CS 412
Introduction to Compilers
Andrew Myers
Cornell University
Lecture 11: Static Semantics
16 Feb 01

Administration

• Programming Assignment 2 due in 1 week

Static Semantics

• Can describe the types used in a program. How to describe type checking?
• Formal description: static semantics for the programming language
• Is to type-checking as grammar is to parsing
• Static semantics defines types for all legal language ASTs
• We will write ordinary language syntax to mean the corresponding AST

Type Judgements

• Static semantics defines how to derive type judgments

Type Judgements

• Type judgment: \( A \vdash E : T \)
  – means “In the context \( A \) (symbol table), the expression \( E \) is a well-typed expression with the type \( T \)”
• Type context is set of type assignments

Deriving a judgment

if (b) 2 else 3 : int

• What do we need to decide that this is a well-typed expression of type int?

  • b must be an bool (b: bool)
  • 2 must be an int (2: int)
  • 3 must be an int (3: int)
Deriving a judgement

- To show
  \[ b: \text{bool}, x: \text{int} \vdash \text{if} (b) 2 \text{ else } x : \text{int} \]
- Need to show:
  \[ b: \text{bool}, x: \text{int} \vdash b : \text{bool} \]
  \[ b: \text{bool}, x: \text{int} \vdash 2 : \text{int} \]
  \[ b: \text{bool}, x: \text{int} \vdash x : \text{int} \]

General Rule

- For any environment \( A \), expression \( E \), statements \( S_1 \) and \( S_2 \), the judgment
  \[ A \vdash \text{if} (E) S_1 \text{ else } S_2 : T \]
  is true if
  \[ A \vdash E : \text{bool} \]
  \[ A \vdash S_1 : T \]
  \[ A \vdash S_2 : T \]

As an Inference Rule

Premises

\[ A \vdash E : \text{bool} \quad A \vdash S_1 : T \quad A \vdash S_2 : T \]

(\( \text{name} \))

Conclusion

- Holds for any choice of the syntactic meta-variables \( E \), \( S_1 \), \( S_2 \), \( T \)

Why inference rules?

- Inference rules: compact, precise language for specifying static semantics (can specify languages in \( \sim 20 \) pages vs. 100’s of pages of Java Language Specification)
- Inference rules correspond directly to recursive AST traversal that implements them
- Type checking is attempt to prove type judgments \( A \vdash E : T \) true by walking backward through rules

Meaning of Inference Rule

- Inference rule says: given that antecedent judgments are true
  – with some substitution for meta-variables \( A, E_1, E_2 \)
- Then, consequent judgment is true
  – with a consistent substitution

Proof Tree = Call graph

- Expression is well-typed if there exists a type derivation for a type judgment
- Type derivation is a proof tree
Implementing a rule

- Work backward from goal:

```java
class Add extends Expr {
    Expr e1, e2;
    Type typeCheck(SymTab A) {
        Type t1 = e1.typeCheck(A),
        t2 = e2.typeCheck(A);
        if (t1 == Int && t2 == Int) return Int;
        else throw new TypeCheckError("+");
    }
}
```

More about Inference Rules

- Rules are implicitly universally quantified over free variables
- No premises: axiom \( A \vdash \text{true} : \text{bool} \)
- Same goal judgment may be provable in more than one way
- Syntax-directed rules: can prove judgements without searching

\[
A \vdash E_1 : \text{float} \quad A \vdash E_2 : \text{int} \\
A \vdash E_1 + E_2 : \text{float} \quad A \vdash E_1 + E_2 : \text{float}
\]

While

- For statements that do not have a value, use the type `unit` to represent their result (\( \text{unit} = \) completed successfully)

\[
\frac{A \vdash E : \text{bool}}{A \vdash \text{while} (E) S : \text{unit}}
\quad (\text{while})
\]

If statements

- Iota: the value of an if statement (if any) is the value of the arm that is executed.
- If no else clause, no value:

\[
\frac{A \vdash E_1 : \text{bool}}{A \vdash \text{if} (E) S : \text{unit}}
\quad (\text{if})
\]

Assignment

- \(id : T \in A\)
- \(A \vdash E : T\)

\[
\frac{A \vdash \text{assign}}{A \vdash \text{array-assign}}
\]

\[
\frac{A \vdash E_1 : T}{A \vdash E_2 : \text{int}}
\frac{A \vdash E_3 : T}{A \vdash \text{array}[T]} \\
\frac{A \vdash E_1[E_2] = E_3 : T}{A \vdash \text{assign}}
\]

\[
\frac{A \vdash E_1 : \text{int}}{A \vdash \text{assign}}
\frac{A \vdash E_2 : \text{float}}{A \vdash \text{array-assign}}
\]

\[
\frac{A \vdash E_1 : \text{int}}{A \vdash \text{assign}}
\frac{A \vdash E_2 : \text{float}}{A \vdash \text{array-assign}}
\]

\[
\frac{A \vdash E_1 : \text{int}}{A \vdash \text{assign}}
\frac{A \vdash E_2 : \text{float}}{A \vdash \text{array-assign}}
\]

\[
\frac{A \vdash E_1 : \text{int}}{A \vdash \text{assign}}
\frac{A \vdash E_2 : \text{float}}{A \vdash \text{array-assign}}
\]

\[
\frac{A \vdash E_1 : \text{int}}{A \vdash \text{assign}}
\frac{A \vdash E_2 : \text{float}}{A \vdash \text{array-assign}}
\]