

# Softwaretechnik

## Lecture 20: Types and Type Soundness

Peter Thiemann

University of Freiburg, Germany

SS 2011

## Types and Type Correctness

- ▶ Large software systems: many people involved
  - ▶ project manager, designer, programmer, tester, ...
- ▶ Essential: divide into components with clear defined interfaces and specifications
  - ▶ How to divide the problem?
  - ▶ How to divide the work?
  - ▶ How to divide the tests?
- ▶ Problems
  - ▶ Are suitable libraries available?
  - ▶ Do the components match each other?
  - ▶ Do the components fulfill their specification?

## Table of Contents

### Types and Type Correctness

J AUS: Java-Expressions (Ausdrücke)  
Evaluation of Expressions  
Type correctness  
Result

## Requirements

- ▶ Programming language/environment has to ensure:
  - ▶ each component implements its interfaces
  - ▶ the implementation fulfills the specification
  - ▶ each component is used correctly
- ▶ Main problem: meet the interfaces and specifications
  - ▶ Minimal interface: **management of names**  
Which operations does the component offer?
  - ▶ Minimal specification: **types**  
Which types do the arguments and the result of the operations have?
  - ▶ See interfaces in Java

## Questions

- ▶ Which kind of security do types provide?
- ▶ Which kind of errors can be detected by using types?
- ▶ How do we provide type safety?
- ▶ How can we formalize type safety?

## Correct and Incorrect Expressions

- ▶ type correct expressions

```
boolean flag;
...
    0
    true
    17+4
    !flag
```

- ▶ expressions with type errors

```
int rain_since_April20;
boolean flag;
...
    !rain_since_April20
    flag+1
    17+(!false)
    !(2+3)
```

## JAUS: Java-Expressions (Ausdrücke)

### Grammar for a subset of Java expressions

$x ::= \dots$	variables
$n ::= 0 \mid 1 \mid \dots$	numbers
$b ::= \text{true} \mid \text{false}$	truth values
$e ::= x \mid n \mid b \mid e+e \mid !e$	expressions

## Typing Rules

- ▶ For each kind of expression a typing rule defines
  - ▶ if an expression is type correct and
  - ▶ how to obtain the result type of the expression from the types of the subexpressions.
- ▶ Five kinds of expressions
  - ▶ Constant numbers have type `int`.
  - ▶ Truth values have type `boolean`.
  - ▶ The expression  $e_1+e_2$  has type `int`, if  $e_1$  and  $e_2$  have type `int`.
  - ▶ The expression  $!e$  has type `boolean`, if  $e$  has type `boolean`.
  - ▶ A variable  $x$  has the type, with which it was declared.

## Formalization of “Type Correct Expressions”

### The Language of Types

$$t ::= \text{int} \mid \text{boolean} \quad \text{types}$$

Typing judgment: expression  $e$  has type  $t$

$$\vdash e : t$$

## Example: Typing Rules for JAUS

- ▶ A number  $n$  has type `int`.

$$\text{(INT)} \frac{}{\vdash n : \text{int}}$$

- ▶ A truth value has type `boolean`.

$$\text{(BOOL)} \frac{}{\vdash b : \text{boolean}}$$

- ▶ An expression  $e_1 + e_2$  has type `int` if  $e_1$  and  $e_2$  has type `int`.

$$\text{(ADD)} \frac{\vdash e_1 : \text{int} \quad \vdash e_2 : \text{int}}{\vdash e_1 + e_2 : \text{int}}$$

- ▶ An expression `!e` has type `boolean`, if  $e$  has type `boolean`.

$$\text{(NOT)} \frac{\vdash e : \text{boolean}}{\vdash !e : \text{boolean}}$$

## Formalization of “Typing Rules”

- ▶ A typing judgment is **valid**, if it is derivable according to the **typing rules**.
- ▶ To infer a valid typing judgment  $J$  we use a **deduction system**.
- ▶ A deduction system consists of a set of typing judgments and a set of typing rules.
- ▶ A typing rule (*inference rule*) is a pair  $(J_1 \dots J_n, J_0)$  which consists of a list of judgments (*assumptions*,  $J_1 \dots J_n$ ) and a judgment (*conclusion*,  $J_0$ ) that is written as

$$\frac{J_1 \dots J_n}{J_0}$$

- ▶ If  $n = 0$ , a rule  $(\varepsilon, J_0)$  is an *axiom*.

## Derivation Trees and Validity

- ▶ A judgment  $J$  is *valid* if a derivation tree for  $J$  exists.
- ▶ A derivation tree for the judgment  $J$  is defined by

1.  $\frac{}{J}$ , if  $\frac{}{J}$  is an axiom

2.  $\frac{\mathcal{J}_1 \dots \mathcal{J}_n}{J}$ , if  $\frac{J_1 \dots J_n}{J}$  is a rule and each  $\mathcal{J}_k$  is a derivation tree suitable for  $J_k$ .

## Example: Derivation Trees

- ▶ (INT)  $\frac{}{\vdash 0 : \text{int}}$  is a derivation tree for judgment  $\vdash 0 : \text{int}$ .
- ▶ (BOOL)  $\frac{}{\vdash \text{true} : \text{boolean}}$  is a derivation tree for  $\vdash \text{true} : \text{boolean}$ .
- ▶ The judgment  $\vdash 17 + 4 : \text{int}$  holds, because of the derivation tree

$$\text{(ADD)} \frac{\text{(INT)} \frac{}{\vdash 17 : \text{int}} \quad \text{(INT)} \frac{}{\vdash 4 : \text{int}}}{\vdash 17 + 4 : \text{int}}$$

## Variable

- ▶ Programs declare variables
- ▶ Programs use variables according to their declaration
- ▶ Declarations are collected in a *type environment*.

$$A ::= \emptyset \mid A, x : t \quad \text{type environment}$$

- ▶ An extended typing judgment contains a type environment: The expression  $e$  has the type  $t$  in the type environment  $A$ .

$$A \vdash e : t$$

- ▶ typing rule for variables:  
A variable has the type, with which it is declared.

$$\text{(VAR)} \frac{x : t \in A}{A \vdash x : t}$$

## Extension of the Remaining Typing Rules

- ▶ The typing rules propagate the environment.

$$\text{(INT)} \frac{}{A \vdash n : \text{int}}$$

$$\text{(BOOL)} \frac{}{A \vdash b : \text{boolean}}$$

$$\text{(ADD)} \frac{A \vdash e_1 : \text{int} \quad A \vdash e_2 : \text{int}}{A \vdash e_1 + e_2 : \text{int}}$$

$$\text{(NOT)} \frac{A \vdash !e : \text{boolean}}{A \vdash e : \text{boolean}}$$

## Example: Derivation with Variable

The declaration `boolean flag;` matches the type assumption

$$A = \emptyset, \text{flag} : \text{boolean}$$

Hence

$$\frac{\text{flag} : \text{boolean} \in A}{\frac{}{A \vdash \text{flag} : \text{boolean}}}{A \vdash ! \text{flag} : \text{boolean}}$$

## Intermediate Result

- ▶ Formal system for
  - ▶ syntax of expressions and types (CFG, BNF)
  - ▶ type judgments
  - ▶ validity of type judgments
- ▶ Open questions
  - ▶ How to evaluate expressions?
  - ▶ Coherence between evaluation and type judgments

## Evaluation of Expressions

## Approach: Syntactic Rewriting

- ▶ Define a binary **reduction relation**  $e \longrightarrow e'$  over expressions
- ▶  $e$  is in relation to  $e'$  ( $e \longrightarrow e'$ ) if one computational step leads from  $e$  to  $e'$ .
- ▶ Example:
  - ▶  $5+2 \longrightarrow 7$
  - ▶  $(5+2)+14 \longrightarrow 7+14$

## Result of Computations

- ▶ A value  $v$  is a number or a truth value.
- ▶ An expression can reach a value in many steps:
  - ▶ 0 steps: 0
  - ▶ 1 step:  $5+2 \longrightarrow 7$
  - ▶ 2 steps:  $(5+2)+14 \longrightarrow 7+14 \longrightarrow 21$
- ▶ but
  - ▶  $!4711$
  - ▶  $1+false$
  - ▶  $(1+2)+false \longrightarrow 3+false$
- ▶ These expressions cannot perform a reduction step. They correspond to run-time errors.
- ▶ Observation: these errors are type errors!

## Formalization: Results and Reduction Steps

- ▶ A value is a number or a truth value.

$$v ::= n \mid b \quad \text{values}$$

- ▶ One reduction step

- ▶ If the two operands are numbers, we can add the two numbers to obtain a number as result.

$$(B-ADD) \frac{}{\lceil n_1 \rceil + \lceil n_2 \rceil \longrightarrow \lceil n_1 + n_2 \rceil}$$

$\lceil n \rceil$  stands for the syntactic representation of the number  $n$ .

- ▶ If the operand of a negation is a truth value, the negation can be performed.

$$(B-TRUE) \frac{}{\lceil \text{true} \rceil \longrightarrow \lceil \text{false} \rceil} \quad (B-FALSE) \frac{}{\lceil \text{false} \rceil \longrightarrow \lceil \text{true} \rceil}$$

## Variable

- ▶ An expression that contains variables cannot be evaluated with the reduction steps.
- ▶ Eliminate variables with **substitution**, *i.e.*, replace each variable with a value. Then reduction can proceed.
- ▶ Applying a substitution  $[v_1/x_1, \dots, v_n/x_n]$  to an expression  $e$ , written as

$$e[v_1/x_1, \dots, v_n/x_n]$$

changes in  $e$  each occurrence of  $x_i$  to the corresponding value  $v_i$ .

- ▶ Example:

- ▶  $(\lceil \text{!flag} \rceil)[\lceil \text{false} \rceil/\lceil \text{flag} \rceil] \equiv \lceil \text{!false} \rceil$
- ▶  $(\lceil \text{m+n} \rceil)[\lceil 25 \rceil/\lceil \text{m} \rceil, \lceil 17 \rceil/\lceil \text{n} \rceil] \equiv \lceil 25+17 \rceil$

## Formalization: Nested Expressions

What happens if the operands of operations are not values? Evaluate the subexpressions first.

- ▶ Negation

$$(B-NEG) \frac{e \longrightarrow e'}{\lceil \text{!}e \rceil \longrightarrow \lceil \text{!}e' \rceil}$$

- ▶ Addition, first operand

$$(B-ADD-L) \frac{e_1 \longrightarrow e'_1}{e_1 + e_2 \longrightarrow e'_1 + e_2}$$

- ▶ Addition, second operand (only evaluate the second, if the first is a value)

$$(B-ADD-R) \frac{e \longrightarrow e'}{v + e \longrightarrow v + e'}$$

## Type Correctness Informally

- ▶ Type correctness: If there exists a type for an expression  $e$ , then  $e$  evaluates to a value in a finite number of steps.
- ▶ In particular, no run-time error happens.
- ▶ For the language JAUS the converse also holds (this is not correct in general, like in full Java).
- ▶ Prove in two steps (after Wright and Felleisen)  
Assume  $e$  has a type, then it holds:  
**Progress:** Either  $e$  is a value or there exists a reduction step for  $e$ .  
**Preservation:** If  $e \longrightarrow e'$ , then  $e'$  and  $e$  have the same types.

## Progress

If  $\vdash e : t$  is derivable, then  $e$  is a value or there exists  $e'$  with  $e \longrightarrow e'$ .

### Proof

Induction over the derivation tree of  $\mathcal{J} \models e : t$ .

If (INT)  $\frac{}{\vdash n : \text{int}}$  is the final step of  $\mathcal{J}$ , then  $e \equiv n$  is a **value** (and  $t \equiv \text{int}$ ).

If (BOOL)  $\frac{}{\vdash b : \text{boolean}}$  is the last step of  $\mathcal{J}$ , then  $e \equiv b$  is a **value** (and  $t \equiv \text{boolean}$ ).

### Progress: Negation

If (NOT)  $\frac{\vdash e_1 : \text{boolean}}{\vdash !e_1 : \text{boolean}}$  is the last step of  $\mathcal{J}$ , it holds that  $e \equiv !e_1$  and  $t \equiv \text{boolean}$  and  $\vdash e_1 : \text{boolean}$  is derivable.

Using the induction hypothesis ( $e_1$  is a value or there exists  $e'$  with  $e \longrightarrow e'$ ) there are two cases.

- ▶ In the case that  $e_1 \longrightarrow e'_1$ , we conclude that there exists  $e'$  with  $e \longrightarrow e'$  using rule (B-NEG).
- ▶ If  $e_1 \equiv v$  is a value, it's easy to prove that  $v$  is a truth value. Hence, we can apply the rule (B-TRUE) or (B-FALSE).

QED

## Progress: Addition

If (ADD)  $\frac{\vdash e_1 : \text{int} \quad \vdash e_2 : \text{int}}{\vdash e_1 + e_2 : \text{int}}$  is the final step of  $\mathcal{J}$ , then it holds that  $e \equiv e_1 + e_2$  and  $t \equiv \text{int}$ . Moreover, it is derivable that  $\vdash e_1 : \text{int}$  and  $\vdash e_2 : \text{int}$ . The induction hypothesis tells us that  $e_1$  is a value or there exists an  $e'_1$  with  $e_1 \longrightarrow e'_1$ .

- ▶ If  $e_1 \longrightarrow e'_1$  holds, we obtain that  $e \equiv e_1 + e_2 \longrightarrow e' \equiv e'_1 + e_2$  cause of rule (B-ADD-L). This is the desired result.
- ▶ In the case  $e_1 \equiv v_1$  is a value, we concentrate on  $\vdash e_2 : \text{int}$ . The induction hypothesis says that  $e_2$  is either a value or there exists an  $e'_2$  with  $e_2 \longrightarrow e'_2$ .
  - ▶ In the second case we can use rule (B-ADD-R) and get:  $e \equiv v_1 + e_2 \longrightarrow e' \equiv v_1 + e'_2$ .
  - ▶ In the first case ( $e_2 \equiv v_2$ ), we can prove easily that  $v_1 \equiv n_1$  and  $v_2 \equiv n_2$  are both numbers. Hence, we can apply the rule (B-ADD) and obtain the desired  $e'$ .

## Preservation

If  $\vdash e : t$  and  $e \longrightarrow e'$ , then  $\vdash e' : t$ .

### Proof

Induction on the derivation  $e \longrightarrow e'$ .

If (B-ADD)  $\frac{}{[n_1] + [n_2] \longrightarrow [n_1 + n_2]}$  is the reduction step, then it holds that  $t \equiv \text{int}$  because of (ADD). We can apply (INT) to  $e' = [n_1 + n_2]$  and obtain the desired result  $\vdash [n_1 + n_2] : \text{int}$ .

If (B-TRUE)  $\frac{}{! \text{true} \longrightarrow \text{false}}$  is the reduction step it holds that  $t \equiv \text{boolean}$  because of (NOT). We can apply (BOOL) to  $e' = \text{false}$  and get the desired result  $\vdash \text{false} : \text{boolean}$ .

The case for rule B-FALSE is analogous.

### Preservation: Addition

If (B-ADD-L)  $\frac{e_1 \longrightarrow e'_1}{e_1 + e_2 \longrightarrow e'_1 + e_2}$  is the occasion for the last step, we obtain through  $\vdash e : t$  that

$$(ADD) \frac{\vdash e_1 : \text{int} \quad \vdash e_2 : \text{int}}{\vdash e_1 + e_2 : \text{int}}$$

holds with  $e \equiv e_1 + e_2$  and  $t \equiv \text{int}$ .

From  $\vdash e_1 : \text{int}$  and  $e_1 \longrightarrow e'_1$  it follows by induction that  $\vdash e'_1 : \text{int}$  holds. Another application of (ADD) on  $\vdash e'_1 : \text{int}$  and  $\vdash e_2 : \text{int}$  yields  $\vdash e'_1 + e_2 : \text{int}$ .

The case of rule (B-ADD-R) is analogous.

### Preservation: Negation

If (B-NEG)  $\frac{e_1 \longrightarrow e'_1}{!e_1 \longrightarrow !e'_1}$  is the occasion for the last step, we get through  $\vdash e : t$ , that

$$(NOT) \frac{\vdash e_1 : \text{boolean}}{\vdash !e_1 : \text{boolean}}$$

holds with  $e \equiv !e_1$  and  $t \equiv \text{boolean}$ .

From  $\vdash e_1 : \text{boolean}$  and  $e_1 \longrightarrow e'_1$  we conclude (using induction) that  $\vdash e'_1 : \text{boolean}$  holds. Another application of rule (NOT) to  $\vdash e'_1 : \text{boolean}$  yields  $\vdash !e'_1 : \text{boolean}$ .

QED

## Elimination of Variables by Substitution

### Intention

If  $x_1 : t_1, \dots, x_n : t_n \vdash e : t$  and  $\vdash v_i : t_i$  (for all  $i$ ), then it holds  $\vdash e[v_1/x_1, \dots, v_n/x_n] : t$ .

### Assertion

If  $A', x_0 : t_0 \vdash e : t$  and  $A' \vdash e_0 : t_0$ , then it holds  $A' \vdash e[e_0/x_0] : t$ .

### Prove

Induction over derivation of  $A \vdash e : t$  with  $A \equiv A', x_0 : t_0$ .

If (VAR)  $\frac{x : t \in A}{A \vdash x : t}$  is the last step of the derivation, there are two

cases: Either  $x \equiv x_0$  or not.

If  $x \equiv x_0$  holds, then  $e[e_0/x_0] \equiv e_0$ . Because of the rule (VAR) it holds  $t \equiv t_0$ . Hence it holds  $A' \vdash e_0 : t_0$  (use the assumption).

If  $x \not\equiv x_0$ , then  $e[e_0/x_0] \equiv e$  and it holds  $x : t \in A'$ . Due to (VAR) it holds  $A' \vdash e : t$ .

### Substitution: Constants

If (INT)  $\frac{}{A \vdash n : \text{int}}$  is the last step, it holds (INT)  $\frac{}{A' \vdash n : \text{int}}$ .

If (BOOL)  $\frac{}{A \vdash b : \text{boolean}}$  is the last step, it holds

$$(BOOL) \frac{}{A' \vdash b : \text{boolean}}.$$



### Substitution: Addition

If (ADD)  $\frac{A \vdash e_1 : \text{int} \quad A \vdash e_2 : \text{int}}{A \vdash e_1 + e_2 : \text{int}}$  is the last step, then the induction hypothesis yields  $A' \vdash e_1[e_0/x_0] : \text{int}$  and  $A' \vdash e_2[e_0/x_0] : \text{int}$ . Apply rule (ADD) yields  $A' \vdash (e_1 + e_2)[e_0/x_0] : \text{int}$ .

### Substitution: Negation

If (NOT)  $\frac{A \vdash e_1 : \text{boolean}}{A \vdash !e_1 : \text{boolean}}$  is the last step, the induction hypothesis yields  $A' \vdash e_1[e_0/x_0] : \text{boolean}$ . Apply rule (NOT) yields  $A' \vdash (!e_1)[e_0/x_0] : \text{boolean}$ .

QED

## Theorem: Type Soundness of JAUS

- ▶ If  $\vdash e : t$ , then there exists a value  $v$  with  $\vdash v : t$  and reduction steps

$$e_0 \longrightarrow e_1, e_1 \longrightarrow e_2, \dots, e_{n-1} \longrightarrow e_n$$

with  $e \equiv e_0$  and  $e_n \equiv v$ .

- ▶ If  $e$  contains variables, then we have to substitute them with suitable values (choose values with same types as the variables).