1. The question is somewhat open-ended, but one solution is as follows. We consider the following three attributes:

**Person:** values 1,2,3, ranged over by \( x \)

**Number:** values s,p, ranged over by \( y \)

**Gender:** values m,f,n, ranged over by \( z \)

The following parameterized rules suffice:

\[
\begin{align*}
S & \rightarrow \text{SPro}[x,y,z] \; \text{Vstem} \; \text{Vsuf}[x,y] \; \text{ReflPro}[x,y,z] \\
\text{Vstem} & \rightarrow \text{love} \mid \text{prepare} \mid \text{congratulate} \\
\text{SPro}[1,s,z] & \rightarrow \text{I} \\
\text{SPro}[1,p,z] & \rightarrow \text{We} \\
\text{SPro}[2,y,z] & \rightarrow \text{You} \\
\text{SPro}[3,s,m] & \rightarrow \text{He} \\
\text{SPro}[3,s,f] & \rightarrow \text{She} \\
\text{SPro}[3,s,n] & \rightarrow \text{It} \\
\text{SPro}[3,p,z] & \rightarrow \text{They} \\
\text{Vsuf}[3,s] & \rightarrow -s \\
\text{Vsuf}[1,s] & \rightarrow \varepsilon, \text{ etc.} \\
\text{ReflPro}[1,s,z] & \rightarrow \text{myself} \\
\text{ReflPro}[2,s,z] & \rightarrow \text{yourself} \\
\text{ReflPro}[3,s,m] & \rightarrow \text{himself} \\
\text{ReflPro}[3,s,f] & \rightarrow \text{herself} \\
\text{ReflPro}[3,s,n] & \rightarrow \text{itself} \\
\text{ReflPro}[1,p,z] & \rightarrow \text{ourselves} \\
\text{ReflPro}[2,p,z] & \rightarrow \text{yourselves} \\
\text{ReflPro}[3,p,z] & \rightarrow \text{themselves}
\end{align*}
\]

2. We want a single constant, Jumbo, and predicates with arities as follows:

- \( \text{elephant}/1 \)
- \( \text{mammal}/1 \)
- \( \text{owns}/2 \)
- \( \text{song}/1 \)
- \( \text{sings_to}/3 \)
- \( \text{danced}/1 \)

We also have a binary equality predicate as a standard part of our FOPL machinery.

The given sentences may be translated into FOPL as follows:

- \text{Jumbo is an elephant}: \text{elephant}(\text{Jumbo})

- \text{An elephant is a mammal}: this could arguably have either of the following meanings.

\[
\forall x. \text{elephant}(x) \Rightarrow \text{mammal}(x)
\]

\[
\exists x. \text{elephant}(x) \land \text{mammal}(x)
\]

---

\(^1\)Thanks to Michael Herrmann for providing the solution for Problem 3.
• *Every elephant has an owner:*

\[ \forall x. \text{elephant}(x) \Rightarrow \exists y. \text{owns}(y, x) \]

or possibly

\[ \exists y. \forall x. \text{elephant}(x) \Rightarrow \text{owns}(y, x) \]

• *Everyone who owns an elephant sings it a song:*

\[ \forall x. \forall y. \text{owns}(x, y) \land \text{elephant}(y) \Rightarrow \exists z. (\text{song}(z) \land \text{sings}_\text{to}(x, z, y)) \]

or possibly

\[ \forall x. \forall y. \exists z. (\text{owns}(x, y) \land \text{elephant}(y)) \Rightarrow (\text{song}(z) \land \text{sings}_\text{to}(x, z, y)) \]

• *Only one elephant danced.*

\[ \exists x. \text{elephant}(x) \land \text{danced}(x) \land (\forall y. (\text{elephant}(y) \land \text{danced}(y)) \Rightarrow y = x) \]

• *Every elephant did not dance:* there is a scoping ambiguity here (cf. the expression *All is not lost.*) Either

\[ \neg (\forall x. \text{elephant}(x) \Rightarrow \text{danced}(x)) \]

or

\[ \forall x. \text{elephant}(x) \Rightarrow \neg \text{danced}(x) \]

3. The rules that take as an argument variables called \( P \) or \( Q \) are type-raising rules. Type raising refers to the case in which functions themselves are fed as an argument to a higher-order function.

The semantics of the given phrases are as follows. For the first two examples, we show the \( \beta \)-reductions steps; for the others, we show only the result after \( \beta \)-reduction.

• John runs:

\[ (\lambda P. P(\text{John}))(\lambda x. \text{run}(x)) \rightarrow_\beta (\lambda x. \text{run}(x))(\text{John}) \rightarrow_\beta \text{run}(\text{John}) \]

• likes ice-cream:

\[ \lambda x. (\lambda P. P(\text{Ice-cream}))(\lambda x. \lambda y. \text{like}(x,y))(x) \]

\[ \rightarrow_\beta \lambda x. (\lambda P. P(\text{Ice-cream}))(\lambda y. \text{like}(x,y)) \]

\[ \rightarrow_\beta \lambda x. (\lambda y. \text{like}(x,y))(\text{Ice-cream}) \rightarrow_\beta \lambda x. \text{like}(x,\text{Ice-cream}) \]

• John likes ice-cream: \( \text{like}(\text{John},\text{Ice-cream}) \)

• an ice-cream: \( \lambda P. \exists x. \text{ice-cream}(x) \land P(x) \)

• likes an ice-cream: \( \lambda y. \exists x. \text{ice-cream}(x) \land \text{like}(y,x) \)

• John likes an ice-cream: \( \exists x. \text{ice-cream}(x) \land \text{like}(\text{John},x) \)

• every cat: \( \lambda P. \forall x. \text{cat}(x) \Rightarrow P(x) \)

• every cat likes ice-cream: \( \forall x. \text{cat}(x) \Rightarrow \exists y. \text{ice-cream}(y) \land \text{like}(x,y) \)

• every cat likes an ice-cream: \( \forall x. \text{cat}(x) \Rightarrow \exists y. \text{ice-cream}(y) \land \text{like}(x,y) \)
a) phrase: John runs
parse tree: (S(NP John)(VP(IV runs)))
semantics: NP.Sem(VP.Sem) → (λx.P(John))(λx.run(x)) (note that there is a different arrow now)
→β (λx.run(x))(John) → β run(John)

b) phrase: likes ice-cream
parse tree: VP((TV likes)(NP ice-cream))
semantics: The semantics of VP tells us how to proceed:

λx.NP.Sem(TV.Sem(x))

We can replace the x in TV.Sem by a z to avoid a clash of variable and insert the semantics
(for simplicity this is not always done in the other examples below):

→λx.(λx.P(Ice-cream))((λy.like(z,y))(x))

Startig from left λz is applied :

→β λx.(λx.P(Ice-cream))(λy.like(x,y))

And then, just as in the previous example

→β λx.(like(x.Ice-cream)) →β λx.(like(x, Ice-cream)
c) phrase: John likes ice-cream

parse tree: \((S(NP \text{John})(VP(TV \text{likes})(NP \text{ice-cream})))\)

semantics: \(\text{NP.Sem}(\text{VP.Sem}) \rightarrow (\lambda P(\text{John}))(\lambda x.\lambda y.\text{like}(x,y))(x)\)

\(\lambda P(\text{Ice-cream})(\lambda x.\lambda y.\text{like}(x,y))(\text{John})\)

\(\lambda y.\text{like}(\text{John},y)(\text{Ice-cream})\)

\(\lambda \text{like}(\text{John},\text{Ice-cream})\)

d) phrase: an ice-cream

parse tree: \((NP(\text{Det an})(\text{N ice-cream}))\)

semantics: \(\text{Det.Sem}(\text{N.Sem}) \rightarrow (\lambda Q.\lambda P.\exists x.\text{Q(x)} \land P(x))(\lambda x.\text{ice-cream}(x))\)

\(\lambda P.\exists x.(\lambda x.\text{ice-cream}(x) \land P(x))\)

\(\lambda P.\exists x.\text{ice-cream}(x) \land P(x)\)

e) phrase: likes an ice-cream

parse tree: \((VP(\text{TV likes})(NP(\text{Det an})(\text{N ice-cream})))\)

semantics: \(\lambda x.\text{NP.Sem}(\text{TV.Sem}(x)) \rightarrow \lambda x.(\text{Det.Sem}(\text{N.Sem}))(\lambda x.\lambda y.\text{like}(x,y))(x)\)

\(\lambda x.((\lambda Q.\lambda P.\exists s.\text{Q(s)} \land P(s))(\lambda w.\text{ice-cream}(w)))\)

\(\lambda x.((\lambda P.\exists s.\text{ice-cream}(s) \land P(s))(\lambda y.\text{like}(x,y))(s))\)

\(\lambda x.((\exists s.\text{ice-cream}(s) \land (\lambda y.\text{like}(x,y))(s))\)

\(\lambda x.((\exists s.\text{冰-cream}(s) \land \text{like}(x,s))\)

[note again that disambiguation of the variables is important]
f) phrase: John likes an ice-cream

parse tree: \( S(NP \text{John})(VP(TV \text{likes})(NP(\text{Det an})(N \text{ice-cream})))) \)

semantics: \( \text{NP.Sem} + \text{VP.Sem} \) 

\[
\begin{align*}
\lambda x.\lambda y.\text{like}(x,y)(z) &\quad \lambda x.\text{like}(x,\text{ice-cream})(y) \\
\lambda x.\text{like}(x,\text{ice-cream})(z) &\quad \lambda x.\text{like}(\text{ice-cream},x)(y)
\end{align*}
\]


g) phrase: every cat

parse tree: \( (NP(\text{Det every})(N \text{cat})) \)

semantics: \( \text{Det.Sem} + \text{N.Sem} \) 

\[
\begin{align*}
\lambda x.\text{like}(x,\text{ice-cream})(y) &\quad \lambda x.\text{like}(x,\text{ice-cream})(z) \\
\lambda x.\text{like}(\text{ice-cream},x)(y) &\quad \lambda x.\text{like}(\text{ice-cream},x)(z)
\end{align*}
\]


h) phrase: every cat likes ice-cream

parse tree: \( (S(NP(\text{Det every})(N \text{cat}))(VP(TV \text{likes})(NP(\text{Det an})(N \text{ice-cream})))) \)

semantics: from the first level of the semantic tree and using (g) and (h) we have

\[
\begin{align*}
\lambda x.\text{like}(x,\text{ice-cream})(y) &\quad \lambda x.\text{like}(x,\text{ice-cream})(z) \\
\lambda x.\text{like}(\text{ice-cream},x)(y) &\quad \lambda x.\text{like}(\text{ice-cream},x)(z)
\end{align*}
\]

Note that this is not exactly the given solution, but equivalent.
i) phrase: every cat likes an ice-cream

parse tree: (S(NP(Det every)(N cat))(VP(TV likes)(NP(Det an)(N ice-cream))))

semantics: NP.Sem(VP.Sem) → (Det.Sem(N.Sem))(λz.(Det.Sem(N.Sem)))(λx.λy.like(x,y)(z))

... seems quite a lot of work, but most is just cut&paste&edit, you don’t need to do every step separately and large parts repeat across the examples which I did not use (except in h). And, sorry for any missing or superfluous brackets, for typos etc.