative way in functional languages:
  - store the backbone of the graph as a algebraic data type (immutable)
  - pass around a map linking from identifiers to their declarations

Functional: Different Points, Different Γ

```
class World {
  int sum;
  void add(int foo) {
    sum = sum + foo;
  }
  void sub(int bar) {
    sum = sum - bar;
  }
  int count;
}
```

$\Gamma_0 = \{(\texttt{sum}, \mathrm{int}), (\texttt{count}, \mathrm{int})\}$

$\Gamma_1 = \Gamma_0[\texttt{foo} := \texttt{int}]$

Γ_0

$\Gamma_1 = \Gamma_0[\texttt{bar} := \texttt{int}]$

Γ_0

Imperative Way: Push and Pop

```
class World {
  int sum;
  void add(int foo) {
    sum = sum + foo;
  }
  void sub(int bar) {
    sum = sum - bar;
  }
  int count;
}
```

$\Gamma_0 = \{(\texttt{sum}, \text{int}), (\texttt{count}, \text{int})\}$

$\Gamma_1 = \Gamma_0[\texttt{foo} := \texttt{int}]$
change table, record change

Γ_0 revert changes from table

$\Gamma_1 = \Gamma_0[\texttt{bar} := \texttt{int}]$
change table, record change

Γ_0 revert changes from table

Imperative Symbol Table

- Hash table, mutable `Map[ID, Symbol]`
- Example:
  - hash function into array
  - array has linked list storing (ID, Symbol) pairs
- **Undo Stack**: to enable entering and leaving scope
- Entering new scope (function, block):
  - add beginning-of-scope marker to undo stack
- Adding nested declaration (ID, sym)
  - lookup old value symOld, push old value to undo stack
  - insert (ID, sym) into table
- Leaving the scope
  - go through undo stack until the marker, restore old values