Rules of Inference

- · We have seen two examples of formal notation specifying parts of a compiler
 - Regular expressions
 - Context-free grammars
- · The appropriate formalism for type checking is logical rules of inference

Prof. Aiken CS 143 Lecture 9

37

Why Rules of Inference?

- · Inference rules have the form If Hypothesis is true, then Conclusion is true
- · Type checking computes via reasoning If E_1 and E_2 have certain types, then E_3 has a certain type
- Rules of inference are a compact notation for "If-Then" statements

Prof. Aiken CS 143 Lecture 9

38

40

From English to an Inference Rule

- · The notation is easy to read with practice
- · Start with a simplified system and gradually add features
- · Building blocks
 - Symbol \wedge is "and"
 - Symbol \Rightarrow is "if-then"
 - x:T is "x has type T"

Prof. Aiken CS 143 Lecture 9

From English to an Inference Rule (2)

If e_1 has type Int and e_2 has type Int, then $e_1 + e_2$ has type Int

(e_1 has type Int \wedge e_2 has type Int) \Rightarrow $e_1 + e_2$ has type Int

(e₁: Int \wedge e₂: Int) \Rightarrow e₁ + e₂: Int

Prof. Aiken CS 143 Lecture 9

From English to an Inference Rule (3)

The statement

(e₁: Int \wedge e₂: Int) \Rightarrow e₁ + e₂: Int

is a special case of

 $\mathsf{Hypothesis}_1 \land \ldots \land \mathsf{Hypothesis}_n \Rightarrow \mathsf{Conclusion}$

This is an inference rule

Prof. Aiken CS 143 Lecture 9

41

Notation for Inference Rules

· By tradition inference rules are written

Hypothesis, ... Hypothesis, Conclusion

· Cool type rules have hypotheses and conclusions

`e:T

means "it is provable that . . . "

Prof. Aiken CS 143 Lecture 9

42

Two Rules

$$e_1$$
: Int
$$e_2$$
: Int
$$e_1 + e_2$$
: Int

43

45

Prof. Aiken CS 143 Lecture 9

Two Rules (Cont.)

- These rules give templates describing how to type integers and + expressions
- By filling in the templates, we can produce complete typings for expressions

Prof. Aiken CS 143 Lecture 9

44

Example: 1 + 2

Prof. Aiken CS 143 Lecture 9

Soundness

- · A type system is sound if
 - Whenever `e:T
 - Then e evaluates to a value of type T
- · We only want sound rules
 - But some sound rules are better than others:

i is an integer i:Object

Prof. Aiken CS 143 Lecture 9

Type Checking Proofs

- · Type checking proves facts e: T
 - Proof is on the structure of the AST
 - Proof has the shape of the AST
 - One type rule is used for each AST node
- In the type rule used for a node e:
 - Hypotheses are the proofs of types of e's subexpressions
 - Conclusion is the type of e
- Types are computed in a bottom-up pass over the AST

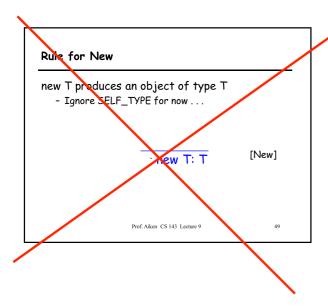
Prof. Aiken CS 143 Lecture 9

Rules for Constants

`false: Bool [Bool]

s is a string constant
 `s: String
 [String]

Prof. Aiken CS 143 Lecture 9 48





Si noti che, qui, il tipo generico T di e2 non viene usato e che il tipo

[Not]

e: Bool

- ¬e: Bool

`e1: Bool

A Problem

· What is the type of a variable reference?

51

· The local, structural rule does not carry enough information to give x a type.

Prof. Aiken CS 143 Lecture 9

A Solution

Two More Rules

Put more information in the rules!

del while è sempre Object

- · A type environment gives types for free variables
 - A type environment is a function from ovvero: nome id -> tipo ObjectIdentifiers to Types
 - A variable is free in an expression if it is not defined within the expression

Prof. Aiken CS 143 Lecture 9 52

Type Environments

Let O be a function from ObjectIdentifiers to Types

The sentence

is read: Under the assumption that variables have the types given by O, it is provable that the expression e has the type T

Prof. Aiken CS 143 Lecture 9

Possiamo quindi pensare che O rappresenti la lista di tabelle dei simboli nel punto in cui consideriamo l'espressione e. Il tipo T per e viene calcolato consultando O per tutti i nomi id presenti in O e tenendo conto degli operatori coinvolti (per esempio e potrebbe essere x + y)

Modified Rules

The type environment is added to the earlier rules:

$$\begin{array}{c} O \cdot e_1 : \text{ Int} \\ \hline O \cdot e_2 : \text{ Int} \\ \hline O \cdot e_1 + e_2 : \text{ Int} \end{array} \hspace{0.5cm} \text{[Add]}$$

54

Per esempio, dato x : 1; e1 = x+2; e2 = x*3; s=e1+e2; Dalla dichiarazione x : 1 in poi, O sarà data dalla sola associazione (x |-> Int) La regola si legge così: se è possibile provare che e1 ed e2 sono Int nello stesso type environment O (ed in questo esempio lo sono) allora è dimostrato che anche la somma e1+ e2 è Int *utilizzando lo stesso type environment O^* .