

## 1. Formation Rules

### 1. Variables

For each type  $\mathfrak{J}$ , there is an infinite list of variables of type  $\mathfrak{J}$ . Note, however, that we employ just a few types of variables, which are type-encoded as follows.

(1)	lower-case math-italic	$x, y, z, \dots$	D
(2)	upper-case times-roman	P, Q, R, ...	$D \rightarrow S$
(3)	upper-case block	$\mathbb{P}, \mathbb{Q}, \mathbb{R}, \dots$	$(D \rightarrow S) \rightarrow S$
(4)	upper-case Greek	$\Phi, \Psi, \dots$	S

### 2. Proper Expressions

The proper expressions of a language are the expressions peculiar to that language. For example, in arithmetic, the proper expressions of type D are proper names ‘0’, ‘1’, ‘2’, etc., and the proper expressions of type  $D \rightarrow S$  are phrases such as ‘is even’, ‘is odd’, etc.

On the other hand, in an abstract language, the proper expressions are just letters, as in elementary logic, whose readings are given by an attendant (often tacit) glossary.

We propose an abstract language, using the following typing convention for the proper expressions.

(1)	small-caps	J, K, L, ...	D
(2)	bold lower-case	<b>f, g, h, ...</b>	$D^* \rightarrow D$
	bold small-caps	<b>F, G, H, ...</b>	$D^* \rightarrow D$
(3)	bold upper-case	<b>P, Q, R, ...</b>	$D^* \rightarrow S$

We also follow elementary logic in writing function-signs using round-parentheses, and writing predicates using square-brackets, which is illustrated in the following glossary.<sup>1</sup>

<b>R</b> [ $\alpha, \beta$ ]	=:	$\alpha$ respects $\beta$
<b>f</b> ( $\alpha$ )	=:	$\alpha$ 's father ; the father of $\alpha$
<b>m</b> ( $\alpha$ )	=:	$\alpha$ 's mother ; the mother of $\alpha$
J	=:	Jay
K	=:	Kay

## 2. Exercises – Part 1

Translate each of the following into the **simple-lambda-calculus**.

Provide

- (1) types
- (2) lambda-translates
- (3) glossary for abbreviations (where necessary)  
{e.g., **R**[ $\alpha, \beta$ ] =:  $\alpha$  respects  $\beta$ }

Note that his language, like the language of intermediate logic, is somewhat impoverished – it does not have case-markers, and it treats common-nouns as one-place predicates, and so it lacks both an autonomous copula and an autonomous indefinite-article. For this reason, the lexicon below and its associated grammar are somewhat unnatural. Keep this in mind in constructing your trees.

<sup>1</sup> We drop brackets for predicates when all its arguments are simple, but we never drop parentheses for function-signs.

	<b>Morpheme</b>	<b>Type</b>	<b>Translation</b>	<b>Glossary</b>
(1)	every			
(2)	some			
(3)	no			
(4)	not			
(5)	and			
(6)	the			
(7)	who			
(8)	[trans] be			
(9)	respects			
(10)	woman is-a-woman			
(11)	man is-a-man			
(12)	the-mother-of 's-mother(def)			
(13)	the-father-of 's-father(def)			
(14)	virtuous is-virtuous			
(15)	happy is-happy			
(16)	next-to is-next-to			
(17)	friend-of is-a-friend-of			
(18)	Jay			
(19)	Kay			
(20)	[mod]			

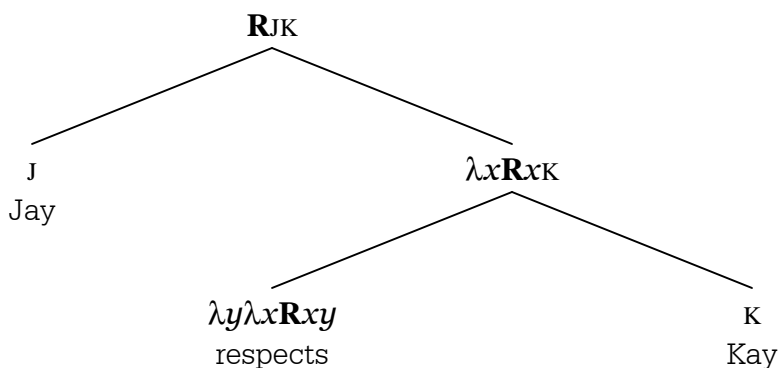
### 3. Exercises – Part 2

For each of the following, using the lexicon above, provide a **semantic-tree**. I.e., write down the most natural tree-structure based on the above lexicon, and assign to each node its translation into the simple-lambda-calculus.

Indicate each composition that does not employ functor-application [lambda-conversion].

1. Not every virtuous woman is happy.
2. The man next-to Kay is Kay's father.
3. No friend of Kay respects Jay.
4. Every woman who respects Jay's father respects Kay's mother.

### 4. Example Analysis (standard-tree format)



$$R[\alpha, \beta] \quad \text{::} \quad \alpha \text{ respects } \beta$$

### 5. Example Analysis (table-tree format)

Jay	respects	Kay
	$\lambda y \lambda x Rxy$	K
J	$\lambda x RxxK$	
<b>RJK</b>		