Sample Solution for Homework 11

Problem 1 Objects and Subtyping (16 Points)

This is a warm-up exercise to get yourself familiar with objects (which you will implement in the second part of this homework) and subtyping (which you will implement in the upcoming final homework assignment).

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

(i) `Num <: Num`
Answer: true

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

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

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

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

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

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

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

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

(b) For each of the following programs: (1) say whether the program can be safely evaluated or not, i.e., whether the evaluation will get stuck or produce a value. If it can be safely evaluated, provide the computed value; (2) determine whether the program is well-typed with subtyping (i.e., with the rule TypeSub). If it is not, provide a brief explanation.

(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 can be safely evaluated. The result value is `4`. However, it is not well-typed. The call to `fun` on line 5 passes an object of type {**let** `f:` **Num**}, which is not a subtype of {**let** `f:` **Num**, **const** `g:` **Bool**}, the expected type of `fun`'s parameter.

(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 can be safely evaluated. The result value is `3`. The program is also well-typed.

(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 can be safely evaluated. The result value is **true**. However, the program is not well-typed. The inferred type of `fun`'s return type is {**let** `f:` **Num**}. On line 5, `fun` is called in a context that expects an object with a field `g`. However, the type {**let** `f:` **Num**} is not a subtype of any object type which has a 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 can be safely evaluated. The result value is `1`. The program is also well-typed. The expressions `x` and `y` on line 3 have a common supertype, e.g., the type {**const** `f:` **Any**, **const** `g:` **Any**}, which can be used to type the field dereference expression on line 4.