CSCI 742 - Compiler Construction

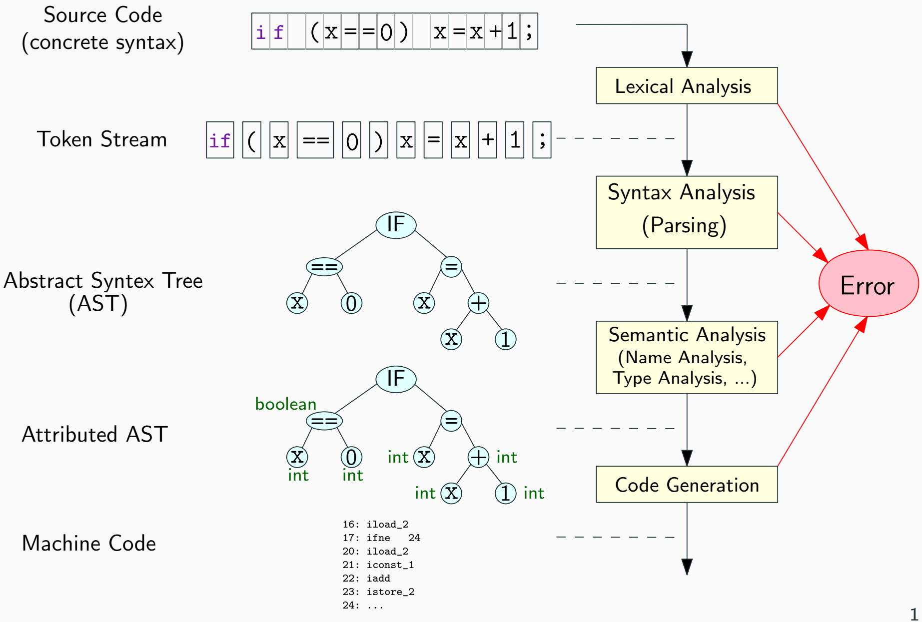
Lecture 21

Introduction to Type Checking

Instructor: Hossein Hojjat

March 19, 2018

Compiler Phases



Compiler Phases

- Type theory covers a huge range of topics
- Several lectures in the courses
  - Programming Language Concepts (344)
  - Programming Language Theory (740)
- In this course we do not cover the theoretical aspects of type system design
- We are mostly interested in type checking as a major component of the semantic analysis phase

What is a type?

- Type: a set of values and a set of operations on those values
- Example: Integers
- `int x,y;` means:
  - $\texttt{x},\texttt{y} \in [-2^{31}, 2^{31})$
  - Operations `+ - < <= mod` ... are possible on `x` and `y`
- Type errors:
  improper, type-inconsistent operations during program execution
- Type safety: absence of type errors at run time

How to Ensure Type-Safety?

Bind (assign) types, then check types

Type binding

- Defines types for constructs in the program (e.g., variables, functions)
- Can be either explicit (`boolean x`) or implicit (`x = false`)
- Type safety: correctness with respect to the type bindings

Type checking

- Static semantic checks to enforce the type safety of the program
- Enforce a set of type-checking rules

Type Check Examples

- Operators (such as +) receive the right types of operands
- User-defined functions receive the right types of operands
- LHS of an assignment should be "assignable"
- Variables are assigned the expected kinds of values
- Return statement must agree with return type
- Class members accessed appropriately

Static vs. Dynamic Typing

- **Statically** typed language: types are defined and checked at compile-time,
  and do not change during the execution of the program
- E.g., C, Java, Pascal
- **Dynamically** typed language: types defined and checked at run-time, during program execution
- E.g., Lisp, Scheme, Smalltalk

Why Static Checking?

- Efficient code: dynamic checks slow down the program
- Guarantees that all executions will be safe
- With dynamic checking, you never know when the next execution of the program will fail due to a type error

Drawbacks

- Adds an annotation burden for programmers
- Static type safety is a conservative approximation of the values that may occur during all possible executions
- It may reject some type-safe programs unfairly

Suitable Formalism

- We have used the following formal notations for specifying the first two phases of compiler:
  - Regular expressions for lexical analysis
  - Context-free grammars for parsers
- We use **inference systems** from logic to formalize type checking
  - Similar to what we did in name analysis
- Inference systems are suitable for performing computations of form:

If the first expression is of type T and
the second expression is of type T' then
the third expression must be of type T''

Background: Inference Systems

- Example inference rule:

All great universities have smart students	Premise 1
RIT is a great university	Premise 2
RIT has smart students	Conclusion

- Example inference rule:

e_1 has type `int`	Premise 1
e_2 has type `int`	Premise 2
$e_1 + e_2$ has type `int`	Conclusion

Background: Inference Systems

- An inference system has two parts:
  1. Definition of **Judgments**
     - Judgment: statement asserting a certain fact for an object
  2. Finite set of **Inference Rules**

- An inference rule has:
  1. a finite number of judgments P_1, P_2, $\cdots$, P_n as premises;
  2. a single judgment C as conclusion
- If a rule has no premises, it is called an **axiom**

$$\frac{P_1 \qquad P_2 \qquad \cdots \qquad P_n}{C}\ \text{(Rule name)}$$

Premises above the line (0 or more)
Conclusion below the line

Background: Inference Systems

Example: Use an inference system to define the set of even numbers

- Judgment: $\mathit{Even}(n)$ asserts that n is an even number
- Inference rules:

- Axiom:

$$\frac{}{\mathit{Even}(0)}\ (\mathsf{Even0})$$

- Successor Rule:

$$\frac{\mathit{Even}(n)}{\mathit{Even}(n+2)}\ (\mathsf{EvenS})$$

Derivation Tree

$$\frac{}{\textit{Even}(0)}\ (\text{Even0}) \qquad\qquad \frac{\textit{Even}(n)}{\textit{Even}(n+2)}\ (\text{EvenS})$$

- To derive more judgments we create **trees** of inference rules

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\textit{Even}(0)}\ (\text{Even0})}{\textit{Even}(2)}\ (\text{EvenS})}{\textit{Even}(4)}\ (\text{EvenS})}{\textit{Even}(6)}\ (\text{EvenS})$$

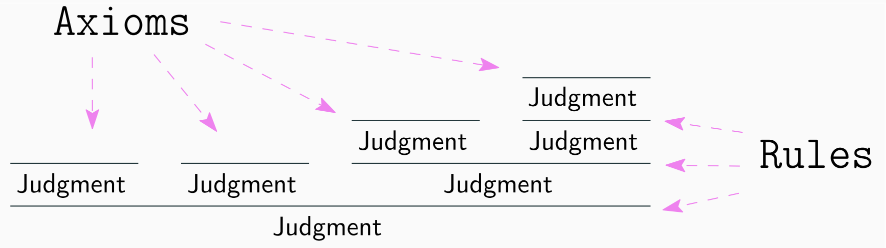
Derivation Tree

$$\frac{}{\textit{Even}(0)}\ (\text{Even0}) \qquad \frac{\textit{Even}(n)}{\textit{Even}(n+2)}\ (\text{EvenS})$$

- To derive more judgments we create **trees** of inference rules

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\textit{Even}(0)}\ (\text{Even0})}{\textit{Even}(2)}\ (\text{EvenS})}{\textit{Even}(4)}\ (\text{EvenS})}{\textit{Even}(6)}\ (\text{EvenS})$$

- Does $\textit{Even}(1)$ hold?
- No, because there exists no possible derivation


Axioms
Judgment
Judgment
Judgment
Judgment
Rules
Judgment
Judgment
Judgment
Judgment

Example: Less-than

Example: Use an inference system to define the less-than relation

- Judgment: $n < m$ asserts that n is smaller than m
- Inference rules:

- Axiom:

$$\frac{}{n < n + 1} \text{ (Suc)}$$

- Transitivity Rule:

$$\frac{k < n \qquad n < m}{k < m} \text{ (Trans)}$$

Exercise: Prove $0 < 3$.

Type Judgments and Type Rules

- e type checks to T under Γ (type environment)

$$\Gamma \vdash e : T$$

  - Types of constants are predefined
  - Type binding: types of variables are specified in the source code

- If e is composed of sub-expressions

$$\frac{\Gamma \vdash e_1 : T_1 \qquad \cdots \qquad \Gamma \vdash e_n : T_n}{\Gamma \vdash e : T}$$

Type Judgments and Type Rules

$$\boxed{\Gamma \vdash e : T}$$

If the (free) variables of e have types given by the type environment gamma, then e (correctly) type checks and has type T

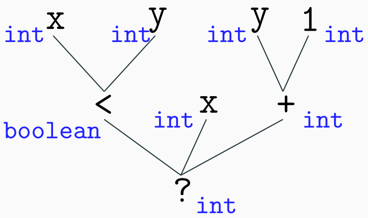
$$\boxed{\dfrac{\Gamma \vdash e_1 : T_1 \qquad \cdots \qquad \Gamma \vdash e_n : T_n}{\Gamma \vdash e : T}}$$

If e_1 type checks in Γ and has type T_1
and ...
and e_n type checks in Γ and has type T_n
then e type checks in Γ and has type T

Type Rules with Environment

$$\left.\begin{array}{l}\texttt{int x;}\\ \texttt{int y;}\end{array}\right\}\ \text{Type Environment } \Gamma$$

```
(x<y) ? x : (y + 1)
```



Type Rules:

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x : T}$$

$$\frac{}{\textit{IntConst}(k) : \texttt{int}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash (e_1 < e_2) : \texttt{boolean}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash (e_1 + e_2) : \texttt{int}}$$

$$\frac{\Gamma \vdash b : \texttt{boolean} \qquad \Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : T}{\Gamma \vdash (b \ ?\ e_1 \ :\ e_2) \ :\ T}$$