Table of Contents

Types and Type Correctness
- JAUS: Java-Expressions (Ausdrücke)
- Evaluation of Expressions
- Type correctness
- Result
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?
Grammar for a subset of Java expressions

\[
x ::= \ldots \quad \text{variables}
\]

\[
n ::= 0 | 1 | \ldots \quad \text{numbers}
\]

\[
b ::= \text{true} | \text{false} \quad \text{truth values}
\]

\[
e ::= x | n | b | e + e | !e \quad \text{expressions}
\]
Correct and Incorrect Expressions

▶ type correct expressions

```java
boolean flag;
...
0
ture
17+4
!flag
```

▶ expressions with type errors

```java
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

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

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 $(J_1 \ldots J_n, J_0)$ 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 $(\varepsilon, J_0)$ is an **axiom**.
Example: Typing Rules for JAUS

- A number $n$ has type `int`.
  \[
  \text{(INT) \quad \vdash n : \text{int}}
  \]

- A truth value has type `boolean`.
  \[
  \text{(BOOL) \quad \vdash b : \text{boolean}}
  \]

- An expression $e_1 + e_2$ has type `int` if $e_1$ and $e_2$ have type `int`.
  \[
  \text{(ADD) \quad \vdash e_1 : \text{int} \quad \vdash e_2 : \text{int}}
  \]
  \[
  \vdash e_1 + e_2 : \text{int}
  \]

- An expression $!e$ has type `boolean`, if $e$ has type `boolean`.
  \[
  \text{(NOT) \quad \vdash e : \text{boolean}}
  \]
  \[
  \vdash !e : \text{boolean}
  \]
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. $\frac{J_1 \ldots J_n}{J}$, if $\frac{J_1 \ldots J_n}{J}$ is an instance of a rule and each $J_k$ is a derivation tree for $J_k$. 

Derivation Trees and Validity
Example: Derivation Trees

(INT)

▶ \( \vdash 0 : \text{int} \) is a derivation tree for judgment \( \vdash 0 : \text{int} \).

(BOOL)

▶ \( \vdash \text{true} : \text{boolean} \) is a derivation tree for \( \vdash \text{true} : \text{boolean} \).

▶ The judgment \( \vdash 17 + 4 : \text{int} \) holds, because of the derivation tree

\[
\begin{array}{c}
\text{(ADD)} \\
\text{(INT)} \quad \text{(INT)} \\
\vdash 17 : \text{int} \quad \vdash 4 : \text{int} \\
\hline
\vdash 17 + 4 : \text{int}
\end{array}
\]
Variable

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

\[ A ::= \emptyset | A, x : t \]  

- 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.

\[ (\text{VAR}) \]
\[ x : t \in A \]
\[ \frac{}{A \vdash x : t} \]
Extension of the Remaining Typing Rules

- The typing rules propagate the typing environment.

\[(\text{INT})\]
\[A \vdash n : \text{int}\]

\[(\text{BOOL})\]
\[A \vdash b : \text{int}\]

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

\[(\text{NOT})\]
\[A \vdash !e : \text{boolean}\]
\[\quad \frac{}{A \vdash e : \text{boolean}}\]
Example: Derivation with Variable

The declaration `boolean flag;` matches the type assumption

\[ A = \emptyset, \text{flag: boolean} \]

Hence the derivation

\[
\frac{
\text{flag: boolean} \in A
}{
A \vdash \text{flag: boolean}
}
\]

\[
\frac{
A \vdash \text{flag: boolean}
}{
A \vdash \neg \text{flag: boolean}
}
\]
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* $e \rightarrow e'$ over expressions.
- Expression $e$ *reduces in one step to* $e'$ (Notation: $e \rightarrow e'$) if one computational step leads from $e$ to $e'$.
- Example:
  - $5 + 2 \rightarrow 7$
  - $(5 + 2) + 14 \rightarrow 7 + 14$
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: $5+2 \rightarrow 7$
  - 2 steps: $(5+2)+14 \rightarrow 7+14 \rightarrow 21$
- but
  - $!4711$
  - $1+\text{false}$
  - $(1+2)+\text{false} \rightarrow 3+\text{false}$
- 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.
    \[(\text{B-ADD})\]
    \[
    \begin{array}{c}
    \lceil n_1 \rceil + \lceil n_2 \rceil \\
    \end{array}
    \rightarrow \lceil n_1 + n_2 \rceil
    \]
    \[
    \begin{array}{c}
    [n] \text{ stands for the syntactic representation of the number } n.
    \end{array}
    \]
  - If the operand of a negation is a truth value, the negation can be performed.
    \[(\text{B-TRUE})\]
    \[
    \begin{array}{c}
    \neg \text{true } \\
    \end{array}
    \rightarrow \text{false}
    \]
    \[(\text{B-FALSE})\]
    \[
    \begin{array}{c}
    \neg \text{false } \\
    \end{array}
    \rightarrow \text{true}
    \]
Formalization: Nested Expressions

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

- **Negation**
  \[
  (B-NEG) \\
  e \rightarrow e' \\
  !e \rightarrow !e'
  \]

- **Addition, first operand**
  \[
  (B-ADD-L) \\
  e_1 \rightarrow e_1' \\
  e_1 + e_2 \rightarrow e_1' + e_2
  \]

- **Addition, second operand (only evaluate the second, if the first is a value)**
  \[
  (B-ADD-R) \\
  e \rightarrow e' \\
  v + e \rightarrow v + e'
  \]
### 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 \([v_1/x_1, \ldots v_n/x_n]\) 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:
  - \((!\text{flag})[false/\text{flag}] \equiv !\text{false}\)
  - \((m+n)[25/m, 17/n] \equiv 25+17\)
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 $e \rightarrow e'$, 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 $e \rightarrow e'$.

Proof
Induction over the derivation tree of $J = \vdash e : t$.

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

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

(ADD)

\[ \frac{\vdash e_1 : \text{int} \quad \vdash e_2 : \text{int}}{\vdash e_1 + e_2 : \text{int}} \]

If \( e \equiv e_1 + e_2 \) 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 \( e_1 \rightarrow e'_1 \).

- If \( e_1 \rightarrow e'_1 \) holds, we obtain that \( e \equiv e_1 + e_2 \rightarrow e' \equiv e'_1 + e_2 \) cause of rule \( \text{(B-ADD-L)} \). 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 \( e_2 \rightarrow e'_2 \).
  
  - In the second case we can use rule \( \text{(B-ADD-R)} \) and get:
    \[ e \equiv v_1 + e_2 \rightarrow e' \equiv v_1 + e'_2. \]
  
  - 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 \( \text{(B-ADD)} \) and obtain the desired \( e' \).
Progress: Negation

\begin{align*}
(\text{NOT}) \quad & 
\begin{prooftree}
\Gamma \vdash e_1 : \text{boolean} \\
\Gamma \vdash \lnot e_1 : \text{boolean}
\end{prooftree}
\end{align*}

If \( \frac{\Gamma \vdash e_1 : \text{boolean}}{\Gamma \vdash \lnot e_1 : \text{boolean}} \) is the last step of \( \mathcal{J} \), it holds that \( e \equiv \lnot e_1 \) and \( t \equiv \text{boolean} \) and \( \Gamma \vdash e_1 : \text{boolean} \) is derivable.

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

- In the case that \( e_1 \rightarrow e'_1 \), we conclude that there exists \( e' \) with \( e \rightarrow e' \) using rule \( (B-\text{NEG}) \).

- 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 \( (B-\text{TRUE}) \) or \( (B-\text{FALSE}) \).

\[ \text{QED} \]
Preservation

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

Proof

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

\[(B\text{-ADD})\]

If \hspace{1cm} is the reduction step, then it holds that \hspace{1cm} $t \equiv \text{int}$ because of (ADD). We can apply (INT) to $e' = \lfloor n_1 + n_2 \rfloor$ and obtain the desired result $\vdash \lfloor n_1 + n_2 \rfloor : \text{int}$.

\[(B\text{-TRUE})\]

If \hspace{1cm} is the reduction step it holds that \hspace{1cm} $t \equiv \text{boolean}$ 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
(B-ADD-L)

If \( e_1 \rightarrow e'_1 \leadsto e_1 + e_2 \rightarrow e'_1 + e_2 \)

is the occasion for the last step, we obtain through
\[ \vdash e : t \] that

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

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

From \( \vdash e_1 : \text{int} \) and \( e_1 \rightarrow e'_1 \) 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

**(B-NEG)**

If \( e_1 \rightarrow e'_1 \) is the occasion for the last step, we get through \( \vdash e : t \), that

\[
\neg
\]

holds with \( e \equiv \neg e_1 \) and \( t \equiv \text{boolean} \).

From \( \vdash e_1 : \text{boolean} \) and \( e_1 \rightarrow e'_1 \) we conclude (using induction) that

\( \vdash e'_1 : \text{boolean} \) holds. Another application of rule (NOT) to

\( \vdash e'_1 : \text{boolean} \) yields \( \vdash \neg e'_1 : \text{boolean} \).

**QED**
Elimination of Variables by 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 $A \equiv A', x_0 : t_0$.

(VAR)

If $x : t \in A$ 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 $A' \vdash x : t$. 

Peter Thiemann (Univ. Freiburg) Software Engineering 08.07.2013 31 / 35
Substitution: Constants

(\text{INT}) \quad (\text{INT})

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

(\text{BOOL}) \quad (\text{BOOL})

If $A \vdash b : \text{boolean}$ is the last step, it holds $A' \vdash b : \text{boolean}$. 
**Substitution: Addition**

\[ \text{(ADD)} \]

If \( A \vdash e_1 : \text{int} \) and \( A \vdash e_2 : \text{int} \) 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 \( \text{(ADD)} \) yields \( A' \vdash (e_1+e_2)[e_0/x_0] : \text{int} \).
Substitution: Negation

(\text{NOT})

\[
\begin{align*}
A \vdash e_1 : \text{boolean} \\
\overline{A \vdash \neg e_1 : \text{boolean}}
\end{align*}
\]

If is the last step, the induction hypothesis yields

\[
A' \vdash e_1[e_0/x_0] : \text{boolean}
\]

Apply rule (\text{NOT}) yields

\[
A' \vdash (\neg e_1)[e_0/x_0] : \text{boolean}
\]

QED
Theorem: Type Soundness of JAUS

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

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

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).