CS 444
Lecture 15, March 2nd, 2020
Matthew Walinga

March 2, 2020

1 Casts and stuff

$$\frac{C, L, \sigma \vdash E : \tau_1 \ \tau_1 := \tau_2 \vee \tau_2 := \tau_1}{C, L, \sigma \vdash (\tau_2)E : \tau_2}$$

$$\frac{C, L, \sigma \vdash E : \tau \ \tau := D \vee D := \tau}{C, L, \sigma \vdash E \text{ instanceof } D : \text{boolean}}$$

2 Methods

- Java closest match is JLS 15.12.2 (we won't do this, we check exact types only)

$$\frac{\forall i : C, L, \sigma \vdash E_i : \tau_i \quad \text{static } \tau \ m(\tau_1, ..., \tau_k) \in \text{continue(D)}}{C, L, \sigma \vdash D.m(E_1, ...E_k) : \text{E}}$$

$$\frac{\forall i : C, L, \sigma \vdash E_i : \tau_i \quad C, L, \sigma \vdash E : D \quad \tau \ m(\tau_1, ..., \tau_k) \in \text{continue(D)}}{C, L, \sigma \vdash E.m(E_1, ...E_k) : \tau}$$

$$\frac{\forall i : C, L, \sigma \vdash E_i : \tau_i \quad D(\tau_1, ..., \tau_k) \in D}{C, L, \sigma \vdash newD(E_1, ...E_k) : D}$$

2.1 Return types

$$\frac{C, L, \sigma \vdash E_i : \tau_i \quad \sigma := \tau}{}$$

$C, L, \sigma \vdash \text{return } E$
$C, L, \sigma \vdash \text{return}$

3 Arrays

$$\frac{C, L, \sigma \vdash E_1 : \tau_1[] \quad C, L, \sigma \vdash E_2 : \tau_2 \quad num(\tau_2)}{C, L, \sigma \vdash E_1[E_2] : \tau_1}$$

$$\frac{C, L, \sigma \vdash E_1 : \tau_1[] \quad C, L, \sigma \vdash E_2 : \tau_2 \quad num(\tau_2) \quad C, L, \sigma \vdash E_3 : \tau_3 \quad \tau_1 := \tau_3}{C, L, \sigma \vdash E_1[E_2] = E_3 : \tau_1}$$

$$\frac{C, L, \sigma \vdash E : \tau[]}{C, L, \sigma \vdash E.length : int}$$

$$\frac{C, L, \sigma \vdash E : \tau_2 \quad num(\tau_2)}{C, L, \sigma \vdash \text{new } \tau_1[E] : \tau_1[]}$$

3.1 Array Assignability

- Arrays can be assigned to the following types:
  Object := $\sigma[]$
  Cloneable := $\sigma[]$
  java.io.Serializable := $\sigma[]$

3.1.1 An evil example

$$\frac{D \leq C}{C[] := D[] \text{ if } C := D \text{ (covariant)}}$$

Here's some code as an example:

```
B[] bs = new B[1];
Object[] os = bs;
os[0] = new C();
B b = bs[0];
```

This program is not type correct! It assigns a C-type object to a B-type object :o
The rule we defined makes the static type system unsound

Solution: To preserve type-correctness, we must "tag" each array element. Java checks the dynamic type at every array store. This may result in a runtime exception: ArrayStore-Exception (JLS 10.10)

The covariant rule we defined above was the problem. Covariants should only be used for data structures that are read-only

3.1.2 Other soundess holes

- Casts are intentional soundness holes (programmer tells the compiler "trust me")

3.1.3 A few more rules

Two-dimensional arrays (not in Joos so who really cares, but here for completeness)
$\sigma[] := \tau[]$
$\sigma[][] := \tau[][]$

4 Type-checking pseudocode (Joos)

"Demand-driven approach"

(A) premises
$\overline{C, L, \sigma \vdash E : \tau}$

```
typecheck(E) {
1. find a rule of the form (A) above
2. check premises (with recursive calls to typecheck subexpressions)
3. return tau
}
```

5 Static Analysis

- analyze a program to prove properties of its runtime behaviour
  - Applications: generate efficient optimized code