Homework Set 1

CMCS 22620/32620, Spring 2004

Assigned: April 5, 2004
Due: April 12, 2004

1. (Subtyping)

(a) I am sure everybody understands that the following line of reasoning is wrong:

\[(a \geq b) \land (a \geq c) \land (b \geq d) \Rightarrow (c \geq d)\]

Explain in English how this is related to the Java array co-variance bug that we discussed in class. If you like, you can illustrate the problem using the following code snippet:

```java
class A {
    . . .
}
class B extends A {
    . . .
}
...
B [] bx = new B [10];
A [] ax = bx;
A c = new A ();
A [0] = c;
```

(You should not need more than a short paragraph for your explanation.)

(b) I added conditional expressions of the form \(e_1?e_2:e_3\) to Minijava. Here, \(e_1\) has to return a value of type `boolean`; if it is `true`, then \(e_2\) gets evaluated and returned, otherwise \(e_3\).

Explain in English what the typing rule for this construct should be. In particular, how is the result type of the whole construct related to the types of \(e_2\) and \(e_3\)? (Hint: In the class hierarchy shown below, think about what the types of the following expressions are:

\(c?\text{new } C():\text{new } D()\)  \(c?\text{new } C():\text{new } E()\)  \(c?\text{new } B():\text{new } A()\)

Here is the class hierarchy:

```java
class A {
    . . .
}
class B extends A {
    . . .
}
class C extends B {
    . . .
}
class D extends B {
    . . .
}
class E extends A {
    . . .
}
```
(c) Let $t_2$ be the type of $e_2$ and $t_3$ that of $e_3$. Using the symbol $\leq$ to denote subtyping, write down a sufficient set of conditions that the answer $t$ to the previous question must satisfy.

To get the idea for what such conditions should look like, consider the following example. Let $T$ be the set of all subtypes of $t$. This means that $T$ must satisfy:

$$ t' \leq t \Rightarrow t' \in T $$
$$ t' \in T \Rightarrow t' \leq T $$

(d) With the answers to the previous questions in mind, look at the code for the Minijava frontend that has been provided. In file `semant.sml`, locate the function that is used to calculate the type of a conditional expression from the types of its subexpressions $e_2$. What is its name? Can you guess why the name was chosen like this? Hint: In mathematical terms, the Minijava subtyping relation is a partial ordering. Given the types of $e_2$ and $e_3$, what is the name for the type of $c?e_2:e_3$ using terminology associated with partial orderings?

(e) The expression `null` is the null-pointer of Java. It can be used where arrays or objects are expected. In `types.sml` you find the definition of a type value called `NULLtyp`. This is the internal type assigned to `null`. This type has no external equivalent. Explain in 2 or 3 sentences what the purpose of such a type is. How is it related to other types in the subtyping hierarchy?

2. (Expressing Invariants as Types)

As has been explained in class, our Tree language differs slightly from that used in our textbook. Instead of making `TEMP` and `MEM` plain expressions (in `exp`) we have a separate type `lexp` for them. There is a new constructor for `exp` that carries an `lexp`, and the `stm`-constructor `MOVE` has been restricted to `lexp`. The purpose of this change is to express the invariant of `MOVE` being restricted to carrying only `TEMP` or `MEM` in its first argument.

One of the first steps of dealing with Tree values is to linearize them. A linearized tree (or better: forest) is a list of statements where each statement in the list satisfies the following additional constraints:

- The use of `SEQ` and `ESEQ` is forbidden.
- The parent of every `CALL` is either `EXP` or `MOVE`.

Using the definition of structure `Tree` (in `tree.sml`) as a template, define a similar structure `LinTree` with types `exp`, `lexp`, and `stm` that encodes these additional invariants. Obviously, your `exp` will not need an `ESEQ` and your `stm` will not need a `SEQ`. To deal with the invariant concerning `CALL`, consider adding a new type in the spirit of how `lexp` was added.