Softwaretechnik

Lecture 03: Types and Type Soundness

Peter Thiemann

University of Freiburg, Germany

SS 2008

Table of Contents

Types and Type correctness

- big software systems : many involved people
  - project manager, designer, programmer, ...
- 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 their interfaces
  - the implementation fulfills the specification
  - each component is used correctly
- Main Problem : meet the interfaces and specifications:
  - simplest interface : *management of names*
    Which operations does the component offer?
  - simplest 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)

The language specifies a subset of Java expressions

x	::=	...	variables
n	::=	0 | 1 | ...	numbers
b	::=	`true` | `false`	truth values
e	::=	x | n | b | e+e | !e	expressions

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 there exists a typing rule, that defines:
  - if an expression is type correct
  - how to obtain the result type of the expression from the types of the subexpressions.
- five kinds of expressions (at first using words):
  - 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

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

Type judgment : expression e has type t

$$\vdash e : t$$

Formalization of "Typing rules"

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

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

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

Example: typing rules for JAUS

- numbers n has type `int`.

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

- truth values has type `boolean`.

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

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

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

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

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

Derivation trees and validity

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

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

Example: derivation trees

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

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

Variable

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

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

- An extended type 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 forward the environment.

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

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

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

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

Example: derivation with variable

The declaration `boolean flag;` matches the type assumption

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

Hence

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

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

(One possible) Approach

- 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
  - $!4711$
  - $1+\texttt{false}$
  - $(1+2)+\texttt{false} \longrightarrow 3+\texttt{false}$
- These expressions can not perform a reduction step. They correspond to runtime 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 number, we can add the two numbers to obtain a number as result.

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

$\lceil n \rceil$ represents the syntactic representation of the number n.

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

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

Formalization: nested expressions

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

- The negation

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

- addition, first operand

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

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

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

Variable

- An expression that contains variables cannot be evaluated with the reduction steps.
- But we can eliminate the variable using *substitution*. This replaces the variable with a value, and then reduction is possible.
- Apply a substitution $[v_1/x_1, \ldots v_n/x_n]$ to an expression e, written as

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

  changes in e each occurrence of x_i to the corresponding value v_i.
- example:
  - $(\texttt{!flag})[\texttt{false}/\texttt{flag}] \equiv \texttt{!false}$
  - $(\texttt{m+n})[25/\texttt{m}, 17/\texttt{n}] \equiv 25+17$

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 runtime 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 $e \longrightarrow e'$.

Prove

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

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

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

Progress: Addition

If $(\mathrm{ADD})\ \dfrac{\vdash e_1 : \texttt{int} \quad \vdash e_2 : \texttt{int}}{\vdash e_1 + e_2 : \texttt{int}}$ 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

If $(\mathrm{NOT})\ \dfrac{\vdash e_1 : \texttt{boolean}}{\vdash\ !e_1 : \texttt{boolean}}$ 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 $(\mathrm{B\text{-}NEG})$.
- 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 $(\mathrm{B\text{-}TRUE})$ or $(\mathrm{B\text{-}FALSE})$.

QED

Preservation

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

Prove

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

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

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

The case for rule B-FALSE is analoguous.

Preservation: Addition

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

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

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

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

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 throw substitution

Intention

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

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

If $(\mathrm{VAR})\ \dfrac{x : t \in A}{A \vdash x : t}$ is the latest 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) is 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

If $(\mathrm{INT})\ \dfrac{}{A \vdash n : \texttt{int}}$ is the latest step, it holds $(\mathrm{INT})\ \dfrac{}{A' \vdash n : \texttt{int}}$.

If $(\mathrm{BOOL})\ \dfrac{}{A \vdash b : \texttt{boolean}}$ is the latest step, it holds

$(\mathrm{BOOL})\ \dfrac{}{A' \vdash b : \texttt{boolean}}$.

Substitution: Addition

If $(\mathrm{ADD})\ \dfrac{A \vdash e_1 : \texttt{int} \quad A \vdash e_2 : \texttt{int}}{A \vdash e_1 + e_2 : \texttt{int}}$ is the latest 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 $(\mathrm{NOT})\ \dfrac{A \vdash e_1 : \texttt{boolean}}{A \vdash\ !e_1 : \texttt{boolean}}$ is the latest 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: 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).