

Homework # 10

due November 11

1 Reading

Please read Chapter 18 and 19 in the textbook.

2 Programming

Extend the counter class with a new “class” of variable counter class, with a new method `setinc` that sets the step size which is used by a new definition for `inc` that increments by the current step size. The code should use ascription so that the record types are given useful names. Use the `fullref` checker to check your code. Here is a simple test case:

```
vc1 = newVarCounter(2);
vc2 = newVarCounter(3);
inc3 vc1;
inc3 vc2;
vc1.get unit;
vc2.get unit;
vc1.setinc(1);
inc3 vc1;
inc3 vc2;
vc1.get unit;
vc2.get unit;
```

The last two numbers printed should be 10 and 19 (if you start your counters at 1).

Leave your code (including the definitions of `Counter`, `CounterRep` and related functions) in a file `varinc.f` in your `homework10` directory. Note that open recursion is not needed for this program.

3 Soundness of FJ

Complete the SASyLF proofs in the skeleton file (`fj.slf`) to prove that FJ is sound. FJ uses algorithmic typing rather than T-SUB which requires that the preservation theorem be permitted to compute a *subtype* for the result. On the other hand, the canonical forms lemma doesn't need to worry about subtyping any more.

Many of the lemmas dealing with the class table (looking up fields etc) are marked as optional. Do *NOT* prove these lemmas unless you finish the other theorems.

4 Extra

Optionally, finish the remaining lemmas in the FJ proof.

5 Graduate Students

1. Even in the absence of interfaces and conditional expressions, Java (and C++, C#, Scala) has problems with subsumption (T-SUB) because of static overloading resolution. Give a simple Java program that would type-check (and run) if subsumption were permitted for any expression, but that does not type-check in standard Java unless up-casts are added (in the place where T-SUB is used). Note that up-casts never fail. Hint: Your example will need to use overloaded functions where the options have different result types. (You may use C++ or C# if you know these languages better than Java.)
2. C++, Java and Cool can be more efficient, because the “self” object is passed as a parameter to the method rather than being available in the environment. Then calling (say) the `set` method of counter `c`, one would write:

```
(c.inc) c
```

assuming we have a counter such as:

```
c = { get = lambda self . !(r.x) ,  
      set = lambda self . lambda i . r.x := i,  
      inc = lambda self . (self.set) self (succ ((self.get) self)) };
```

Redo the open recursion through “self” examples from Chapter 18 using this approach (classes `SetCounter` and `InstrSetCounter`). Use `fulluntypedref` to test your implementation. Put your result in `object.f` in your `homework10` directory. You will be syntactically required to specify types for lambda-bound variables, but the types are ignored. Define `type SelfType = Unit`; and then use this for the type of `self`.

3. Of course `Unit` is the wrong type. What type should you use instead? Find a published reference on how to type “self” in object-oriented programming. Discuss!