Assignment 5: SCOOP type system

ETH Zurich

1 Subtyping

1.1 Background

Have a look at the attributes shown in listing 1.

Listing 1: Attributes

1 \texttt{px: PROCESSOR}
   \texttt{py: PROCESSOR}
3 \texttt{a: separate X}
5 \texttt{b: separate <px> X}
\texttt{c: separate <py> X}
7 \texttt{d: X}
\texttt{e: detachable separate X}
9 \texttt{f: detachable separate <px> X}
\texttt{g: detachable X}

1.2 Task

Decide whether the following attachments are valid or not. Justify your answer.

1. \( a := b \)
2. \( a := d \)
3. \( b := a \)
4. \( b := c \)
5. \( b := d \)
6. \( d := a \)
7. \( d := b \)
8. \( a := e \)
9. \( e := a \)
1.3 Solution

1. The assignment $a := b$ is valid. All type components of $b$ are conformant to the type components of $a$.
2. The assignment $a := d$ is valid. All type components of $d$ are conformant to the type components of $a$.
3. The assignment $b := a$ is invalid. The $\top$ processor tag does not conform to the explicit processor tag.
4. The assignment $b := c$ is invalid. The two explicit processor tags are not conformant to each other. The two explicit processor tags denote different processors.
5. The assignment $b := d$ is invalid. The non-separate processor tag does not conform to the explicit processor tag. The explicit processor tag denotes a processor different than the current processor.
6. The assignment $d := a$ is invalid. The $\top$ processor tag does not conform to the non-separate processor tag. Statically the $\top$ processor tag can denote any processor.
7. The assignment $d := b$ is invalid. The explicit processor tag does not conform to the non-separate processor tag. The explicit processor tag denotes a processor different than the current processor.
8. The assignment $a := e$ is invalid. A detachable type does not conform to an attached type.
9. The assignment $e := a$ is valid. All type components of $a$ are conformant to the type components of $e$.

2 Valid targets

2.1 Background

Have a look at listing 2.

Listing 2: Enclosing Feature

```
p: PROCESSOR
2
r (a: detachable separate X; b: separate <p> X; c: separate X)
4 local
   d: separate <p> X
6 e: separate <c.handler> X
   f: separate X
8 do
10 end
```

Imagine that the class $X$ has a function $g: X$ and a procedure `do_something`. 
2.2 Task

Decide for each of the following feature calls, whether the calls are valid or not when they appear in feature r of listing 2.

1. c.do_something
2. c.g.do_something
3. e := c; e.do_something
4. f := c; f.do_something
5. a.do_something
6. d := b; d.do_something

2.3 Solution

1. The call c.do_something is valid. The target c is attached and it appears as a formal argument in the enclosing routine.

2. The call c.g.do_something is valid. The expression c has an implicit type (!, c.handler, X). The result type combiner yields (!, c.handler, X) as the type of c.g. Thus the target c.g is attached and has a qualified explicit processor tag denoting an attached formal argument of the enclosing routine.

3. The call e.do_something is valid. The target e is attached and has a qualified explicit processor tag denoting an attached formal argument of the enclosing routine.

4. The call in f := c; f.do_something is invalid. The entity f is separate and does not correspond to any of the attached formal arguments in the enclosing routine. At runtime the entity f will be attached to a controlled object. Therefore an object test would help to make the call valid.

5. The call a.do_something is invalid. The target a is not attached.

6. The call d.do_something is valid. The target d is attached and it has the same same unqualified explicit processor tag as one of the attached formal arguments in the enclosing routine.

3 Separate generics or generic separate?

3.1 Background

The interplay between generics and separate types are important to understand, and enforce a good understanding of the type system.

3.2 Task

Consider the differences between:

- separate LIST [BOOK]
- LIST [separate BOOK]

Explain the distinction using the object/processor diagram.
3.3 Solution

A separate list of books:

A list of separate books:

4 Basic library: type combiner

4.1 Background

Consider the classes in listing 3. These classes belong to a basic library implementation.

Listing 3: Basic Library

```plaintext
class LIST[G]
  feature
    last : G
      -- Last element.
  put(a_element: G)
```
--- Add the element to the list.

```java
8  do
9      ...
10  end
```

class `LIBRARY`

```java
12  class `LIBRARY`
14     feature
```

### 4.2 Task
What is the result type of `books.last` from the perspective of the library? What is the type of an actual argument in the call `books.put (...)` from the perspective of the library? Justify your answer.

### 4.3 Solution
The type of the target `books` is `(!, ●, `LIST`[(!, T, BOOK)])`. The result type of `last` is `(!, T, BOOK)`. As a result one gets `(!, ●, `LIST`[(!, T, BOOK)]) ⊗ (!, T, BOOK) = (!, T, BOOK)`. The type of the formal argument of `put` is `(!, T, BOOK)`. Thus the combination yields `(!, ●, `LIST`[(!, T, BOOK)]) ⊗ (!, T, BOOK) = (!, T, BOOK)`.  

### 5 Stack library: type combiner

#### 5.1 Background
Consider the alternative stack based library implementation shown in listing 4.

Listing 4: Stack Library

```java
class `LIST`[G]
2     feature
3         last: `G` -- Last element.
4     end

6 class `STACK`[G]
7     feature
8         top: `G` -- Top element.
9     end
10  class `LIBRARY`
12     feature
16     end
```

#### 5.2 Task
What is the result type of `books.last.top` from the perspective of the library? Justify your answer.
5.3 Solution

The result type can be determined by applying the result type combiner several times as shown in the following.

\[
(!, \bullet, \text{LIST}[B]) \ast \underbrace{(!, \bullet, \text{STACK}[A])} \ast (!, \top, \text{BOOK}) =
\]

\[
(!, \bullet, \text{STACK}[A]) \ast (!, \top, \text{BOOK}) = (!, \top, \text{BOOK})
\]