SFJ: An implementation of Semantic Featherweight Java

Artem Usov and Ornela Dardha[0000−0001−9927−7875]

School of Computing Science, University of Glasgow, United Kingdom
{artem.usov, ornela.dardha}@@glasgow.ac.uk

Abstract. The abstract should briefly summarize the contents of the paper in 15–250 words.

Keywords: nominal and structural subtyping · Featherweight Java · object-oriented languages · semantic subtyping · type theory.

1 Introduction

A typing system for a programming language is a set of deduction rules that allow

$$\tau ::= \alpha \mid \mu \tag{1}$$

Since we want our types to represent sets of values, we restrict our types to finite trees whose leaves are constants with no cycles. For example a recursive type $\alpha = [a : \alpha]$ would be an infinite tree **new** $C($**new** $C(\cdots))$. Similarly the types $\alpha = [b : \beta]$, $\beta = [a = \alpha]$ would also be impossible to instantiate.

Therefore, to not allow this, when processing the AST of a program, we do not allow the type of the class we are defining to be a field and we mark any classes with only basic types in it's fields as *resolved* and otherwise if it contains a class type, we mark it as *unresolved*. After processing the whole AST, we perform the following algorithm to decide whether the type definitions in the program are valid.

```
boolean resolutionOccured = false
do
    for class that is unresolved:
        boolean resolved = true

        for field that contains a class type:
            if class type is not resolved
                resolved = false

        if resolved is true
            set class to be resolved
            resolutionOccured = true
```

```
while resolutionOccured is true

if all classes are not resolved
    program contains invalid type definition
```

Given that we now know that all the types in the program are valid, we can create the subtyping relation which will be a map of types to a set of its subtypes. For every program, we assume the initial subtyping relation for all of our basic types. We would like to highlight that Int is not a subtype of Float as it cannot represent the whole domain of Int accurately therefore the domain is not fully contained in Float. Similarly so for Double and Long.

$$
\begin{aligned}
Boolean &= \{Boolean\} \\
Double &= \{Double, Float, Int, Short, Byte\} \\
Float &= \{Float, Short, Byte\} \\
Long &= \{Long, Int, Short, Byte\} \\
Int &= \{Int, Short, Byte\} \\
Short &= \{Short, Byte\} \\
Byte &= \{Byte\}
\end{aligned}
$$

Knowing that all our types are finite trees with leaves as constants and given this initial relation, we can now confidently create a subtyping relation for all class types using the following algorithm.

```
function generateRelation(classes):
    List<class> untyped = []

    for class in classes:
        if addClass(class) is false:
            untyped.add(class)

    if untyped is not []:
        generateRelation(untyped)

function addClass(class):
    for existing class type in relation:
        if tryAddSubtype(class, existingClass) is false:
            return false

        tryAddSubtype(existingClass, class)

        add class to its own subtype relation

function tryAddSubtype(class, other):
```

```
boolean flag = true

for field in class:
    if field contains type not in relation:
        return false

    if other does not have field:
        flag = false
    else:
        if other.field.types not fully contains field.types:
            flag = false


for method in class:
    if method contains type not in relation:
        return false

    if other does not have method:
        flag = false
    else:
        if other.method.types not fully contains method.types:
            flag = false

if flag == true:
    add class to other subtype relation
```

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 1 gives a summary of all heading levels.

Table 1. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Displayed equations are centered and set on a separateline.

$$x + y = z \tag{2}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead.

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal articles [1].

References

1. Dardha, O., Gorla, D., Varacca, D.: Semantic Subtyping for Objects and Classes. Computer Journal **60**(5), 636–656 (apr 2017). https://doi.org/10.1093/comjnl/bxw080