

Assignment 2

Progress and preservation

15-814: Types and Programming Languages
Jan Hoffmann & C.B. Aberlé

Due Tuesday, September 19, 2023
75 pts

You should hand in two files

- `hw02.pdf` with your written solutions to the questions.
- `hw02.lam` with the code, where the solutions to the problems are clearly marked and auxiliary code (either from lecture or your own) is included so it passes the LAMBDA checker.

1 Y Combinator

Task 1 (Lambda 15 pts) The Lucas function (a variant on the Fibonacci function) is defined mathematically by

$$\begin{aligned}\text{lucas } 0 &= 2 \\ \text{lucas } 1 &= 1 \\ \text{lucas } (n + 2) &= \text{lucas } n + \text{lucas } (n + 1)\end{aligned}$$

Give an implementation of `lucas` using the Y combinator. If you used the Y -combinator in your previous implementation of `lucas` from Assignment 1, give an implementation of `lucas` using primitive recursion instead. Provide both implementations in your code, for comparison.

You may copy the functions from [nat.lam](#) to the beginning of your file `hw02.lam` to use as helper functions. Test your new implementation on inputs 0, 1, 9, and 11, expecting results 2, 1, 76, and 199.

In the previous homework, you recorded the number of β -reductions taken by your implementation of `lucas`. Compare this record to your new implementation. Which of the two implementations is more “efficient” (in the sense of number of β -reductions)?

2 Proof by Rule Induction

Since this is the first time we (that is, you) are proving theorems about judgments defined by rules, we ask you to be very explicit. In particular:

- We already provided state the overall structure of your proofs: they proceed by rule induction and the derivation of a judgment that we identified. In general, we ask you to explicitly

state the structure of your proof: whether it proceeds by rule induction, and, if so, on the derivation of which judgment, or by structural induction, or by inversion, or just directly. If you need to split out a lemma for your proof, state it clearly and prove it separately. If you need to generalize your induction hypothesis, clearly state the generalized form.

- Explicitly list all cases in an induction proof. If a case is impossible, prove that it is impossible. Often, that's just inversion, but sometimes it is more subtle.
- Explicitly note any appeals to the induction hypothesis.
- Any appeals to inversion should be noted as such, as well as the rules that could have inferred the judgment we already know. This could lead to zero cases (a contradiction—the judgment could not have been derived), one case (there is exactly one rule whose conclusion matches our knowledge), or multiple cases, in which case your proof now splits into multiple cases.
- We recommend that you follow the line-by-line style of presentation where each line is justified by a short phrase. This will help you to check your proof and us to read and verify it.

3 Progress and preservation for System T

For the following questions, please refer to System T handout for the static and dynamic semantics of System T. We will work with the *call-by-value* operational semantics of System T. You may assume the following lemmas without proof:

Lemma 1 (Admissibility of substitution) *If $\Gamma \vdash e : \tau$ and $\Gamma, x : \tau \vdash e' : \tau'$, then $\Gamma \vdash e'[e/x] : \tau'$.*

Lemma 2 (Admissibility of weakening) *If $\Gamma \vdash e : \tau$, then $\Gamma, x : \tau' \vdash e : \tau$ for any variable x that does not occur in Γ .*

Lemma 3 (Canonical forms lemma) *Given $e : \tau$ and e val, the following holds:*

1. *If $\tau = \text{nat}$, then $e = \text{zero}$ or $e = s(e')$ for some e' .*
2. *If $\tau = \tau_1 \rightarrow \tau_2$, then $e = \lambda x : \tau_1. e'$ for some e' .*

Lemma 4 (Inversion for typing) *Given the conclusion of a typing rule, we have that the premises of the rule hold as well.*

Theorem 5 (Preservation (PFPL, Theorem 9.3 (1))) *If $e : \tau$ and $e \mapsto e'$, then $e' : \tau$.*

The proof proceeds by induction on the derivation of $e \mapsto e'$. As an example, the cases for E_{ap1} and E_{ap3} look as follows:

Proof: Case:

$$\frac{E_{\text{AP1}} \quad e_1 \mapsto e'_1}{e_1(e_2) \mapsto e'_1(e_2)}$$

Suppose that $e_1(e_2) : \tau$. We need to show that $e'_1(e_2) : \tau$. By inversion for T_{ap} , we know that $\tau = \tau_1 \rightarrow \tau_2$, $e_1 : \tau_1 \rightarrow \tau_2$, and $e_2 : \tau_1$. By the induction hypothesis on e_1 , we have that $e'_1 : \tau_1 \rightarrow \tau_2$,

so the result follows by applying T_{ap} .

Case:

$$\frac{E_{\text{AP3}}}{(\lambda x : \tau_2. e)(e_2) \mapsto e[e_2/x]}$$

Suppose that $(\lambda x : \tau_2. e)(e_2) : \tau$. We want to show that $e[e_2/x] : \tau$. By applying inversion twice (once on T_{ap} and once on T_{lam}), we have that $x : \tau_2 \vdash e : \tau$ and $e_2 : \tau_2$ for some τ_2 . Therefore we may apply Lemma 1 to obtain $e[e_2/x] : \tau$. \square

Task 2 (30pts) Write out the case for the recursor E_{rec3} .

Theorem 6 (Progress (PFPL, Theorem 9.3 (2))) Given $e : \tau$, either $e \mapsto e'$ for some e' or e val.

The proof proceeds by induction on the derivation of $e : \tau$. As an example, the case for the successor T_s should look as follows:

Proof: Case:

$$\frac{T_s \quad e : \text{nat}}{s(e) : \text{nat}}$$

We have to prove that $s(e) \mapsto e'$ for some e' or $s(e)$ val. By the induction hypothesis, we know that either $e \mapsto e''$ for some e'' or e val. Proceed by cases. If $e \mapsto e''$, then $s(e) \mapsto s(e'')$ by E_s , and so we may take $e' = s(e'')$. Otherwise, we have e val, and so $s(e)$ val by V_s . \square

Task 3 (30pts) Write out the case for abstraction T_{lam} and application T_{ap} .