

1. .... Function Subtyping

Assume that `Square <: Rectangle` and `Rectangle <: Shape`. Which of the following subtype relationships hold in principle?

- i) `(Rectangle → Rectangle) <: (Rectangle → Rectangle)`
- ii) `(Rectangle → Square) <: (Rectangle → Rectangle)`
- iii) `(Rectangle → Shape) <: (Rectangle → Rectangle)`
- iv) `(Shape → Rectangle) <: (Rectangle → Rectangle)`
- v) `(Square → Rectangle) <: (Rectangle → Rectangle)`
- vi) `(Shape → Square) <: (Rectangle → Rectangle)`
- vii) `(Square → Square) <: (Rectangle → Rectangle)`
- viii) `(Shape → Shape) <: (Rectangle → Rectangle)`
- ix) `(Square → Rectangle) → Shape <: (Rectangle → Square) → Shape`
- x) `(Square → Rectangle) → Shape <: (Square → Rectangle) → Rectangle`
- xi) `(Square → Square) → Square <: (Rectangle → Rectangle) → Rectangle`
- xii) `(Square → Square) → Square <: (Rectangle → Square) → Rectangle`

2. .... Subtyping and Specifications

In the Eiffel programming language, methods can have preconditions and postconditions. These are boolean expressions that must be true before the method is called and true afterwards. For example, the `pop` method in the following `Stack` class has a precondition `size > 0`. This means that in order for the `pop` method to execute properly, the stack should be nonempty beforehand. The postcondition `size' = size - 1` means that the size after executing `pop` will be one less than the size before this method is called. The syntactic convention here is that a “primed variable,” such as `size'`, indicates the value of that variable after execution of the method.

```
class Stack {
    ...
    int size
    ...
    void pop() pre: size > 0
                post: size' = size - 1 {
        ...
    }
    ...
}
```

This question asks you about subtyping when preconditions and postconditions are considered part of the type of a method.

- (a) In an implementation of stacks where each stack has a maximum size, given by a constant `MAX_SIZE`, the `push` method might have the following form:

```
class FixedStack {
    ...
    int size
    ...
    void push() pre: size < MAX_SIZE {
        ...
    }
    ...
}
```

while without this size restriction, we could have a stack class

```
class Stack {
    ...
    int size
    ...
    void push() pre: true {
        ...
    }
    ...
}
```

with the precondition of `push` always satisfied. (For simplicity, there are no postconditions in this example.) If this is the only difference between the two classes, which one should be considered a subtype of the other? Explain briefly.

- (b) Suppose that we have two classes that are identical except for the preconditions and postconditions on one method.

```
class A {
    ...
    int size
    ...
    void f() pre: PA, post: TA {
        ...
    }
    ...
}
class B {
    ...
    int size
    ...
    void f() pre: PB, post: TB {
        ...
    }
    ...
}
```

What relationships between `PA`, `TA`, `PB` and `TB` should hold in order to have `B <: A`? Explain briefly.