We've been assigning the same syntactic types to nouns like *woman* and verb phrases like *loves John* or *sleeps*. There is an obvious error in doing so. Under this assumptions the sentence *Every woman woman* must be as grammatical as *Every woman loves John*, but it isn't. Strictly speaking, English nouns are not expressions that take a noun phrase to their right to form sentences. From here on we will correct this error by taking nouns to be of type N, rather than $S\backslash NP$. With this move, we will have two different syntactic types with the same semantic type, namely, both $S\backslash NP$ and N will be of type et. There is no harm in this, since semantic types are the types of the logical interpretation of an expression.

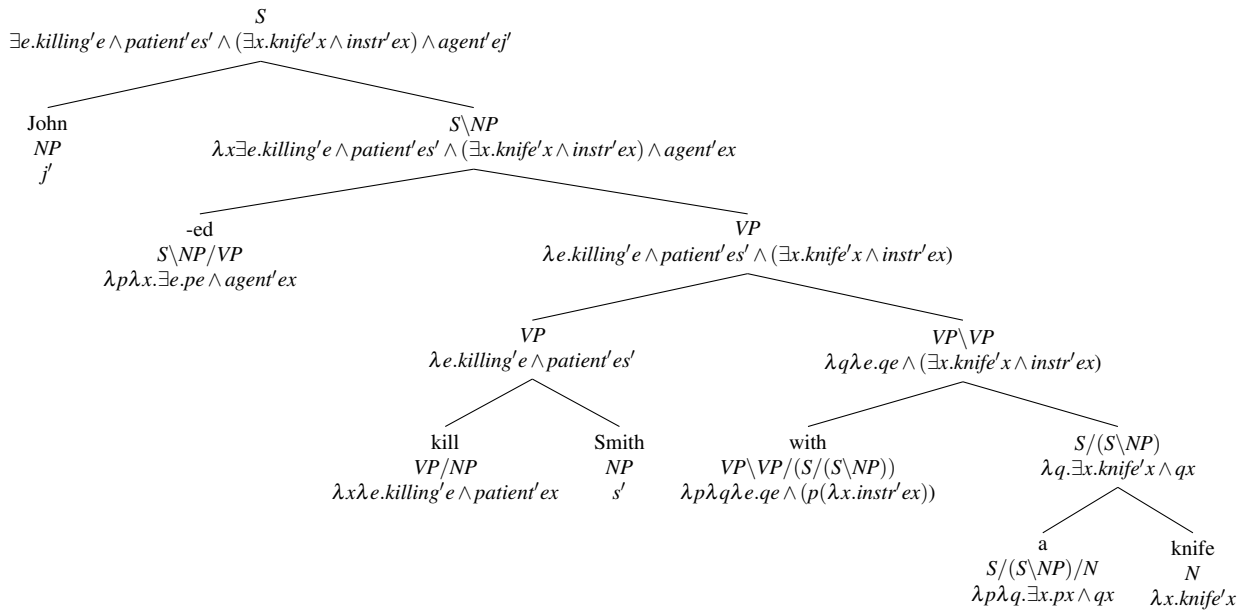
Altering the syntactic category of nouns will effect the category of determiners/quantifers. We also add the object position categories of *a* and *every*. And, we update the type declarations to the minimalist format. Here is a simple lexicon:

(1)

sleeps :=	$S\backslash NP$	$:\lambda x.sleep'x$	$::et$
loves :=	$(S\backslash NP)/NP$	$:\lambda x\lambda y.love'xy$	$::e(et)$
John :=	NP	$:j'$	$::e$
Mary :=	NP	$:m'$	$::e$
woman :=	N	$:\lambda x.woman'x$	$::et$
a :=	$(S/(S\backslash NP))/N$	$:\lambda p\lambda q.\exists x.px \wedge qx$	$::et(ett)$
a :=	$(S\backslash NP)\backslash(S\backslash NP/NP)/N$	$:\lambda p\lambda q\lambda z.\exists x.px \wedge qxz$	$::?$
every :=	$(S/(S\backslash NP))/N$	$:\lambda p\lambda q.\forall x.px \rightarrow qx$	$::et(ett)$
every :=	$(S\backslash NP)\backslash(S\backslash NP/NP)/N$	$:\lambda p\lambda q\lambda z.\forall x.px \rightarrow qxz$	$::?$

If you remeber, when we were integrating event semantics to our interpretative model, we discussed the benefits of, or reasons for, making subjects totally external to the verb phrase. Here is the derivation of our previous example:

(2) John killed Smith with a knife.



Q1.

Complete the missing semantic types of *every* and *a* in Lexicon 1. Eliminate as many parentheses as you can.

Solution: $et(e(et)(et))$

Q2.

Let us assume that eventualities are of a different type than other individuals. Let's say s. We now have three types: t for truth values, s for eventualities, and e for individuals.

Give the semantic types of *-ed*, *with*, and *kill Smith* in 2.

Solution: *kill Smith* : st
-ed : $st(et)$
with : $ett(st(st))$

Q3.

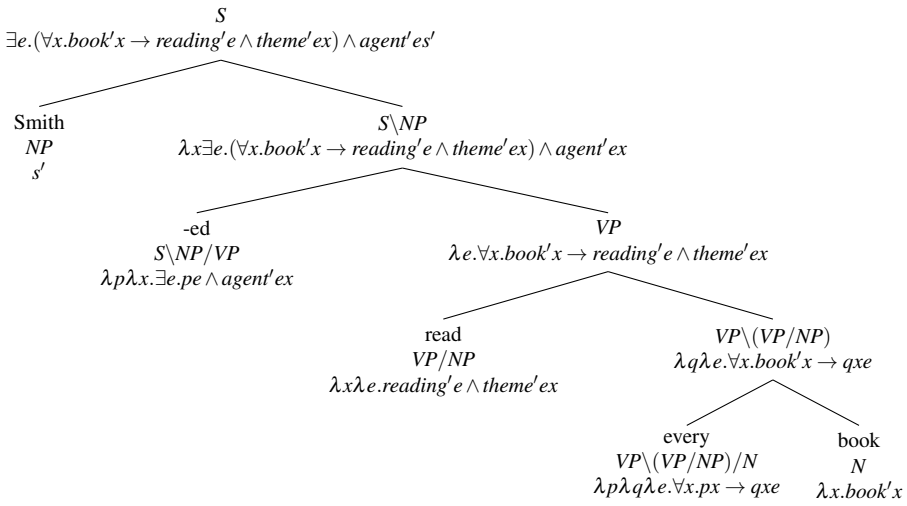
Derive the meaning of the following sentence, including the event semantics.

(3) Smith read every book.

Solution:

In the semantics of *read*, we call the object "theme", rather than "patient", in accord with the general practice. But the choice is not ultimately important.

Here is a first shot at deriving a meaning for (3):



The semantic interpretation we arrived at does not look satisfactory. A more accurate interpretation would be:

(4) $\exists e.reading'e \wedge (\forall x.book'x \rightarrow theme'ex) \wedge agent'es'$

This is an interpretation that we take the reading of all the novels as a single event. Of course one can take the meaning to encode separate reading events for each book, in which case the right interpretation would be:

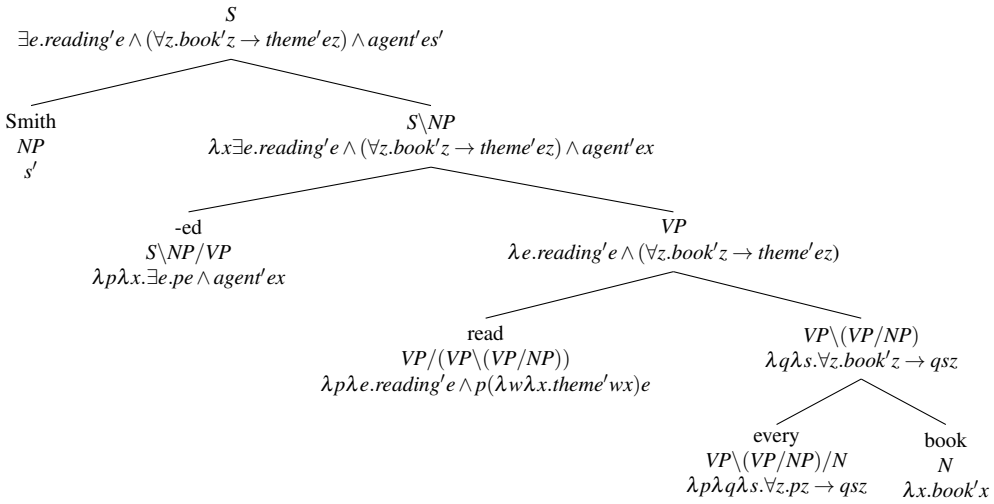
(5) $\forall x.book'x \rightarrow \exists e.reading'e \wedge theme'ex \wedge agent'es'$

Let's leave this "distrubitive" reading aside for the moment, and concentrate on (4). Is this reading different from the one we derived above? Unfortunately it is. Take a model where there are no books and Smith wrote a letter. In such a model, the interpretation we derived above comes out true, but (4) comes out false. Therefore, we were unable to derive the correct interpretation.

What we need to be able to derive the correct interpretation is to "value raise" the verb *read* as follows:

(6) read := $VP/(VP\backslash(VP/NP))$: $\lambda p \lambda e.reading'e \wedge p(\lambda w \lambda x.patient'wx)e$

Here is the derivation for the sentence with this lexical assumption, where we make some α-conversions to avoid possible confusions:



Well, this is complicated. But, don't worry, we will discuss whether this complexity pays back anything.