Homework for Tuesday September 23, 2014

1 Type theory and lambdas

- What are the types of the following expressions?
  1. devour: $\langle e, \langle e, t\rangle\rangle$
  2. fond: $\langle e, \langle e, t\rangle\rangle$
  3. part of New Brunswick: $\langle e, t\rangle$
  4. show Porky: $\langle e, \langle e, t\rangle\rangle$
  5. white cat from Providence: $\langle e, t\rangle$

- Evaluating claims.
  1. $(\lambda x.\lambda y.\mathit{licks}'(x)(y))(\mathrm{u})(\mathrm{p}) =_\beta (\lambda y.\mathit{licks}'(\mathrm{u})(y))(\mathrm{p})$
     $=_\beta \mathit{licks}'(\mathrm{u})(\mathrm{p})$ ✓
  2. $(\lambda x.f(x)(y))(y) =_\beta f(y)(y)$ ✓
     - ▷ This is a valid β-reduction. No variable that was unbound before the reduction becomes bound after the reduction. In other words, there's no variable capture and no need for an α-conversion to avoid a potential variable capture.
  3. $(\lambda x.\lambda y.f(x)(y))(y) =_\alpha (\lambda x.\lambda z.f(x)(z))(y)$
     $=_\beta \lambda z.f(y)(z)$ ✓
     - ▷ Unlike the previous example, we need to exploit an α-equivalence to avoid capturing the free y when we perform a β-reduction. Otherwise, y would start out free but become bound.
  4. $\lambda x.\mathit{kiss}'(x) =_\alpha \lambda y.\mathit{kiss}'(y)$
     $=_\eta \lambda y.\lambda x.\mathit{kiss}'(y)(x)$ ✓
     - ▷ This one turned out to be very tricky; almost nobody got it right. It exploits both an α-equivalence and an η-equivalence. Simply saying that the two are equivalent since they take their arguments in the same order isn't quite explicit enough. If you don't see how the η-equivalence works, get in touch.
  5. $\lambda x.\mathit{kiss}'(x) =_\eta \lambda x.\lambda y.\mathit{kiss}'(x)(y)$ ✗
     - ▷ Again, it's important to say why the equivalence fails, by appealing to the rules of the lambda calculus that make it so. The right-hand side here is α-equivalent to the term we obtained in the previous example (but the right-hand side in the assignment is not); you might see if you can verify this.

- Simplify the following expressions as much as possible.
  1. $(\lambda x.\mathit{kiss}'(x)(y))(y) =_\beta \mathit{kiss}'(y)(y)$
     - ▷ As above, there is no issue of variable capture here. No variable that was free prior to the β-reduction is bound after it.
  2. $(\lambda P.\lambda x.P(x))(\mathit{run}') =_\beta \lambda x.\mathit{run}'(x)$
     $=_\eta \mathit{run}'$
     - ▷ The η-equivalence wasn't necessary, but many of you saw it anyway.
  3. $(\lambda R.R(a)(b))(\lambda y.\lambda x.\mathit{kiss}'(y)(x)) =_\beta (\lambda y.\lambda x.\mathit{kiss}'(y)(x))(a)(b)$
     $=_\beta (\lambda x.\mathit{kiss}'(a)(x))(b)$
     $=_\beta \mathit{kiss}'(a)(b)$
  4. $(\lambda f.f(x))(\lambda y.\lambda x.g(x)(y)) =_\beta (\lambda y.\lambda x.g(x)(y))(x)$
     $=_\alpha (\lambda y.\lambda z.g(z)(y))(x)$
     $=_\beta \lambda z.g(z)(x)$

▷ Many of you did one more simplification and wound up with $g(x)$. It's important to see that $\lambda z.g(z)(x)$ is distinct from $(\lambda z.g(z))(x)$. The former is fully reduced (in *normal form*). The latter is not.

5. $(\lambda\mathcal{P}.\mathcal{P}(\lambda p.p))(\lambda k.k(meows'(x))) =_\beta (\lambda k.k(meows'(x)))(\lambda p.p)$
   $=_\beta (\lambda p.p)(meows'(x))$
   $=_\beta meows'(x)$

- A function in the simply-typed lambda calculus can **never** apply to itself. Suppose f has type $\langle\sigma,\tau\rangle$ (for some type σ and some type τ). Then $f(f)$ is not defined. For $f(f)$ to be defined, it would have to be the case, impossibly, that $\langle\sigma,\tau\rangle = \sigma$, since functional application is only defined when the functor has type $\langle\alpha,\beta\rangle$ and the argument has type α (for some types α and β). A few answers referenced recursion. While it's true that natural language seems to be recursive in the sense that, say, DPs can contain other DPs (and, by the same token, expressions of type e can be built up from other expressions of type e), this is a different thing from a function applying to itself.

2 Composition inside DP

- Four derivations follow. Figures 1 and 2 give two possible derivations for the sentence in question using **FA** and **PM**. In both cases, **FA** combines is, a, from, and the composite predicate is a big white cat from PVD with their arguments, and **PM** applies everywhere else. Figure 3 gives a derivation using only $\text{mod}_\emptyset$ and **FA**. Figure 4 assigns a meaning to an unusual parse of the string in question. You should find this mildly disturbing: is a big white doesn't seem like much of a constituent (e.g. you *really* can't topicalize it). All four derivations **yield the same results**.

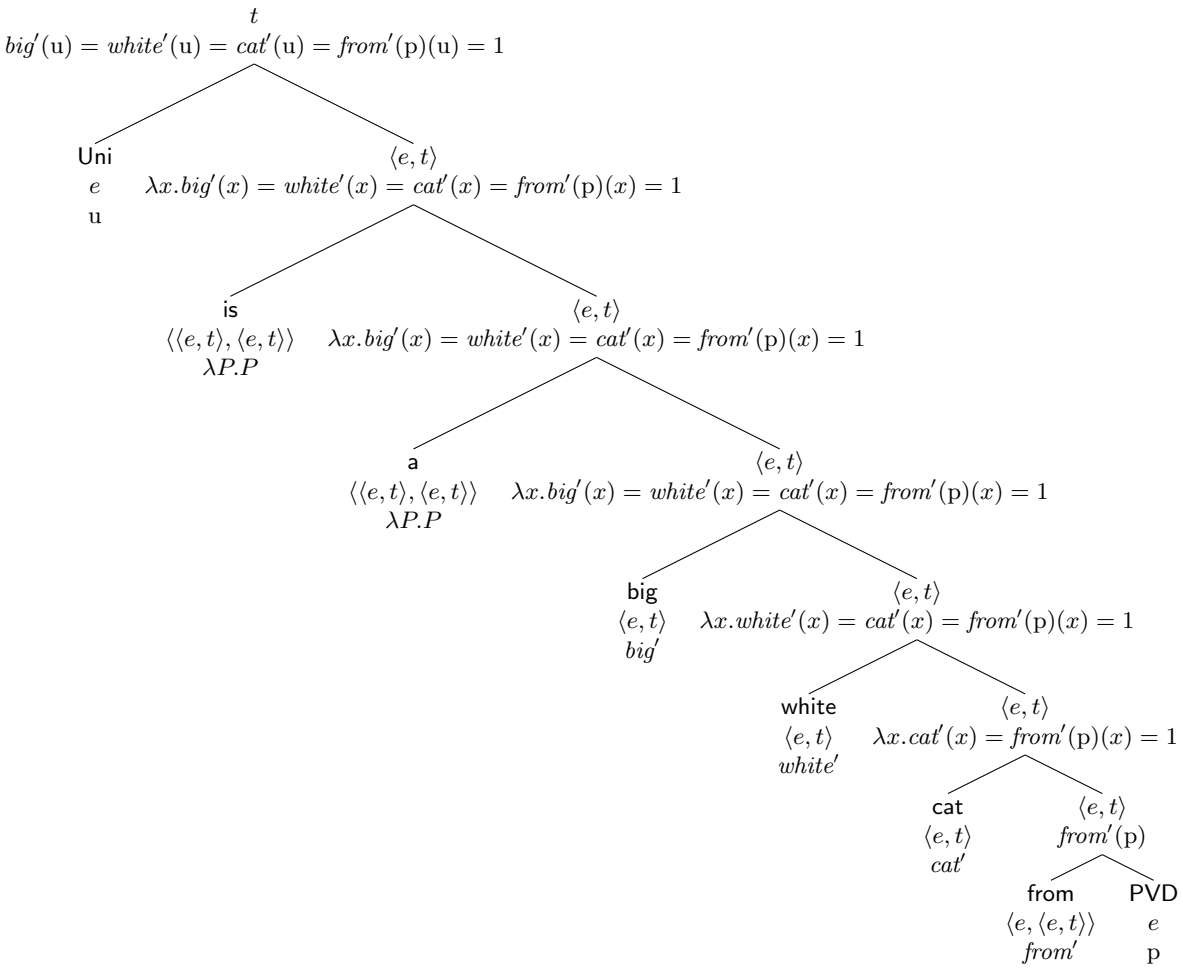

Figure 1: One derivation for ⟦Uni is a big white cat from Providence⟧

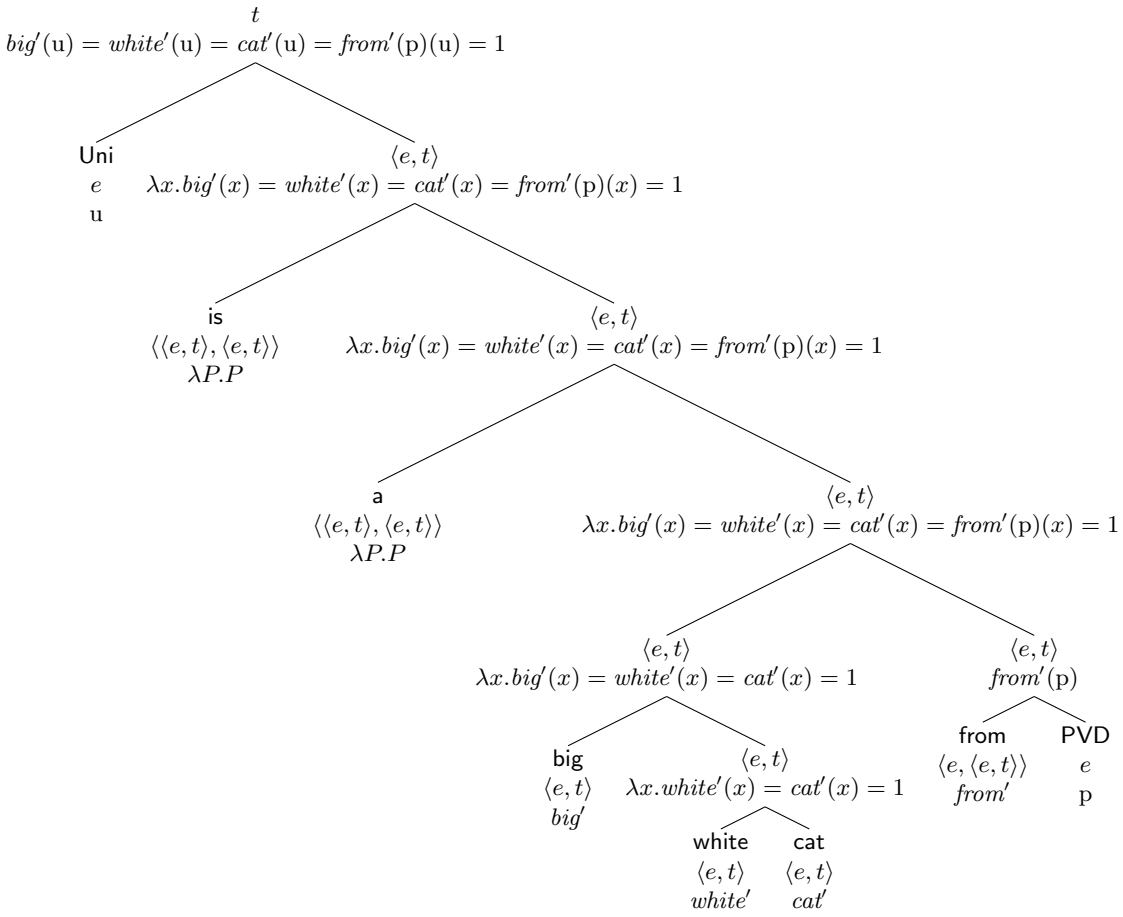

Figure 2: Another derivation for ⟦Uni is a big white cat from Providence⟧

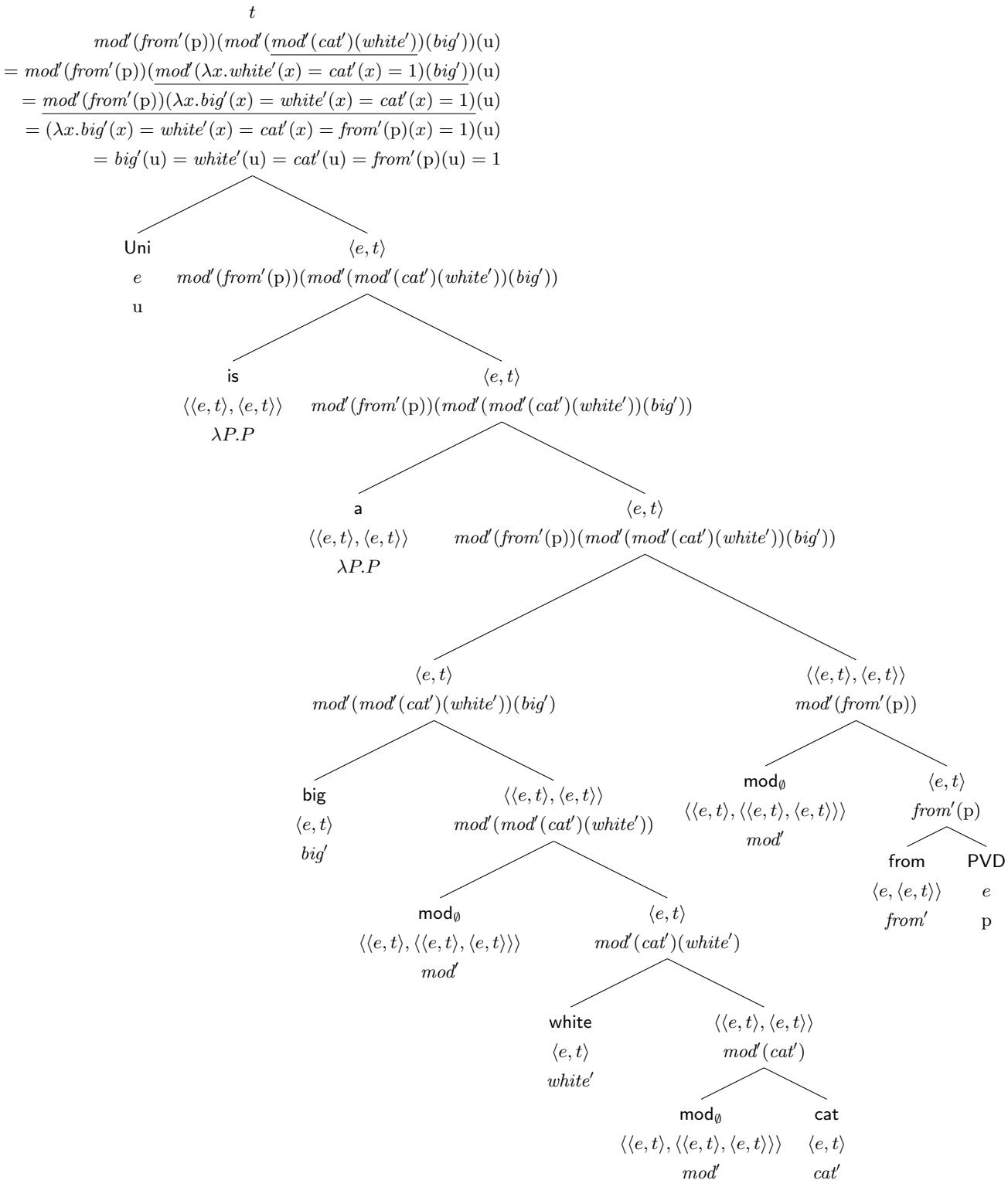

Figure 3: $[\![$Uni is a big white cat from Providence$]\!]$ using $\mathsf{mod}_\emptyset$ and only **FA**. $[\![\mathsf{mod}_\emptyset]\!]$ denotes a function that takes two properties Q and P and returns a third, the result of "intersecting" P with Q. More formally, $[\![\mathsf{mod}_\emptyset]\!] = \lambda P.\lambda Q.\lambda x.Q(x) = P(x) = 1$, and for any Q and P, $[\![\mathsf{mod}_\emptyset]\!](Q)(P) = \lambda x.P(x) = Q(x) = 1$.

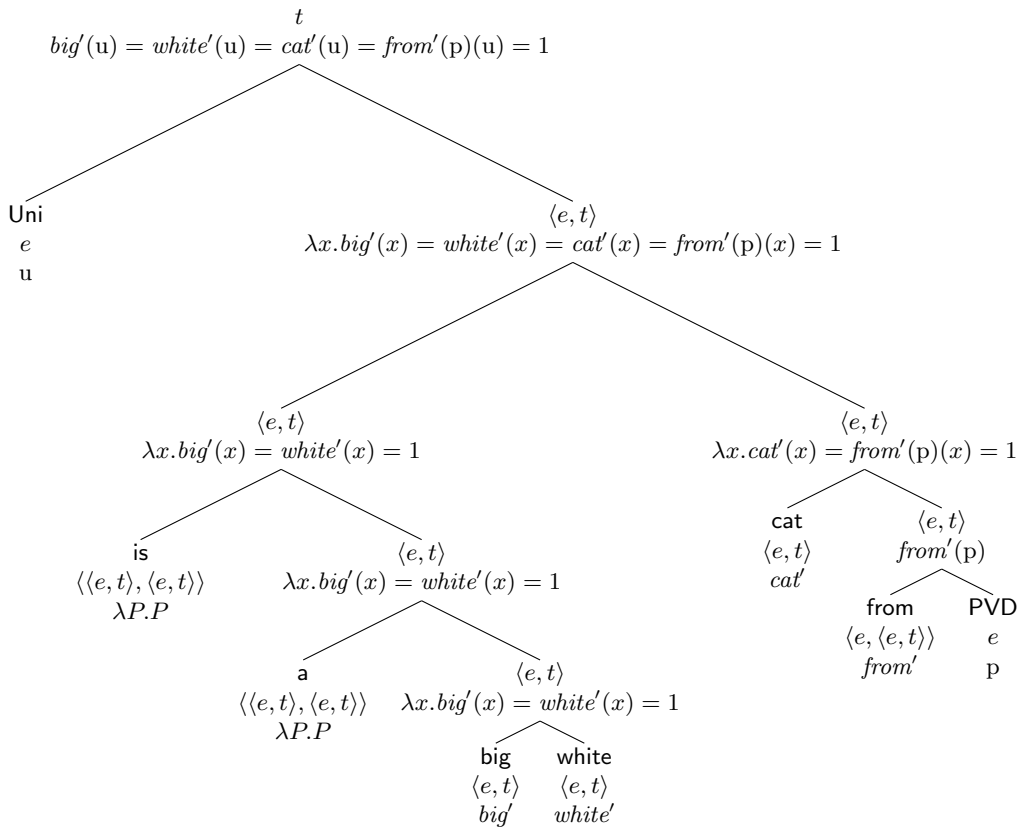

Figure 4: Strange derivation for ⟦Uni is a big white cat from Providence⟧