Sample Solution for Homework 12

Problem 1 Objects and Subtyping (14 Points)

This is a warm-up exercise to get yourself familiar with subtyping and the computation of joins and meets in the subtype lattice.

(a) Indicate whether the following JakartaScript subtype relationships are true or false.

(i) `Bool <: Num`
Answer: false

(ii) `{let f: Num} <: {let f: Any}`
Answer: false

(iii) `{let f: Num} <: {const f: Any}`
Answer: true

(iv) `{const f: Num} <: {let f: Num}`
Answer: false

(v) `{let f: Num} <: {let g: Num}`
Answer: false

(vi) `{const f: {}} <: {const f: Any, let g: Bool}`
Answer: false

(vii) `Any => Bool <: Bool => Any`
Answer: true

(viii) `Bool => Bool <: Any => Any`
Answer: false

(b) For each of the following programs, determine whether the program is well-typed with subtyping. If it is well-typed, give the type that is inferred for the whole program. If it is not well-typed, provide a brief explanation of the type error.

(i)

```
1 const x = {let f: 3};
2 const fun = function(y: {let f: Num, const g: Bool}) {
3     y.f = 4; return y;
4   };
5 fun(x).f
```

The program is not well-typed. The type error is at the function call `fun(x)` because the inferred type for `x` is `{let f: Num}`, which is not a subtype of the type of the parameter `y` of `fun`.

(ii)

```
1 const x = {let f: 3, const g: true};
2 const fun = function(y: {let f: Num}) {
```

```
3       return y;
4     };
5   fun(x).f
```

The program is well-typed. The inferred type is **Num**.

(iii)

```
1  const x = {let f: 3, const g: true};
2  const fun = function(y: {let f: Num}) {
3       return y;
4     };
5  fun(x).g
```

The program is not well-typed. The problem is that the program dereferences field `g` of the object returned by `fun(x)`. However, the inferred return type of function `fun` is {**let** `f:` **Num**}, which has no field `g`.

(iv)

```
1    const x = {let f: 1, const g: true};
2    const y = {const f: false, let g: 2};
3    const z = true ? x : y;
4    z.f
```

The program is well-typed. The inferred type is **Any**.

(c) For each of the following pairs of types τ_1 and τ_2, compute their join $\tau_1 \sqcup \tau_2$ and meet $\tau_1 \sqcap \tau_2$.

(i) $\tau_1 =$ **Num**, $\tau_2 =$ {**const** `f:` **Num**}
$\tau_1 \sqcup \tau_2 =$ **Any**
$\tau_1 \sqcap \tau_2 =$ **Nothing**

(ii) $\tau_1 = \{\}$, $\tau_2 =$ {**let** `f:` **Num**, **const** `g:` **Bool**}
$\tau_1 \sqcup \tau_2 = \{\}$
$\tau_1 \sqcap \tau_2 =$ {**let** `f:` **Num**, **const** `g:` **Bool**}

(iii) $\tau_1 =$ {**let** `f:` **Num**}, $\tau_2 =$ {**const** `g:` **Bool**}
$\tau_1 \sqcup \tau_2 = \{\}$
$\tau_1 \sqcap \tau_2 =$ {**let** `f:` **Num**, **const** `g:` **Bool**}

(iv) $\tau_1 =$ {**let** `f:` **Num**, **const** `g:` {**let** `h:` **Any**}},
$\tau_2 =$ {**let** `f:` **Num**, **const** `g:` {**const** `h:` **Bool**}}
$\tau_1 \sqcup \tau_2 =$ {**let** `f:` **Num**, **const** `g:` {**const** `h:` **Any**}}
$\tau_1 \sqcap \tau_2 =$ **Nothing**

(v) $\tau_1 =$ (**Any** `=>` **Bool**), $\tau_2 =$ (**Bool** `=>` **Any**)
$\tau_1 \sqcup \tau_2 =$ **Bool** `=>` **Any**
$\tau_1 \sqcap \tau_2 =$ **Any** `=>` **Bool**

(vi) $\tau_1 =$ (**Bool** => **Num**) , $\tau_2 =$ (**Num** => **Bool**)

$\tau_1 \sqcup \tau_2 =$ **Nothing** => **Any**

$\tau_1 \sqcap \tau_2 =$ **Any** => **Nothing**