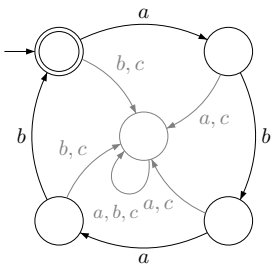
Automata

Ex. 1

Draw a complete deterministic automaton which recognizes all the words on $\{a, b, c\}^*$ wich are composed of an even number of occurrences of the pattern ab.

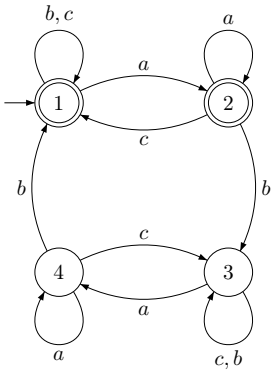
Answer

The language described in the question is the set of words **only** composed of even repetitions of ab. In other words, the language $\{(ab)^k \ / \ k \geqslant 0\}$.
The black automaton given here is deterministic and recognizes this language (to make it complete we have to add the gray part).



A second interpretation of the problem was often found in the copies you turned in: the language you tried to recognize is the set of words such that they comprise an even number of occurrences of the string ab, possibly interwined with other letters with no other constraint.
To recognize such a language, we need to distinguish several states:

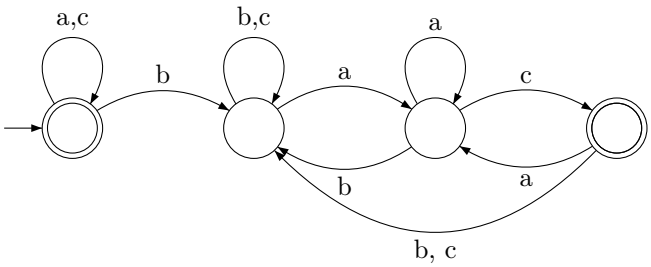
- an initial state which is also a recognition state (no occurrence of the pattern): state 1.
- a state in which an a was seen, but since the pattern ab is incomplete, we can stop there, it is also a recognition state: state 2.
- a state where having seen the pattern ab once (or more generally an odd number of times), we expect another occurrence of ab: state 3.
- a state in which we may be in the middle of the expected ab pattern, in which case a b would lead to recognition, but since we still need a b to accept the word, it's not a recognition state: state 4.



Ex. 2

Draw a complete deterministic automaton which recognizes all the words on $\{a, b, c\}^*$ such that if they include the letter b they must end with the factor ac.

Answer



Formal Grammars

Ex. 3

Propose a (context-free) grammar that engenders the language ab^ncb^na with $n \geqslant 0$.

Answer

We start with a rule to produce the unique pair of as at both ends of the word. Then X is in charge of producing b^ncb^n recursively. The case $n = 0$ corresponds to the sequence of rewritings $S \longrightarrow aXa \longrightarrow aca$.

$$\begin{array}{l} S \to aXa \\ X \to bXb \\ X \to c \end{array}$$

Note that the grammar on the right is not working properly: the two instances of X can produce an arbitrary number of bs, so it is not longer warranted that we have the same number of b. This grammar engenders the language ab^ncb^ma (or ab^*cb^*a).

$$\begin{array}{l} S \to aXcXa \\ X \to bX \\ X \to \varepsilon \end{array}$$

Ex. 4

A λ-term is either a variable, or a lambda abstraction, or a functional application. Considering that the terminal alphabet is $\{\lambda, ., (,), x, y, z, \ldots\}$, give a (context-free) grammar that engenders the language of pure λ-calculus. Give the syntactic trees (according to your grammar) of the λ-terms $\lambda x.\lambda y.(y)x$ and $(\lambda x.\lambda y.y)x$

Answer

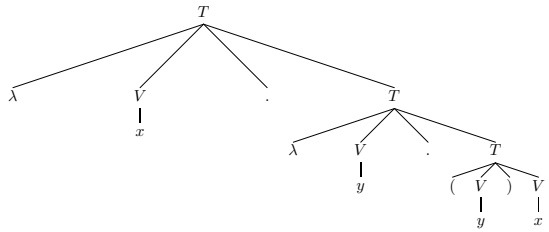
Simplest proposition :

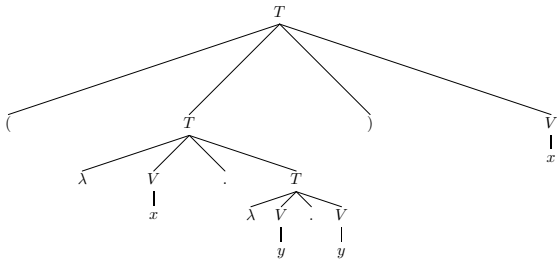
$$\begin{array}{l} T \to \lambda V.T \\ T \to (T)T \\ T \to V \\ V \to x \mid y \mid z \ldots \end{array}$$

Alternatively, to make things more legible, additional "categories" could be used:

$$\begin{array}{l} T \to L \mid F \mid V \\ L \to \lambda V.T \\ F \to (T)T \\ V \to x \mid y \mid z \ldots \end{array}$$

The trees below correspond to the first proposition.





Ex. 5

Let's consider the language L_1 on the vocabulary $V = \{$*l'*, *homme*, *ours*, *qui*, a, *vu*$\}$ that include all finite sentences of the form *l'homme qui a vu l'ours*, *l'homme qui a vu l'homme qui a vu l'ours*, *l'homme qui a vu l'homme qui a vu ... qui a vu l'ours*.

(a) Give a context-free grammar that generates L_1.

Let L_2 be the language engendered by the grammar $\mathcal{G}_2$:

S	→	NP Rel
NP	→	l'homme
	|	l'ours
Rel	→	qui a vu NP Rel
	|	qui a vu l'ours

(b) Propose a grammar $\mathcal{G}_3$ that engenders the language L_3, which is a superset of L_2 in which the symbols *ours* and *homme* are **strictly** commutable.
(c) Describe informally the differences between L_1 and L_2.
(d) Give a context-free grammar of the language $L_2 \setminus L_1$ (L_2 minus L_1).

Answer

(a) There are many options:

$S \to$ *l' homme qui a vu* S
$\quad \to$ *l' homme qui a vu l' ours*

$S \to$ *l' homme qui a vu* S'
$S' \to S \mid$ *l' ours*

$S \to$ *l' homme qui a vu* X *l' ours*
$X \to$ *l' homme qui a vu* X
$\quad \to \varepsilon$

$S \to H$ *l' ours*
$H \to$ *l' homme qui a vu* H
$\quad \to$ *l' homme qui a vu*

The following grammar also works, but with the disadvantage of introducing unnecessary ambiguity:

$S \to H\ O$
$O \to$ *l' ours*
$H \to H\ H$
$\quad \to$ *l' homme qui a vu*

Getting inspiration from the grammar introduced in the next question, one could also propose a version slightly more in line with a linguistic analysis:

S $\to$ NP Rel
NP $\to$ l'homme
Rel $\to$ qui a vu NP Rel
$\quad \mid$ qui a vu l'ours

One important point is to make sure that the grammar does not overgenerate.

(b) A grammar $\mathcal{G}_3$ for L_3.

To make sure that *homme* and *ours* are strictly commutable, we have to ensure that no rule makes a difference between the two words. The grammar $\mathcal{G}_3$ here differs from $\mathcal{G}_2$ only in one place:

S $\to$ NP Rel
NP $\to$ l'homme $\mid$ l'ours
Rel $\to$ qui a vu NP Rel
$\quad \mid$ qui a vu ~~l'ours~~ NP

(c) In L_1, *ours* occurs only once at the end, the NP head is *homme* everywhere else. In L_2, *ours* can be the head of any NP, as well as *homme*, but all well-formed expressions end with *ours*. So the two nouns are almost commutable (interchangeable) but not entirely. To make sure *ours* and *homme* have exactly the same possible contexts, it has to be possible that a well-formed expression ends with *homme*. That's made possible by $\mathcal{G}_3$ (previous question).

(d) Expressions that belong to L_2 without belonging to L_1 are expressions in which *ours* occurs at least once inside an NP (not counting the last occurrence). To achieve this, we need to make sure the derivation cannot be ended until at least one *ours* was introduced. The following grammar is one way (out of many) to achieve this:

$S \to$ *l' homme qui a vu* S
$\quad \to$ *l' ours qui a vu* Y
$Y \to$ *l' homme qui a vu* Y
$\quad \to$ *l' ours qui a vu* Y
$\quad \to$ *l'ours*

Predicate Logic

Ex. 6

Translate as precisely as possible the following sentences intro predicate logic. Please make sure that each formula has as many pairs of parentheses as there are binary operators.

(1) a. No nurse appreciate a surgeon.
b. Everyone is looking for something that not everyone finds.
c. Exactly two persons came.
d. All students should declare their intership if they take one.

Answer

(1a) *No nurse appreciate a surgeon.*
That is a typical case where there might be a scope ambiguity between the two quantified NPs. The formula for the congruent analysis (higher scope for the universal negative) is (2a). *No nurse* can also be represented with a universal quantifier, the corresponding formula would then be (2b). An inverse scope reading would be represented as (2c).
The formula (2d) expresses that for each nurse there is a surgeon that she does not appreciate. It does not seem to be a possible reading of (1a).

(2) a. $\neg\exists x\,(Nx \wedge \exists y\,(Sy \wedge Axy))$
b. $\forall x\,(Nx \rightarrow \neg\exists y\,(Sy \wedge Axy))$
c. $\exists y\,(Sy \wedge \neg\exists x\,(Nx \wedge Axy))$
d. $\forall x\,(Nx \rightarrow \exists y\,(Sy \wedge \neg Axy))$

(1b) *Everyone is looking for something that not everyone finds.*
The first possible reading would be that *there is a unique thing that everyone is looking for and that not everyone finds*, and it can be represented by (3a). Note that this reading is not congruent with the syntactic structure, since it gives wide scope to *something*. A more syntactically congruent reading would have that the things that people look for may be different for each person. Something like (3b). But we need then to make clear how to interpret the last part of the sentence. A straightforward reading would be that it's never the case that everyone finds a thing that some person is looking for: (3c). This reading is probably the easiest to get compositionaly. However, another interpretation could probably be found: an interpretation according to which everyone has a thing that they are looking for, and not everyone finds the thing(s) that they look for. But this is not precise enough: is it the case that some people don't find any of the things they look for (3d)? or is it the case that some people don't find all the things they look for (but only maybe some of them) (3e) ? Depending on the answer to this question, the final formula will the conjunction of (3b) and (3d) or (3e).
The formula (3f) was often proposed. It means that for every person there are things that they search for and that they don't find. I don't think it is a possible interpretation of (1b).

(3) a. $\exists x\,(Tx \wedge \forall y\,(Py \rightarrow Lyx) \wedge \neg\forall y\,(Py \rightarrow Fyx))$
b. $\forall y\,(Py \rightarrow \exists x\,(Tx \wedge Lyx))$
c. $\forall y\,(Py \rightarrow \exists x\,(Tx \wedge Lyx \wedge \neg\forall z\,(Pz \rightarrow Fzx)))$
d. $\neg\forall z\,(Pz \rightarrow \forall x\,((Tx \wedge Lzx) \rightarrow \neg Fzx))$
e. $\neg\forall z\,(Pz \rightarrow \neg\forall x\,((Tx \wedge Lzx) \rightarrow Fzx))$
f. $\forall x\,(Px \rightarrow \exists y\,(Ty \wedge Lxy \wedge \neg Fxy))$

(1c) *Exactly two persons came.*
To refer to exactly two (different) entities we have to use the equal sign in the language.

(4) a. $\exists x\exists y\,(Px \wedge Py \wedge x \neq y \wedge Cx \wedge Cy \wedge \neg\exists z\,(Pz \wedge z \neq x \wedge z \neq y \wedge Cz))$

(1d) *All students should declare their intership if they take one.*
This is a typical donkey sentence. Assuming Dxy for x should declare y, a compositional treatment would lead to a formula like (5a) where a variable is free. There is however a way to account for the truth conditions of the sentence, with the formula (5b) or (5c) which is equivalent.

(5) a. $\forall x\,(Sx \rightarrow (\exists y\,(Iy \wedge Txy) \rightarrow Dxy))$
b. $\forall x\forall y\,(Sx \rightarrow ((Iy \wedge Txy) \rightarrow Dxy))$
c. $\forall x\forall y\,((Sx \wedge Iy \wedge Txy) \rightarrow Dxy))$

Montague Programme (« Grammar engineering »)

Ex. 7

(A) Consider the sentence (6).

(6) Joe is sick.

Let's assume that the semantic representation for this sentence is the formula $\boxed{(S)j}$. We also assume that the semantic contribution of the proper noun *Joe* is the individual constant j. Let's assume in addition that the copula *is* doesn't bring any contribution to the formula. Draw the syntactic tree and give the composition rules so that the fragment containing only (6) is fully defined syntactically and compositionaly.

(B) Let's now assume that *is* has a semantic contribution corresponding to the identity function, of which the only role is to make sure that the λ-term that corresponds to the predicate is passed up in the tree. Give the correct λ-term that should be associated with *is*. Of what type is this term ? Show that the β-reduction(s) yield the expected result.

(C) Let's consider the λ-term (7). Give the type of all its subformulae and of its variables, assuming that x is of type e. S is a predicate constant.

(7) $\lambda Q \lambda x.\ ((S)x \wedge (Q)x)$

(D) Consider the sentence (8).

(8) Joe is sick and depressed.

We suppose first that the syntax of the adjectival phrase *sick and depressed* is "flat", given by the rule $\boxed{AP \rightarrow A' \text{ and } A'}$. As a consequence, we assume a composition rule of the form $\boxed{0 \leftarrow ((2)1)3}$. Which λ-term can we associate with *and* to get the correct representation for (8)?

(E) We assume now that the syntax of coordinated adjectives is not flat, in order to get a binary syntactic tree. The grammar now contains rules of the form $\boxed{AP \rightarrow A'\ CoordP}$ and $\boxed{CoordP \rightarrow and\ A'}$. What difference does it make with the preceding proposal ?

(F) We now associate the λ-term $\boxed{\lambda P.\ (P)j}$ to the proper noun *Joe*. Under this new hypothesis, propose a complete fragment that includes both (6) and (8) (with the options taken in questions B and D).

(G) Consider the sentence (9).

(9) A sick man entered.

We assume that the λ-term to be associated with *sick* is (7). Show that the combination with *man* will yield an expression of the right type.

(H) If now we want the contribution of the adjective *sick* to be $\boxed{\lambda x.\ (S)x}$ we need to introduce en operator Φ such that $\boxed{(\Phi)\lambda x.(S)x =_\beta \lambda Q \lambda x.\ ((S)x \wedge (Q)x)}$. Propose a definition for Φ and give its type.

(I) Enrich the previous fragment so that (9) will get compositionaly the representation (10) (where Sx means $(S)x$, etc.).

(10) $\exists x\ ((Sx \wedge Mx) \wedge Ex)$

. . . . . Answer . . . . .

(A) *Consider the sentence (11).*

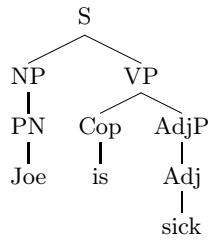
(11) Joe is sick.

Let's assume that the semantic representation for this sentence is the formula $\boxed{(S)j}$*. We also assume that the semantic contribution of the proper noun Joe is the individual constant* j*. Let's assume in addition that the copula is doesn't bring any contribution to the formula. Draw the syntactic tree and give the composition rules so that the fragment containing only (11) is fully defined syntactically and compositionaly.*

To fully define the fragment we need either (A) to provide a complete CFG grammar in which each (syntactic) rule is associated with a semantic rule, or, at least in simple cases like this one, (B) to draw a "decorated" syntactic tree where each node is associated with a lambda-term. In that case the composition rules can be read on the tree and may not be fully made explicit.

(A) A possible grammar could be the one below, which corresponds to the tree on the right. Note: only cases (a) (in which the VP applies to the NP) and (b) (in which the copula contribution is simply ignored) are to be distinguished. The rest, including the lexicon from (c) on, is straightforward.

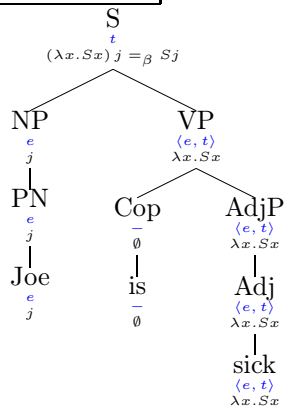
Syntax		Semantics		
S	→ NP VP	$[\![S]\!]$	$= ([\![VP]\!]) [\![NP]\!]$	(a)
VP	→ Cop AdjP	$[\![VP]\!]$	$= [\![AdjP]\!]$	(b)
AdjP	→ Adj	$[\![AdjP]\!]$	$= [\![Adj]\!]$	
NP	→ PN	$[\![NP]\!]$	$= [\![PN]\!]$	
Adj	→ sick	$[\![Adj]\!]$	$= \lambda x.Sx$	(c)
PN	→ Joe	$[\![PN]\!]$	$= j$	
Cop	→ is	$[\![Cop]\!]$	$= \emptyset$	



(A′) An alternative way to present the grammar is to adopt a notation closer to the one used by computer scientists (0 represents the semantics value of the left-hand side of the rule (the root of the local tree), 1, 2, 3, *etc.* represent the semantic values of the right-hand side symbols of the rule (the daughters of the tree)).

S	→	NP VP		Adj	→	sick
0	←	(2) 1		0	←	$\lambda x.Sx$
VP	→	Cop AdjP		PN	→	Joe
0	←	2		0	←	j
AdjP	→	Adj		Cop	→	is
0	←	1		0	←	$\emptyset$
NP	→	PN				
0	←	1				

(B) The "decorated" tree on the right contains essentially the same information as the grammar(s) above. We've added the types of λ-terms which are useful to understand in which order the terms are meant to combine.



(B) *Let's now assume that* is *has a semantic contribution corresponding to the identity function, of which the only role is to make sure that the* λ*-term that corresponds to the predicate is passed up in the tree. Give the correct* λ*-term that should be associated with* is*. Of what type is this term ? Show that the* β*-reduction(s) yield the expected result.*

The λ-term associated with *is* is (a version of) the identity function $\boxed{\lambda P.P}$, where P is a type $\langle e,t\rangle$ variable. Consequently, the type of the λ-term is $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$.

Computation:

$$\begin{array}{ll} & ([\![\text{is}]\!])\,[\![\text{sick}]\!])\,[\![\text{Joe}]\!] \\ = & ((\lambda P.P)\,\lambda x.Sx)\,j \\ =_\beta & (\lambda x.Sx)\,j \\ =_\beta & Sj \end{array}$$

Note: it was also possible to have a computation working with the term $\boxed{\lambda P.\lambda z.(P)z}$, which have the same type $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$. The computation has one more β-reduction but yields the right result. However, this term is not the most general way to encode the identity function and it doesn't bring much in this case.

Computation:

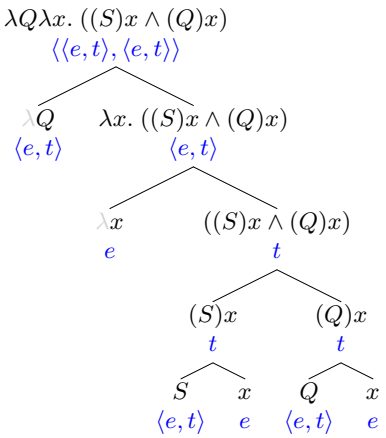
$$\begin{array}{ll} & ((\llbracket\text{is}\rrbracket)\,\llbracket\text{sick}\rrbracket)\,\llbracket\text{Joe}\rrbracket \\ = & ((\lambda P.\lambda z.(P)z)\,\lambda x.Sx)\,j \\ =_\beta & (\lambda z.(\lambda x.Sx)z)\,j \\ =_\beta & (\lambda z.Sz)\,j \\ =_\beta & Sj \end{array}$$

(C) *Let's consider the λ-term (12). Give the type of all its subformulae and of its variables, assuming that x is of type e. S is a predicate constant.*

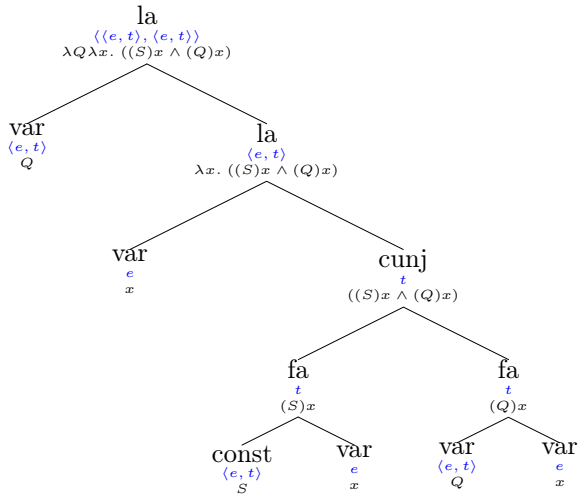
(12) $\lambda Q\lambda x.\,((S)x \wedge (Q)x)$

Since we know that x is of type e, Q (which is functionally applied to x) has to have a type of the form $\langle e, X\rangle$. In addition, since conjunctions can only connect type t expressions, we know that $(Q)x$ is of type t. Therefore Q has to be of type $\langle e,t\rangle$. This fixes the types of all the other subformulae.

The simplest way to make sure we deal with all the term's subformulae is to represent it by a syntactic tree, in which (almost) every node is a subformulae (in fact the left node of lambda-abstractions shows the variable introduce by the lambda, it's not strictly a subformula of the formula).



More inline with what was done in class, this alternative representation could also be proposed, in which every node indicates the nature of the syntactic rule (la means lambda-abstraction, fa means functional application).

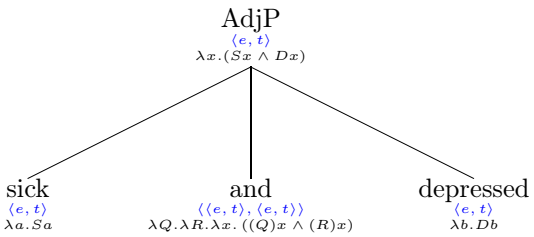


(D) Consider the sentence (13).

(13) Joe is sick and depressed.

We suppose first that the syntax of the adjectival phrase sick and depressed is "flat", given by the rule $\boxed{AP \to A'\ \text{and}\ A'}$. *As a consequence, we assume a composition rule of the form* $\boxed{0 \leftarrow ((2)1)3}$. *Which λ-term can we associate with* and *to get the correct representation for (13)?*

The lambda term for *and* has to be able to apply in succession to two predicates (type $\langle e,t\rangle$), and make sure those two predicates apply to the same variable to form a new predicate. This is why we can propose the term $\boxed{\lambda Q.\lambda R.\lambda x.\,((Q)x \wedge (R)x)}$. The subtree on the right illustrates the computation.



Computation:

$$\begin{array}{ll} & ((2)1)3 \\ = & ((\lambda Q.\lambda R.\lambda x.\,((Q)x \wedge (R)x))\,\lambda a.Sa)\,\lambda b.Db \\ =_\beta & (\lambda R.\lambda x.\,((\lambda a.Sa)x \wedge (R)x))\,\lambda b.Db \\ =_\beta & (\lambda R.\lambda x.\,(Sx \wedge (R)x))\,\lambda b.Db \\ =_\beta & \lambda x.\,(Sx \wedge (\lambda b.Db)x) \\ =_\beta & \lambda x.\,(Sx \wedge Dx) \end{array}$$

(E) *We assume now that the syntax of coordinated adjectives is not flat, in order to get a binary syntactic tree. The grammar now contains rules of the form* $AP \rightarrow A'\ CoordP$ *and* $CoordP \rightarrow and\ A'$. *What difference does it make with the preceding proposal ?*

The type and form of the representation of *and* is unchanged, the only thing changing is the set of semantic rules associated with syntax. Instead of the description on the left (corresponding to question D), we have the one on the right.

AP	$\rightarrow$	A′ *and* A′
0	$\leftarrow$	((2) 1) 3

AP	$\rightarrow$	A′ CoordP
0	$\leftarrow$	(2) 1
CoordP	$\rightarrow$	*and* A′
0	$\leftarrow$	(1) 2

(F) *We now associate the* λ*-term* $\lambda P.\,(P)j$ *to the proper noun Joe. Under this new hypothesis, propose a complete fragment that includes both (11) and (13) (with the options taken in questions B and D).*

The complete fragment, using the generalised quantifier approach, is described by the following grammar. According to this grammar the complete computation can be summarized by the "formula":

(Joe) (is) ((and) sick) depressed

S	$\rightarrow$	NP VP
0	$\leftarrow$	(1) 2
VP	$\rightarrow$	Cop AdjP
0	$\leftarrow$	(1) 2
AdjP	$\rightarrow$	Adj
0	$\leftarrow$	1
AdjP	$\rightarrow$	AdjP Cunj AdjP
0	$\leftarrow$	((2) 1) 3
NP	$\rightarrow$	PN
0	$\leftarrow$	1

Adj	$\rightarrow$	sick
0	$\leftarrow$	$\lambda a.Sa$
Adj	$\rightarrow$	depressed
0	$\leftarrow$	$\lambda b.Db$
PN	$\rightarrow$	Joe
0	$\leftarrow$	$\lambda P.Pj$
Cop	$\rightarrow$	is
0	$\leftarrow$	$\lambda P.P$
Cunj	$\rightarrow$	and
0	$\leftarrow$	$\lambda Q.\lambda R.\lambda x.((Q)x \wedge (R)x)$

(G) *Consider the sentence (14).*

(14) *A sick man entered.*

We assume that the λ*-term to be associated with* sick *is (12). Show that the combination with* man *will yield an expression of the right type.*

Computation:
$$\begin{aligned} & (\llbracket \text{sick} \rrbracket)\ \llbracket \text{man} \rrbracket \\ = \quad & (\lambda Q.\lambda x.((S)x \wedge (Q)x))\ \lambda a.Ma \\ =_\beta \quad & \lambda x.((S)x \wedge (\lambda a.Ma)x) \\ =_\beta \quad & \lambda x.((S)x \wedge (M)x) \end{aligned}$$

The outcome is a (complex) predicate, of type $\langle e, t\rangle$, which is what is expected (same type as *man*).

(H) *If now we want the contribution of the adjective* sick *to be* $\lambda x.\,(S)x$ *we need to introduce en operator* Φ *such that* $(\Phi)\lambda x.(S)x =_\beta \lambda Q\lambda x.\,((S)x \wedge (Q)x)$. *Propose a definition for* Φ *and give its type.*

$\Phi = \lambda R.\lambda Q.\lambda x.(Rx \wedge Qx)$, and its type is $\langle\langle e, t\rangle, \langle\langle e, t\rangle, \langle e, t\rangle\rangle\rangle$

(I) *Enrich the previous fragment so that (14) will get compositionaly the representation (15) (where* Sx *means* $(S)x$*, etc.).*

(15) $\exists x\ ((Sx \wedge Mx) \wedge Ex)$

What is needed to have a complete fragment, given what we already have, is to express how an adjective like *sick*, when operating as a modifier, has to be type-raised to combine with its head. The rest of the grammar is already known, assuming that the indefinite determiner has the usual quantificational representation. The tree below summarizes the proposition.

S	→	NP VP
0	←	(1) 2
VP	→	Cop AdjP
0	←	(1) 2
VP	→	V
0	←	1
AdjP	→	Adj
0	←	1
AdjP	→	AdjP Cunj AdjP
0	←	((2) 1) 3
NP	→	PN
0	←	1
NP	→	Det N′
0	←	(1) 2
N′	→	AdjP N′
0	←	(($\lambda R.\lambda Q.\lambda x.(Rx \wedge Qx)$) 1) 2
N′	→	N
0	←	1

Adj	→	sick
0	←	$\lambda a.Sa$
N	→	man
0	←	$\lambda c.Mc$
Adj	→	depressed
0	←	$\lambda b.Db$
V	→	entered
0	←	$\lambda d.Ed$
PN	→	Joe
0	←	$\lambda P.Pj$
Cop	→	is
0	←	$\lambda P.P$
Cunj	→	and
0	←	$\lambda Q.\lambda R.\lambda x.((Q)x \wedge (R)x)$
Det	→	a
0	←	$\lambda P.\lambda Q.(Px \wedge Qx)$