

Homework # 13

due Thursday, December 4

1 Reading

Please read Chapters 23, 24 in your textbook.

2 Proofs

Prove the type soundness of pure System-F (see Figure 23-1, page 343) in SASyLF in the same style as previous proofs (canonical forms, progress, substitution and preservation). Unlike previous weeks, you are not given a skeleton file. You should start with an earlier proof, and take out anything not related to pure System F (unit, pairs, sums, fold etc) and add the new System F specific terms, contexts, values, evaluation forms and type rules. Because of a “feature” of SASyLF, you will need to write an unused judgment:

```
judgment istypevar: T in Gamma
assumes Gamma
```

```
----- type-var
X in (Gamma, X)
```

Use the partial solutions to Exercises 23.5.1 and 23.5.2 to help you write the proof. The solution to Exercise 23.5.1 mentions a new substitution lemma. This is a built-in proof rule (**substitution**) in SASyLF—you do not need to prove the lemma.

3 Programming

The programming questions should be done using the `fullomega` type checker, and you should copy in the church-numeral encoding of pairs and lists from `test.f` in the `fullomega` checker directory.

- Exercise 23.4.2 $\frac{1}{2}$ [$\star \not\rightarrow$]: Write a recursive `sum` function with type:

```
sum : (List Nat) → Nat
```

- Write another implementation of `sum` is *not* recursive but which has the same type and the same behavior. (You are permitted to still use a recursive `plus`.)
- Exercise 23.4.11 $\frac{1}{2}$ [$\star\star \not\rightarrow$]: Write a *non-recursive* `map` using the Church encoding of lists. It should have the same type as on page 346.
- Exercise 24.2.5 $\frac{1}{2}$ [$\star\star\star \not\rightarrow$]: The `List` type defined on page 351 exposes the internal representation. For example, we couldn't substitute an implementation using recursive types. The following definition fixes this problem

```
OOList = lambda X.
  {Some R, {state:R, nil:R,
            isnil: R->Bool,
            cons: X->R->R,
            head: R->X,
            tail: R->R}};
```

- Write a term `oonil` that has type $\forall X.OOList\ X$; use the pre-existing `List X` definition.
- Write definitions of `ooisnil`, `oocons`, `oohead`, `ootail` so that they have the following types:

```
ooisnil : ∀X. (OOList X) → Bool
oocons  : ∀X. X → (OOList X) → (OOList X)
oohead  : ∀X. (OOList X) → X
ootail  : ∀X. (OOList X) → (OOList X)
```

(c) Write `oomap` to use these primitives (you may assume `fix`). Test your program by running

```
oohead[Bool]
(oomap[Int] [Bool] iseven
 (oocons[Nat] 1 (oocons[Nat] 2 (oonil[Nat]))))
```

Leave all your code in `list.f`.