Section 7 Quiz

测验, 7 个问题

8 points

1。
Check the box if the statement is true.

- [ ] ML is dynamically typed.
- [ ] ML is weakly typed.
- [x] Racket is dynamically typed.
- [ ] Racket is weakly typed.

8 points

Check the box if the statement is true.

- [x] A "type system" that rejects every program is sound but useless.
- [ ] A "type system" that rejects every program is complete but useless
- [ ] A "type system" that accepts every program is sound but useless.
- [x] A "type system" that accepts every program is complete but useless.

8 points

3。
For each of the following, check the box if and only if it is an accurate description of an advantage of (sound) static typing (for preventing the sort of things that ML's type system prevents).

- [x] If you change the argument type of a function, the type-checker can give you a list of callers that no longer type-check as a result.
- [x] If you "comment out" a function and the program still type-checks, then you can be sure that no execution of the program will "try" to call the function.
- [ ] If you add a function to a program and the program still type-checks, then you can be sure that the added function will always be called when the program is executed.
- [ ] There is no reason to have exceptions in a programming language with static typing, so programmers do not have to worry about handling exceptions.

Section 7 Quiz

测验, 7 个问题

8 points

4。

For the remaining questions, we will consider changing ML's type system in the following way: We allow a function that takes a tuple argument `t1 * t2 * ... tn` to be called not only with a tuple of that exact type but also with *wider* tuples `t1 * t2 * ... tn * t(n+1) * t(n+2) * ...`. The typing rule for such calls is that the first *n* parts of the tuple must still have the "right" type for the call and the "extra" parts each need to have *some* type, but any type is fine. The evaluation rule for such calls is that the tuple argument is fully evaluated (including the "extra" parts) and the extra parts are simply inaccessible (ignored) by the function called.

Note the *only* typing rule we change is the one for function calls.

We assume the goals of the ML type system ("what it aims to prevent") are unchanged except that is okay to use these "too-wide tuples".

Check a box below if and only if the corresponding statement is true.

- [ ] ML *without* the change described above has a *sound* type system.
- [ ] ML *with* the change described above has a *sound* type system.
- [ ] ML *without* the change described above has a *complete* type system.
- [ ] ML *with* the change described above has a *complete* type system.

---

12 points

5。

Check the box below only if the ML code does *not* type-check in regular ML *and does* type-check if we make the change to the type system described in Question 4. All the code below uses this function:

```
1  fun f1 (x,y) = x + y
```

- [ ]
```
1  val z1 = f1 (3,4,5)
2
```

- [ ]
```
1  val z2 = f1 (3,4,x)
2
```

- [ ]
```
1  val p1 = (7,9)
2  val z3 = f1 p1
3
```

- [ ]
```
1  val p2 = (7,9,11)
2  val z4 = f1 p2
3
```

Section 7 Quiz

测验, 7 个问题

```
1  val z5 = if true then f1 (3,4) else f1 (5,6,7)
```

```
1  val z6 = f1 (if true then (3,4) else (5,6,7))
2
```

4 points

6.
Which of the reasons below is *not* an *advantage* of making the type-system change described in Question 4.

- If we had a function taking three arguments in a tuple, we could change it to take two arguments in a tuple without having to change any existing callers.
- In some situations, it lets you build one tuple that you can use to call multiple functions even if those functions take tuples of different widths.
- It makes it easier to convert between a multi-argument function using currying and one using tupling.
- It allows an expression like `foo(bar(),baz(),print "about to call foo")` to be equivalent to `foo(bar(),let val x = baz() in (print "about to call foo"; x) end)` while being shorter and probably easier to read.

4 points

7.
Which of the reasons below is *not* a *disadvantage* of making the type-system change described in Question 4.

- Passing too many arguments is often an error, but the type system will now not always prevent this error.
- It makes type inference more difficult due to polymorphism. For example, if `id` has `'a -> 'a`, should `id (3,4,5)` return a triple or a pair?
- It adds a special case to the language even though there are other places where a too-wide tuple could be allowed, for example in pattern-matching like `let val (x,y) = (3,4,5) in ... end`.
- It is a bad idea for functions to ignore some of their arguments, and it is impossible for a function in regular ML to ignore an argument.

我（**Feitao YI**）了解提交不是我自己完成的作业 将永远不会通过此课程或导致我的 Coursera 帐号被关闭。 了解荣誉准则的更多信息

Section 7 Quiz

测验, 7 个问题

Submit Quiz