

Sample solution for Pierce 15.5.2.

(1) If we drop the first premise of S-REF, we have

$$\frac{T_1 <: S_1}{\text{Ref } S_1 <: \text{Ref } T_1} \quad (\text{S-REF1})$$

Since $\{x:\text{Bool}, y:\text{Bool}\} <: \{x:\text{Bool}\}$, we have by S-REF1 that $\text{Ref } \{x:\text{Bool}\} <: \text{Ref } \{x:\text{Bool}, y:\text{Bool}\}$. So the program

```
(λr:Ref {x:Bool, y:Bool} .!r.y)(ref {x=true})
```

or, equivalently,

```
let r = ref {x=true} as Ref {x:Bool, y:Bool} in
  !r.y
```

will typecheck, since we can apply T-SUB to the argument to make it match the declared type (or ascription) for r . But the program will clearly get stuck attempting to access the y field from a record that has none.

(2) If we drop the second premise of S-REF, we have

$$\frac{S_1 <: T_1}{\text{Ref } S_1 <: \text{Ref } T_1} \quad (\text{S-REF2})$$

This gives $\text{Ref } \{x:\text{Bool}, y:\text{Bool}\} <: \text{Ref } \{x:\text{Bool}\}$. So the program

```
let r = ref {x=true, y=true}
in (λu:Ref {x:Bool}. u := {x=true}) r;
  r.y
```

or, equivalently,

```
let r = ref {x=true, y=true} in
let u = r as Ref {x:Bool} in
  u := {x=true};
  r.y
```

will typecheck, since we can apply T-SUB to r to make it match the declared type (or ascription) for u . But the program will again get stuck attempting to access the y field from a record that has none.

Sample solution for Pierce 15.5.3

Compiling and executing the program

```
class T {  
    public static void main(String argv[]) {  
        Object[] a = new String[1];  
        a[0] = new Object();  
    }  
}
```

yields

```
Exception in thread "main" java.lang.ArrayStoreException  
    at T.main(T.java:4)
```