Theoretical Questions Homework 3:

Question 1:

a. {f : [T2 → T3], g : [T1 → T2], a : Number} ⊢ (f (g a)) : T3

This statement is true:

1. We know that 'a' is of type Number, which is a subtype of T1.
2. 'g' takes T1 as input and returns T2.
3. 'f' takes T2 as input and returns T3.
4. So, (g a) will return a value of type T2.
5. Then, f takes this T2 value and returns T3. Therefore, (f (g a)) will indeed be of type T3.

b. {f : [T1 → [T2 → Boolean]], x : T1, y : T2} ⊢ (f x y) : Boolean

This statement is false:

1. 'f' is a function that takes T1 and returns a function [T2 → Boolean].
2. 'x' is of type T1, so (f x) will return a function [T2 → Boolean].
3. However, the statement (f x y) is not properly parenthesized.
4. It should be ((f x) y) to be correct. As written, (f x y) is not a valid application, so this statement is false.

c. {f : [T1 × T2 → T3], y : T2} ⊢ (lambda (x) (f x y)): [T1 → T3]

This statement is true:

1. 'f' takes a pair of T1 and T2 and returns T3.
2. 'y' is of type T2.
3. The lambda function takes 'x' as an argument (implicitly of type T1).
4. Inside the lambda, (f x y) is called, which matches the signature of f.
5. This lambda effectively takes T1 and returns T3. Therefore, the type of the lambda function is indeed [T1 → T3].

d. {f : [T2 → T1], x : T1, y : T3} ⊢ (f x) : T1

This statement is true:

1. 'f' is a function that takes T2 as input and returns T1.
2. 'x' is of type T1.
3. Although 'x' is of type T1 and not T2, the statement is only about the result type of (f x).
4. Regardless of whether the application is valid, if we were to evaluate (f x), the result type would be T1.
5. The type system is only concerned with the result type in this statement, not whether the application is valid.

Question 2:

Question 2.1: Simplifying TExps

a.(inter number boolean)

Simplified: never

Explanation: There are no values that are both numbers and booleans, so their intersection is empty.

b. (inter any string)

Simplified: string

Explanation: The intersection of 'any' (which includes all types) and 'string' is just 'string'.

c. (union any never)

Simplified: any Explanation: The union of 'any' with 'never' (empty set) is still 'any'.

d. (diff (union number string) string)

Simplified: number

Explanation: This is equivalent to all numbers and strings, minus all strings, which leaves just numbers.

e. (diff string (union number string))

Simplified: never

Explanation: This is equivalent to strings minus all strings (and numbers), which leaves nothing.

f. (inter (union boolean number) (union boolean (diff string never)))

Simplified: boolean

Explanation:

- (diff string never) simplifies to string
- (union boolean string) includes all booleans and strings
- The intersection of (boolean | number) and (boolean | string) is just Boolean

Question 2.2: Completing the code snippet

(define (good_in_L52 : ((union number boolean) -> boolean))

```
 (lambda ((z : (union number boolean))) : boolean

  (if (isBoolean z)

      z

      (> z 0))))
```

This function will pass type checking in L52 but not in L51 because:

1. It uses the type predicate 'isBoolean' which is only available in L52.
2. In L52, the type checker understands that if 'isBoolean z' is true, then z must be a boolean in the 'then' branch.
3. In L51, this would fail because z could be a number or a boolean, so returning z directly wouldn't be allowed.

Question 2.3: Completing the return type for f

(union string boolean number)

Explanation:

1. If x is a number (first branch of outer if):
   - It returns "positive" or "negative", both of type string
2. If x is a boolean (second branch of outer if):
   - If it's true, it returns the boolean true
   - If it's false, it returns the number 1

Therefore, the function can return a string, a boolean, or a number, which is represented by the union of these types.