24.903
Language & Structure III: Semantics and Pragmatics
Spring 2003, 2-151, MW 1-2.30
March 5, 2003
Assignment 4, due in class on March 12

1. Assume that the meaning of **spouse**, $[\![spouse]\!]$, is a function that maps (married) people to their spouses (which assumes monogamy), and that the meaning of **mother** is a function that maps people to their mother. Define the meaning of **mother in law** in terms of the meaning of **spouse** and **mother**, using the λ notation.

Using function composition, we can say that
$[\![mother\ in\ law]\!] = [\![mother]\!] \circ [\![spouse]\!]$
To get $[\![mother\ in\ law]\!]$ in λ terms, therefore we need to function compose $[\![mother]\!]$ and $[\![spouse]\!]$. We know that $\lambda f.\lambda g.\lambda x.[f(g(x))]$ when applied to functions f and g returns $f \circ g$, the composition of f and g. Let us therefore apply $\lambda f.\lambda g.\lambda x.[f(g(x))]$ to $\lambda x.[mother(x)]$ and $\lambda x.[spouse(x)]$ in sequence.
$\lambda f.\lambda g.\lambda x.[f(g(x))](\lambda x.[mother(x)])(\lambda x.[spouse(x)])$
$\lambda g.\lambda x.[(\lambda x.[mother(x)])(g(x))](\lambda x.[spouse(x)])$
$\lambda g.\lambda x.[mother(g(x))](\lambda x.[spouse(x)])$
$\lambda x.[mother((\lambda x.[spouse(x)])(x))]$
$\lambda x.[mother(spouse(x))]$
Many of you got this one wrong. This was largely because many of you assumed the following definitions for $[\![mother]\!]$ and $[\![spouse]\!]$ respectively: $\lambda x\lambda y.[mother(x)(y)]$, and $\lambda x\lambda y.[spouse(x)(y)]$. By these definitions, *mother* and *spouse* have types eet and not type ee as was intended. Once this incorrect assumption had been made, what followed was automatically incorrect.

2. Let j be a constant of type e; M of type $< e,t >$; S of type $<< e,t >,< e,t >>$; and R of type $<< e,t >,t >$. Furthermore x is a variable of type e, and Y is a variable of type $< e,t >$. Determine which of the following sequences are well-formed expressions. If an expression is well-formed, give its type. Assume that the operator $\wedge$ (and) requires both its arguments to be of type t.

a. $\lambda x[M(x)](R)$
No.
b. $\lambda x[M(x)](j)$
t
c. $\lambda x[M(j)]$
et
d. $S(\lambda x[M(x)])$
et
e. $\lambda Y[Y(j)](M)$
t
f. $\lambda x[\lambda Y[Y(x)]]$
$e(et)t$
g. $\lambda x[M(x)] \wedge M(j)$
No.
h. $\lambda x[M(x) \wedge M(j)]$
et
i. $(S(\lambda Y[Y(x)]))(M)$
No.
j. $\lambda Y[R(\lambda x[Y(x)])](M)$
t
k. $\lambda x[\lambda Y[Y(x)](M)](j)$
t
l. $\lambda x[\lambda Y[Y(x)](j)](M)$
No.
m. $\lambda x[\lambda Y[Y(x)]](j)(M)$
t
n. $\lambda Y[(S(\lambda x[M(x)]))(j) \wedge R(Y)](M)$
t

3. Reduce the following λ terms as far as possible. Note that hese are λ terms that take functions as their

arguments; f, g etc. have been used as variables over functions.
a) $\lambda f[f(3)](\lambda y[5+y])$
$\lambda y[5+y](3)$
$5+3$
8

b) $\lambda f[f(3)(4)](\lambda x\lambda y[x+y])$
$\lambda x\lambda y[x+y](3)(4)$
$\lambda y[3+y](4)$
$3+4$
7

c) $\lambda f\lambda g\lambda x[g(5)(f(2)(x))](\lambda x\lambda y[x+y])(\lambda x\lambda y[x-y])(8)$
$\lambda g\lambda x[g(5)((\lambda x\lambda y[x+y])(2)(x))](\lambda x\lambda y[x-y])(8)$
$\lambda g\lambda x[g(5)((\lambda y[2+y])(x))](\lambda x\lambda y[x-y])(8)$
$\lambda g\lambda x[g(5)(2+x)](\lambda x\lambda y[x-y])(8)$
$\lambda x[\lambda x\lambda y[x-y](5)(2+x)](8)$
We can do either inner or outer λ conversion here:
inner:
$\lambda x[\lambda y[5-y](2+x)](8)$
$\lambda x[5-(2+x)](8)$
$\lambda x[3-x](8)$
$3-8$
-5
outer:
$\lambda x\lambda y[x-y](5)(2+8)$
$\lambda x\lambda y[x-y](5)(10)$
$\lambda y[5-y](10)$
$5-10$
-5
The order in which you do various λ-conversions does not influence the final output.

d) $\lambda f[\lambda x[\text{the mother of } f(x)](Jo)](\lambda x[\text{the father of } x])$
$\lambda f[\text{the mother of } f(Jo)](\lambda x[\text{the father of } x])$ (doing internal λ-conversion first)
the mother of $\lambda x[\text{the father of } x](Jo)$
the mother of the father of Jo

4. Given a λ expressions, we can usually make inferences regarding the relationships between the types of the expressions that make up the λ expression. For example, if we assume the type of X to be σ and the type of Y to be τ, then the type of $\lambda X[Y]$ is $<\sigma,\tau>$. Similarly if a function f applies to an argument x and x is of type γ, then the type of f must be of the format $<\gamma, ??>$.
Given this background what can you say about the type of f and the whole λ expression in:
$\lambda f[f(f)](\lambda f[f(f)])$

It is not possible to assign any type to f, $\lambda f[f(f)]$, or $\lambda f[f(f)](\lambda f[f(f)])$. To see why this is so, assume that it is in fact possible to assign f a type. Let this type be σ. Since f takes f as an argument, the type of the outer f must be $<\sigma,\tau>$, for some τ. But the type of outer f and the inner f cannot be distinct. f can be of type σ, or of type $<\sigma,\tau>$, but not both simultaneously.[1] A similar line of argument shows that it is impossible to assign a type to $\lambda f[f(f)]$. Consequently, it is also not possible to assign a type to $\lambda f[f(f)](\lambda f[f(f)])$.
Another way of thinking about this curious expression: assume that the type of f is σ, then the result of applying f to f is not well-formed because there is no way to apply a function of type σ to an argument of

[1] One way to remember this is through the dictum: the type of a function is always *bigger* than the type of its argument.

type σ. Hence the entire expression is just not well-formed *with respect to types.* The paradoxical nature of this expression stems from our unwarranted asumption that it is possible to assign types to all well-formed λ expressions. In this course we will only concern ourself with that subset of well-formed λ expressions to which types can be assigned. This subset is called the well-typed λ calculus. $\lambda f[f(f)](\lambda f[f(f)])$ is a well-formed expression of the **pure** λ calculus, but not of the well-typed λ calculus.

From Heim & Kratzer

1. Exercise on Page 66.

4. Exercise on page 66.
Structure assumed:
Kaline is a [[[gray cat] [in Texas]] [fond of Joe]]
$[\![gray\ cat]\!] = \lambda x \in D_e.[cat(x) \wedge gray(x)]$
$[\![in\ Texas]\!] = \lambda x \in D_e.[in(Tx)(x)]$
$[\![fond\ of\ Joe]\!] = \lambda x \in D_e.[fond(Joe)(x)]$
Combining via PM
$[\![gray\ cat\ in\ Texas]\!] = \lambda x \in D_e.[cat(x) \wedge gray(x) \wedge in(Tx)(x)]$
Combining via PM
$[\![gray\ cat\ in\ Texas\ fond\ of\ Joe]\!] = \lambda x \in D_e.[cat(x) \wedge gray(x) \wedge in(Tx)(x) \wedge fond(Joe)(x)]$
$[\![Kaline\ is\ a\ gray\ cat\ in\ Texas\ fond\ of\ Joe]\!] =$
$cat(Kl) \wedge gray(Kl) \wedge in(Tx)(Kl) \wedge fond(Joe)(Kl)$

2. Exercise 1 on Page 67.

Structure assumed:
Kaline is a [[[gray cat] [in Texas]] [fond of Joe]]
$[\![gray]\!] = \lambda f \in D_{et} \lambda x \in D_e.[f(x) \wedge gray(x)]$
Combining via FA
$[\![gray\ cat]\!] = \lambda x \in D_e.[cat(x) \wedge gray(x)]$
$[\![in\ Texas]\!] = \lambda f \in D_{et} \lambda x \in D_e.[f(x) \wedge in(Tx)(x)]$
Combining via FA
$[\![gray\ cat\ in\ Texas]\!] = \lambda x \in D_e.[cat(x) \wedge gray(x) \wedge in(Tx)(x)]$
$[\![fond\ of\ Joe]\!] = \lambda f \in D_{et} \lambda x \in D_e.[f(x) \wedge fond(Joe)(x)]$
Combining via FA
$[\![gray\ cat\ in\ Texas\ fond\ of\ Joe]\!] = \lambda x \in D_e.[cat(x) \wedge gray(x) \wedge in(Tx)(x) \wedge fond(Joe)(x)]$
$[\![Kaline\ is\ a\ gray\ cat\ in\ Texas\ fond\ of\ Joe]\!] =$
$cat(Kl) \wedge gray(Kl) \wedge in(Tx)(Kl) \wedge fond(Joe)(Kl)$

3. Exercise 2 on Page 67. Present both solutions.

be_1 in D_{et}
$[\![be_1]\!] = \lambda x \in D_e.[1]$

be_2 in $D_{((et)et)et}$
$[\![be_2]\!] = \lambda F \in D_{(et)et}.[F(\lambda x \in D_e.[1])]$