

Homework # 13

due Tuesday, December 13

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 very much in the 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. (Up to now, we had only one binding context in Gamma, now we will need another binding: Γ, 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. That is given for you in the (fragmentary) skeleton file. This fragment implies the exists of a judgment $\Gamma \vdash T$ and rules B-VAR, B-ARROW and B-ALL.

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:

$$\text{sum} : (\text{List Nat}) \rightarrow \text{Nat}$$

- 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}};

```

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

```

ooisnil :  $\forall X. (\text{OOList } X) \rightarrow \text{Bool}$ 
oocons  :  $\forall X. X \rightarrow (\text{OOList } X) \rightarrow (\text{OOList } X)$ 
oohead  :  $\forall X. (\text{OOList } X) \rightarrow X$ 
ootail  :  $\forall X. (\text{OOList } X) \rightarrow (\text{OOList } X)$ 

```

- (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`.

4 Graduate Students

Read the section on erasure and type reconstruction. Answer the following questions:

1. What if (unlike 23.6.2), we keep actual type parameters, but only erase types for lambda abstraction (as was done last week). Can we use the constraint-based technique from last week? Why or why not? If we can use it, then do we have totality, soundness and completeness? Why or why not? You are encouraged to find citations.
2. Give a *useful* example of *polymorphic recursion* and how it can be typed in System F, but not in ML. Is this example Rank 2 (see page 359)?
3. Why does theorem 23.6.2 (proved first) not provide a proof for 23.6.1 (proved later)? Explain.