Software Engineering

Lecture 17: Types and Type Soundness

Peter Thiemann

University of Freiburg, Germany

08.07.2013

Table of Contents

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?

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?

JAUS: Java-Expressions (Ausdrücke)

Grammar for a subset of Java expressions

$$
\begin{array}{lcll}
x & ::= & \ldots & \text{variables} \\
n & ::= & \mathtt{0} \mid \mathtt{1} \mid \ldots & \text{numbers} \\
b & ::= & \mathtt{true} \mid \mathtt{false} & \text{truth values} \\
e & ::= & x \mid n \mid b \mid e{+}e \mid !e & \text{expressions}
\end{array}
$$

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)
```

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

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

Typing judgment: expression e has type t

$$\vdash e : t$$

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 (ε, J_0) is an *axiom*.

Example: Typing Rules for JAUS

- A number n has type `int`.

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

- A truth value has type `boolean`.

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

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

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

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

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

Derivation Trees and Validity

- A judgment J is *valid* if a derivation tree for J exists.
- **Definition**: A *derivation tree for the judgment J* is either
  1. J, if J is an instance of an axiom, or
  2. $\dfrac{\mathcal{J}_1 \dots \mathcal{J}_n}{J}$, if $\dfrac{J_1 \dots J_n}{J}$ is an instance of a rule and each $\mathcal{J}_k$ is a derivation tree for J_k.

Example: Derivation Trees

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

$$\dfrac{(\mathrm{ADD})\quad \dfrac{(\mathrm{INT})}{\vdash 17 : \mathtt{int}} \qquad \dfrac{(\mathrm{INT})}{\vdash 4 : \mathtt{int}}}{\vdash 17 + 4 : \mathtt{int}}$$

Variable

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

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

- An open 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.

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

Extension of the Remaining Typing Rules

- The typing rules propagate the typing environment.

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

$$\text{(BOOL)}\quad A \vdash b : \texttt{int}$$

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

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

Example: Derivation with Variable

The declaration `boolean flag;` matches the type assumption

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

Hence the derivation

$$\cfrac{\cfrac{\texttt{flag} : \texttt{boolean} \in A}{A \vdash \texttt{flag} : \texttt{boolean}}}{A \vdash \,!\, \texttt{flag} : \texttt{boolean}}$$

Intermediate Result

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

Evaluation of Expressions

Approach: Syntactic Rewriting

- Define a binary **reduction relation** $e \longrightarrow e'$ over expressions
- Expression e *reduces in one step to* e' (Notation: $e \longrightarrow e'$) 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 after many steps:
    - 0 steps: `0`
    - 1 step: $\texttt{5+2} \longrightarrow \texttt{7}$
    - 2 steps: $\texttt{(5+2)+14} \longrightarrow \texttt{7+14} \longrightarrow \texttt{21}$
- but
    - `!4711`
    - `1+false`
    - $\texttt{(1+2)+false} \longrightarrow \texttt{3+false}$
- 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.

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

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

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

    $\lceil n \rceil$ stands for the syntactic representation of the number n.
  - If the operand of a negation is a truth value, the negation can be performed.

$$\text{(B-TRUE)} \quad \frac{}{!\texttt{true} \longrightarrow \texttt{false}} \qquad \text{(B-FALSE)} \quad \frac{}{!\texttt{false} \longrightarrow \texttt{true}}$$

Formalization: Nested Expressions

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

- Negation

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

- Addition, first operand

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

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

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

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:
  - $(!\mathtt{flag})[\mathtt{false}/\mathtt{flag}] \equiv !\mathtt{false}$
  - $(\mathtt{m}+\mathtt{n})[25/\mathtt{m}, 17/\mathtt{n}] \equiv 25+17$

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 type.

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 $\mathcal{J} = \vdash e : t$.

(INT)

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

(BOOL)

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

Progress: Addition

If $\dfrac{\vdash e_1 : \texttt{int} \quad \vdash e_2 : \texttt{int}}{\vdash e_1{+}e_2 : \texttt{int}}$ (ADD) is the final step of $\mathcal{J}$, then it holds that $e \equiv e_1{+}e_2$ and $t \equiv \texttt{int}$. Moreover, it is derivable that $\vdash e_1 : \texttt{int}$ and $\vdash e_2 : \texttt{int}$. 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 $\vdash e_2 : \texttt{int}$. 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 $(e_2 = v_1)$, 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'.

Progress: Negation

(NOT)

If $\dfrac{\vdash e_1 : \texttt{boolean}}{\vdash !e_1 : \texttt{boolean}}$ is the last step of $\mathcal{J}$, it holds that $e \equiv !e_1$ and $t \equiv \texttt{boolean}$ and $\vdash e_1 : \texttt{boolean}$ 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

Preservation

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

Proof

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

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

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

The case for rule $\mathrm{B\text{-}FALSE}$ is analoguous.

Preservation: Addition

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

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

holds with $e \equiv e_1{+}e_2$ and $t \equiv \texttt{int}$.
From $\vdash e_1 : \texttt{int}$ and $e_1 \longrightarrow e_1'$ it follows by induction that $\vdash e_1' : \texttt{int}$ holds. Another application of (ADD) on $\vdash e_1' : \texttt{int}$ and $\vdash e_2 : \texttt{int}$ yields $\vdash e_1'{+}e_2 : \texttt{int}$.
The case of rule (B-ADD-R) is analoguous.

Preservation: Negation

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

$$\stackrel{\text{(NOT)}}{\dfrac{\vdash e_1 : \texttt{boolean}}{\vdash !e_1 : \texttt{boolean}}}$$

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

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 $\vdash e[v_1/x_1, \dots, v_1/x_1] : t$.

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 $A \equiv A', x_0 : t_0$.

(VAR)

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

Substitution: Constants

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

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

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

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

Substitution: Addition

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

Substitution: Negation

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

QED

Theorem: Type Soundness of JAUS

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

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

  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).