Software Engineering, Exercise Sheet 1

Martin Mehlmann
(mehlmann@informatik.uni-freiburg.de)

April 30, 2009

Exercise 1

- Code given:

```
s = "some random string";
s.x = 42;
s.x;
```

- Output of Rhino:

```
~/tmp/rhino1_7R1$ java -jar js.jar
Rhino 1.6 release 7 2007 08 19
js> s = "some random string";
some random string
js> s.x = 42;
42
js> s.x;
js>
```

Exercise 1

- Problem: Javascript inserts conversion code **automatically**

```
s = "some random string";
new String(s).x = 42;
new String(s).x;
```

- "Correct" code:

```
s = new String("some random string");
s.x = 42;
s.x;
```

- Output is now

```
js> s = new String("some random string");
some random string
js> s.x = 42;
42
js> s.x;
42
```

- Static typing catches these kind of errors

Exercise 2

(a) $1 + \texttt{true}$ is not type correct: $\texttt{true}$ has type $\texttt{boolean}$, but $+$ adds two expressions of type $\texttt{int}$.

(b) $23 + (47 - 11)$ has type $\texttt{int}$:

$$\text{(ADD)}\ \dfrac{\text{(INT)}\ \dfrac{}{\vdash 23 : \texttt{int}} \qquad \text{(SUB)}\ \dfrac{\text{(INT)}\ \dfrac{}{\vdash 47 : \texttt{int}} \qquad \dfrac{}{\vdash 11 : \texttt{int}}\ \text{(IN}}{\vdash 47 - 11 : \texttt{int}}}{\vdash 23 + (47 - 11) : \texttt{int}}$$

(c) $!(!\texttt{false})$ has type $\texttt{boolean}$

$$\text{(NOT)}\ \dfrac{\text{(NOT)}\ \dfrac{\text{(BOOL)}\ \dfrac{}{\vdash \texttt{false} : \texttt{boolean}}}{\vdash !\texttt{false} : \texttt{boolean}}}{\vdash !(!\texttt{false}) : \texttt{boolean}}$$

Exercise 2

(d) $y + x$ is not type correct: y has type `boolean`, but $+$ adds two expressions of type `int`.

(e) $!y$ has type `boolean`

$$\text{(NOT)}\ \dfrac{\text{(VAR)}\ \dfrac{y : \texttt{boolean} \in A}{A \vdash y : \texttt{boolean}}}{A \vdash !y : \texttt{boolean}}$$

where $A = (\emptyset, x : \texttt{int}, y : \texttt{boolean})$

Exercise 3

(a) $23 + (47 - 11) \longrightarrow 23 + 36 \longrightarrow 59$

$$\text{(B-ADD-R)}\ \dfrac{\text{(B-SUB)}\ \dfrac{}{47 - 11 \longrightarrow 36}}{23 + (47 - 11) \longrightarrow 23 + 36}$$

$$\text{(B-ADD)}\ \dfrac{}{23 + 36 \longrightarrow 59}$$

59 is a value

(b) $(1 + 1) + \texttt{true} \longrightarrow 2 + \texttt{true}$

$$\text{(B-ADD-L)}\ \dfrac{\text{(B-ADD)}\ \dfrac{}{1 + 1 \longrightarrow 2}}{(1 + 1) + \texttt{true} \longrightarrow 2 + \texttt{true}}$$

$2 + \texttt{true}$ is **not** a value. Note that the original expression is ill-typed.

Exercise 4

Lemma (Normalization)

For every expression e_0, there exists an expression e_n such that

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

and no expression e_{n+1} exists with $e_n \longrightarrow e_{n+1}$.

Proof. Define the size of an expression as follows:

$$\text{size}(e) = \begin{cases} 1 & \text{if } e = x \text{ or } e = b \text{ or } e = \lceil m \rceil \\ 1 + \text{size}(e') + \text{size}(e'') & \text{if } e = e' + e'' \\ 1 + \text{size}(e') & \text{if } e = !e' \end{cases}$$

We can easily prove that $e \longrightarrow e'$ implies $\text{size}(e) > \text{size}(e')$.
(The proof is by induction on the derivation of $e \longrightarrow e'$.)

Exercise 4

We now assume the contraposition of the lemma to prove. That is, we assume that for some expression e_0 there exists an infinite reduction sequence

$$e_0 \longrightarrow e_1 \longrightarrow e_2 \longrightarrow \dots \longrightarrow e_i \longrightarrow e_{i+1} \longrightarrow \dots$$

Then we argue: Because an expression's size decreases with every reduction step and because the size of an expression is never negative, there exists some e_i with $\text{size}(e_i) = 1$.
But $e_i \longrightarrow e_{i+1}$, so $\text{size}(e_{i+1}) < \text{size}(e_i) = 1$ which is a contradiction.

Exercise 4

Lemma (Multi-step preservation)

If $\vdash e_0 : t$ *and* $e_0 \longrightarrow e_1 \longrightarrow e_2 \longrightarrow \ldots \longrightarrow e_{n-1} \longrightarrow e_n$ *then* $\vdash e_n : t$*.*

Proof. By induction on n:

- $n = 0$. Then $\vdash e_n : t$ by assumption.
- $n > 0$ and the claim holds for $n - 1$. Hence, $\vdash e_{n-1} : t$ and $e_{n-1} \longrightarrow e_n$. The preservation lemma now gives as $\vdash e_n : t$ as required.

Exercise 4

Theorem (Type soundness)

If $\vdash e_0 : t$ *then there exists a value* e_n *such that* $\vdash e_n : t$ *and*

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

Proof. By the *normalization lemma*, we know that there exists some expression e_n with

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

and no e_{n+1} exists with $e_n \longrightarrow e_{n+1}$.
The *progress lemma* now tells us that e_n is a value (otherwise, e_n would reduce to some e_{n+1}).
The *multi-step preservation lemma* gives us $\vdash e_n : t$.