Consider the following abstract syntax together with its associated Montague-like semantics:

\[
\begin{align*}
\text{ALICE} & : \text{np} & [\text{ALICE}] &= \lambda P. P \text{ alice} \\
\text{SOMEONE} & : \text{np} & [\text{SOMEONE}] &= \lambda P. \exists x. P \ x \\
\text{LEFT} & : \text{np} \rightarrow s & [\text{LEFT}] &= \lambda S.S (\lambda x. \text{left} \ x) \\
\text{THINK} & : \text{cc} \rightarrow \text{np} \rightarrow s & [\text{THINK}] &= \lambda C.\lambda s.s (\lambda x. \text{think} \ x \ (C \ (\lambda t.t))) \\
\text{THAT} & : (\text{np} \rightarrow s) \rightarrow (\text{np} \rightarrow \text{cc}) & [\text{THAT}] &= \lambda P.\lambda s.\lambda r.s (\lambda x. P (\lambda q.r \ (q \ x)))
\end{align*}
\]

where \([\ ]\) denotes both the interpretation of the syntactic types into the semantic types and the interpretation of the abstract terms into the semantic terms. Moreover:

\[
\begin{align*}
\text{alice} & : e & \text{human} & : e \rightarrow t \\
\text{left} & : e \rightarrow t & \text{think} & : e \rightarrow t \rightarrow t
\end{align*}
\]

1. Check that the following term \(t\) is well-typed. What is its type?

\[
t = \text{THINK} \ (\text{THAT LEFT SOMEONE}) \ \text{ALICE}
\]

Let \(\Pi_1\) =
\[
\text{THAT} : (\text{np} \rightarrow s) \rightarrow \text{np} \rightarrow \text{cc} \vdash \text{THAT} : (\text{np} \rightarrow s) \rightarrow \text{np} \rightarrow \text{cc} \quad \text{LEFT} : \text{np} \rightarrow s \vdash \text{LEFT} : \text{np} \rightarrow s \quad \text{THAT LEFT} : \text{np} \rightarrow \text{cc}
\]

\[
\text{THAT} : (\text{np} \rightarrow s) \rightarrow \text{np} \rightarrow \text{cc}, \text{LEFT} : \text{np} \rightarrow s \vdash \text{THAT LEFT} : \text{np} \rightarrow \text{cc} \quad \text{SOMEONE} : \text{np} \vdash \text{SOMEONE} : \text{np}
\]

\(\Pi_2\) proves that \(\text{THAT LEFT SOMEONE}\) is correctly typed of type \(\text{cc}\).

Let \(\Pi_3\) =
\[
\text{THINK} : \text{cc} \rightarrow \text{np} \rightarrow s \vdash \text{THINK} : \text{cc} \rightarrow \text{np} \rightarrow s = \text{THAT} : (\text{np} \rightarrow s) \rightarrow \text{np} \rightarrow \text{cc}, \text{LEFT} : \text{np} \rightarrow s, \text{SOMEONE} : \text{np} \vdash \text{THAT LEFT SOMEONE} : \text{cc}
\]

\[
\text{THINK} : \text{cc} \rightarrow \text{np} \rightarrow s, \text{THAT} : (\text{np} \rightarrow s) \rightarrow \text{np} \rightarrow \text{cc}, \text{LEFT} : \text{np} \rightarrow s, \text{SOMEONE} : \text{np} \vdash \text{THINK} \ (\text{THAT LEFT SOMEONE}) : \text{np} \rightarrow s
\]

\(\Pi_3\) proves that \(\text{THINK} \ (\text{THAT LEFT SOMEONE}) \ \text{ALICE}\) is well typed of type \(s\)

\[
\text{THINK} : \text{cc} \rightarrow \text{np} \rightarrow s, \text{THAT} : (\text{np} \rightarrow s) \rightarrow \text{np} \rightarrow \text{cc}, \text{LEFT} : \text{np} \rightarrow s, \text{SOMEONE} : \text{np}, \text{ALICE} : \text{np} \vdash \text{THINK} \ (\text{THAT LEFT SOMEONE}) \ \text{ALICE} : s
\]

2. The syntactic category assigned to the complement clause \(\text{that someone left}\) is \(\text{cc}\).
(a) Express the type of \( \text{THINK} \) in terms of \([cc], [np], \) and \([s] \).

According to the homomorphism requirement between syntactic types and semantic types the type of \( \text{THINK} \) is:
\[
[cc \rightarrow np \rightarrow s] = [cc] \rightarrow [np] \rightarrow [s]
\]

(b) According to the present hypothesis (given in the above mentioned lexicon), what are the semantic categories (semantic types) corresponding to \( np, s, \) and \( cc \)? Justify your answers for each of these types.

- Because \( \text{alice} \) is of type \( e \) and \( \text{ALICE} \) is of type \( np, \) \( [np] = (e \rightarrow \alpha) \rightarrow \alpha. \) Moreover, this is also the type of \( [\text{SOMEONE}] \) hence \( \alpha \) is the type of \( \exists x.P \) which is \( t. \) So \( [np] = (e \rightarrow t) \rightarrow t. \)
- Then because the type of \( [\text{LEFT}] = \lambda S. S(\text{LEFT}) \) is \( [np] \) \rightarrow \([s]\) and we know \( S \) in this term is of type \( (e \rightarrow t) \rightarrow t, \) we have that \( S(\text{LEFT}) : t = [s]. \)
- Looking at \( [\text{THINK}] \), \( C \) is of type \( [cc] = (\beta \rightarrow \beta) \rightarrow t \) (because \( C(\lambda t.t) \) is the second argument of \( \text{think, hence of type t, and because the argument of C is the identity}. \) \( [cc] \) is also the type of \( \lambda r.s (\lambda x.P (\lambda q.r (q x))) \) in \( [\text{THAT}] \). But \( P \) here is of type \( [np] \) \rightarrow \([s]\), so \( \lambda q.r (q x) \) is of type \( [np] = (e \rightarrow t) \rightarrow t \) so \( q \) is of type \( e \rightarrow t \) and \( r (q x) \) of type \( t. \) So \( r \) is of type \( t \rightarrow t \) and \( \lambda r.s (\lambda x.P (\lambda q.r (q x))) \) is of type \( (t \rightarrow t) \rightarrow t = [cc]. \)

3. Compute the semantic representation \( m \) of the sentence \( \text{Alice thinks that someone left}, \) whose abstract syntax is given by the term \( t. \)

\[
[t] = [\text{THINK}(\text{THAT} \ \text{LEFT} \ \text{SOMEONE}) \ \text{Alice}] \\
= [\text{THINK} \ (([\text{THAT}] \ [\text{LEFT}] \ [\text{SOMEONE}]) \ [\text{Alice}])]
\]

\[
[\text{THAT}] \ [\text{LEFT}] \ [\text{SOMEONE}] = (\lambda P. \lambda s. \lambda r.s (\lambda x.P (\lambda q.r (q x)))) \ [\text{LEFT}] \ [\text{SOMEONE}]
= \lambda r.([\text{SOMEONE}](\lambda x.([\text{LEFT}](\lambda q.r (q x))))
= \lambda r.([\text{SOMEONE}](\lambda x.\lambda S.S(\text{LEFT})(\lambda q.r (q x))))
= \lambda r.([\text{SOMEONE}](\lambda x.\lambda q.r (q x) (\text{LEFT}))
= \lambda r.([\text{SOMEONE}](\lambda x.r (\text{LEFT} x)))
= \lambda r.\exists x.\lambda x.r (\text{LEFT} x) x
= \lambda r.\exists x.r (\text{LEFT} x)
\]

\[
[\text{THINK} \ (([\text{THAT}] \ [\text{LEFT}] \ [\text{SOMEONE}])]) = (\lambda C. \lambda s.\lambda x.\text{think} x (C (\lambda t.t))) (\lambda x.\exists x.r (\text{LEFT} x))
= \lambda s.\lambda x.\text{think} x ((\lambda r.\exists x.r (\text{LEFT} x)) (\lambda t.t))
= \lambda s.\lambda x.\text{think} x (\exists x.\lambda t.t (\text{LEFT} x))
= \lambda s.\lambda x.\text{think} x (\exists x.\text{LEFT} x)
\]

---

1Remember that \( \lambda x.\text{LEFT} x \) is \( \eta \)-equivalent to \( \text{LEFT}. \)
Examen Master Sciences de la Cognition et Applications

\[
\text{[THINK]}(\text{[THAT]} \text{[LEFT]} \text{[SOMEONE]})) \text{[ALICE]} = (\lambda s. (\lambda x. \text{think } x (\exists x. \text{left } x))) (\lambda P. P \text{ alice}) \\
= (\lambda P. P \text{ alice}) (\lambda x. \text{think } x (\exists x. \text{left } x)) \\
= (\lambda x. \text{think } x (\exists x. \text{left } x)) \text{ alice} \\
= \text{think alice} (\exists x. \text{left } x)
\]

(a) According to \(m\), does Alice necessarily have an idea of who left?

No. She could have some clue that someone left without knowing precisely who is that person.

(b) Given to expressions \(s_1\) and \(s_2\), when do we say that \(s_1\) intuitively entails \(s_2\)?

\(s_1\) intuitively entails \(s_2\) if and only if for all models \(M\) in \(T\) \([S_1]_M \leq [S_2]_M\).

4. Give a semantic recipe to THINK such that the semantic reading associated to Alice thinks that someone left is now There’s a precise person such that Alice thinks this person left.

\[
[\text{THINK}'] = \lambda C. \lambda s. (\lambda x. C(\lambda t. \text{think } x t))
\]

is such that

\[
[\text{THINK}'](\text{[THAT]} \text{[LEFT]} \text{[SOMEONE]})) \text{[ALICE]} = \exists x. \text{think alice } (\text{left } x)
\]

because

\[
[\text{THINK}'](\text{[THAT]} \text{[LEFT]} \text{[SOMEONE]})) = (\lambda C. \lambda s. (\lambda x. C(\lambda t. \text{think } x t)))(\lambda r. \exists x. r (\text{left } x)) \\
= \lambda s. (\lambda x. (\lambda r. \exists x. r (\text{left } x))) (\lambda t. \text{think } x t)) \\
= \lambda s. \lambda y. (\lambda r. \exists x. r (\text{left } x)) (\lambda t. \text{think } y t)) \\
= \lambda s. \lambda y. \exists x. (\lambda t. \text{think } y t) (\text{left } x)) \\
= \lambda s. \lambda y. \exists x. \text{think } y (\text{left } x))
\]

Hence

\[
[\text{THINK}](\text{[THAT]} \text{[LEFT]} \text{[SOMEONE]})) \text{[ALICE]} = (\lambda s. (\lambda y. \exists x. \text{think } y (\text{left } x))) (\lambda P. P \text{ alice}) \\
= (\lambda P. P \text{ alice}) (\lambda y. \exists x. \text{think } y (\text{left } x))) \\
= (\lambda y. \exists x. \text{think } y (\text{left } x)) \text{ alice} \\
= \exists x. \text{think alice } (\text{left } x))
\]

5. Check that \[\text{THAT}\] is well typed.

It amounts to check that \([\text{THAT}]\) is of type

\[
[(np \rightarrow s) \rightarrow (np \rightarrow cc)] \\
= [(np \rightarrow s) \rightarrow (np \rightarrow cc)] \\
= (\|np\| \rightarrow \|s\|) \rightarrow \|np\| \rightarrow \|cc\] \\
= ((e \rightarrow t) \rightarrow t) \rightarrow ((e \rightarrow t) \rightarrow t) \rightarrow (t \rightarrow t) \rightarrow t
\]
Let $\Pi =$

\[
\begin{array}{c}
q : e \to t \vdash q : e \to t \\
x : e \vdash x : e
\end{array}
\]

\[
\begin{array}{c}
r : t \to t \vdash r : t \to t \\
q : e \to t, x : e \vdash q x : t
\end{array}
\]

\[
\begin{array}{c}
r : t \to t, q : e \to t, x : e \vdash r (q x) : t \\
r : t \to t, x : e \vdash \lambda q. r (q x) : (e \to t) \to t
\end{array}
\]

Then the following derivation proves that $\|\text{THAT}\|$ is well typed.

\[
\begin{array}{c}
P : [np] \to [s] \vdash P : [np] \to [s] \\
r : t \to t, x : e \vdash \lambda q. r (q x) : [np]
\end{array}
\]

\[
\begin{array}{c}
P : [np] \to [s] , r : t \to t, x : e \vdash P(\lambda q. r (q x)) : t \\
\vdash \lambda r.s. \lambda x.P(\lambda q. r (q x)) : (t \to t) \to t
\end{array}
\]

\[
\begin{array}{c}
P : [np] \to [s], r : t \to t \vdash s(\lambda x.P(\lambda q. r (q x))) : [np] \to (t \to t) \to t
\end{array}
\]

\[
\begin{array}{c}
s : [np], P : [np] \to [s] \vdash s(\lambda x.P(\lambda q. r (q x))) : (t \to t) \to t
\end{array}
\]

\[
\begin{array}{c}
P : [np] \to [s] \vdash \lambda P. s r.s(\lambda x.P(\lambda q. r (q x))) : [np] \to (t \to t) \to t
\end{array}
\]