# SPNLP Assignment 1: Sample Answers

Alex Lascarides & Ewan Klein

March 7, 2008

## Exercise 1

| | Subformula | Free variables |
|---|---|---|
| 1 | $(((Q\,x) \vee \exists x.\forall y.((P\,z\,x) \wedge (Q\,a))) \vee \forall z.(R\,x\,z\,x))$ | $\{x, z\}$ |
| 2 | $((Q\,x) \vee \exists x.\forall y.((P\,z\,x) \wedge (Q\,a)))$ | $\{x, z\}$ |
| 3 | $(Q\,x)$ | $\{x\}$ |
| 4 | $\exists x.\forall y.((P\,z\,x) \wedge (Q\,a))$ | $\{z\}$ |
| 5 | $\forall y.((P\,z\,x) \wedge (Q\,a))$ | $\{z, x\}$ |
| 6 | $((P\,z\,x) \wedge (Q\,a))$ | $\{z, x\}$ |
| 7 | $(P\,z\,x)$ | $\{z, x\}$ |
| 8 | $(Q\,a))$ | $\{\}$ |
| 9 | $\forall z.(R\,x\,z\,x)$ | $\{x\}$ |
| 10 | $(R\,x\,z\,x)$ | $\{x, z\}$ |

Points to note:

- Every formula is a subformula of itself.
- If $\mathcal{Q}$ is a quantifier, then $\phi$ is a subformula of $\mathcal{Q}x.\phi$; however, $\mathcal{Q}x$ is not a subformula of $\mathcal{Q}x.\phi$ so the question of $x$ being free in $\mathcal{Q}x.\phi$ does not arise.

## Exercise 2

We had the following formulas (for better readability, expressed in relational rather than functional form):

(1) $\forall x.\exists y.neighbour(x, y)$

(2) $\forall x.\neg neighbour(x, x)$

(3) $\forall x.\forall y.\forall z.(neighbour(x, y) \wedge neighbour(y, z) \rightarrow neighbour(x, z)$

Although it can be helpful to use the NLTK style models for exploring truth conditions, we can also look at the argument more abstractly.

Let's assume that there are exactly two individuals in the model, named $a$ and $b$. From (1) we can infer (i) and (ii) in the table below. From (2) we can infer (iii) and (iv). Given that we have only two individuals, in (3) we replace $x$ by $a$, $y$ by $b$ and $z$ by $a$ again. Now the antecedent of the conditional is true, but the consequent contradicts (iii).

$$
\begin{array}{rl}
(i) & neighbour(a,b) \\
(ii) & neighbour(b,a) \\
(iii) & \neg neighbour(a,a) \\
(iv) & \neg neighbour(b,b) \\
(v) & neighbour(a,b) \wedge neighbour(b,a)) \rightarrow neighbour(a,a)
\end{array}
$$

More generally, assume we have a finite, non-empty domain $D = \{d_1, \ldots, d_n\}$, and draw a graph in which edges correspond to the *neighbour* relation. At some point, we add the edge $(d_{n-1}, d_n)$ to the graph, where $d_n$ is the last element of $D$. By virtue of (1), we need to add $(d_n, d_i)$, for some $d_i, i \leq n$ which is already in the graph. So we have a loop. Now by repeated appeal to (3), we can conclude that $(d_i, d_n)$ is in the graph, so again by (3), $(d_i, d_i)$ is in the graph, contradicting (2).

## Exercise 3

The two formulae:

(4) a. $\forall x.\forall y.((P\, y\, x) \rightarrow (P\, x\, y))$
b. $\forall x.\forall y.((Q_1\, y\, x) \rightarrow (Q_2\, x\, y))$

**Model:**

$$
\begin{array}{lcl}
P & = & \{(a,b),(b,a)\} \\
Q_1 & = & \{(c,d)\} \\
Q_2 & = & \{(d,c)\}
\end{array}
$$

**NL Relations:** We have to find a symmetric relation for $P$, and instances of $Q_1$, $Q_2$ which are converses of each other. For example, $P \Rightarrow$ 'is married to', $Q_1 \Rightarrow$ 'mother of', $Q_2 \Rightarrow$ 'child of'.

## Exercise 4

### Predicates

| Type | Vocabulary |
|---|---|
| IND $\rightarrow$ BOOL | block<br>cube<br>pyramid<br>red<br>green<br>yellow<br>table |
| (IND $\rightarrow$ (IND $\rightarrow$ BOOL)) | larger<br>same-size<br>smaller<br>on |

### Model

$M = \langle D, V \rangle$ where:

- $D = \{c_1, c_2, p_1, p_2, t\}$

- $V(\textit{block}) = \{c_1, c_2, p_1, p_2\}$
  $V(\textit{cube}) = \{c_1, c_2\}$
  $V(\textit{pyramid}) = \{p_1, p_2\}$
  $V(\textit{red}) = \{c_1, c_2\}$
  $V(\textit{green}) = \{p_1\}$
  $V(\textit{yellow}) = \{p_2\}$
  $V(\textit{table}) = \{t\}$
  $V(\textit{larger}) = \{\langle p_1, p_2\rangle\}$
  $V(\textit{same-size}) = \{\langle c_1, c_2\rangle, \langle c_2, c_1\rangle, \langle c_1, c_1\rangle, \langle c_2, c_2\rangle, \langle p_1, p_1\rangle, \langle p_2, p_2\rangle, \langle t, t\rangle\}$
  $V(\textit{smaller}) = \{\langle p_2, p_1\rangle\}$
  $V(\textit{on}) = \{\langle c_1, t\rangle, \langle c_2, t\rangle, \langle p_1, t\rangle, \langle p_2, c_1\rangle\}$

Note that we have tried to construct a model which corresponds fairly intuitively to our understanding of the natural language expressions. In particular, *smaller* is a relation, not a set. Suppose, instead, that it is interpreted as a set, say $S = \{p_2\}$, with the assumption that this means that $p_2$ is smaller than $p_1$, which is not in the set $S$. The problem is that this won't generalize to three elements. Suppose $p_3$ is smaller than $p_2$, which in turn is smaller than $p_1$. So $p_2$ is in $S$ because it is smaller than $p_1$ but it is also not in $S$ because it is bigger than $p_3$.

### Formulae

Now here is a possible description of the scenario:

$$\begin{aligned}\exists x_0 \exists x_1 \exists x_2 \exists x_3 \exists x_4 (&\textit{block}(x_0) \wedge \textit{block}(x_1) \wedge \textit{block}(x_2) \wedge \textit{block}(x_3) \wedge \textit{table}(x_4) \wedge \\ &x_0 \neq x_1 \wedge x_0 \neq x_2 \wedge x_0 \neq x_3 \wedge \\ &x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \\ &x_2 \neq x_3 \wedge \\ &\forall y(\textit{block}(y) \rightarrow (y = x_0 \vee y = x_1 \vee y = x_2 \vee y = x_3)) \wedge \\ &\textit{cube}(x_0) \wedge \textit{cube}(x_1) \wedge \textit{pyramid}(x_2) \wedge \textit{pyramid}(x_3) \wedge \\ &\textit{same-size}(x_0, x_1) \wedge \textit{larger}(x_2, x_3) \wedge \\ &\textit{red}(x_0) \wedge \textit{red}(x_1) \wedge \textit{green}(x_2) \wedge \textit{yellow}(x_3) \wedge \\ &\textit{on}(x_0, x_4) \wedge \textit{on}(x_1, x_4) \wedge \textit{on}(x_2, x_4) \wedge \textit{on}(x_3, x_0) \vee \textit{on}(x_3, x_1)\end{aligned}$$

We can be somewhat more succinct by making it one big conjuncion, but it can also be split up into smaller formulae.

However, this description doesn't do justice to our intuitive understanding of the sets and relations in the scenario. So we should look at adding some 'background knowledge' to supplement the description:

$$\begin{aligned}&\forall x \forall y(\textit{larger}(x, y) \leftrightarrow \textit{smaller}(y, x)) \\ &\forall x \forall y((\textit{larger}(x, y) \vee \textit{smaller}(x, y)) \leftrightarrow \neg \textit{same-size}(x, y)) \\ &\forall x(\textit{same-size}(x, x)) \\ &\forall x \forall y(\textit{same-size}(x, y) \rightarrow \textit{same-size}(y, x)) \\ &\forall x(\textit{block}(x) \rightarrow (\textit{cube}(x) \leftrightarrow \neg \textit{pyramid}(x))) \\ &\forall x(\textit{block}(x) \leftrightarrow \neg \textit{table}(x)) \\ &\forall x \forall y(\textit{on}(x, y) \leftrightarrow \neg \textit{on}(y, x)) \\ &\forall x \neg(\textit{on}(x, x))\end{aligned}$$

### Alternative Models

An alternative vocabulary would not include *same-size*, instead definining it in terms of larger and smaller. Similarly, one could remove *smaller*, and replacing it with '$\neg$*larger*', or *vice versa.* The above model respects the background knowledge given above, but if the model didn't have to satisfy this background knowledge, then the extensions of *larger* and *smaller* need not be anti-symmetric, and the extension of *same-size* need not be an equivalence relation. We could also increase the extensions of the colour predicates (making some objects both red and yellow, for instance). We could remove $\langle p_2, c_1 \rangle$ from $V(on)$ and replace it with $\langle p_2, c_2 \rangle$. And finally, we could add further individuals to the model that aren't blocks or tables.

## Exercise 5

Let's assume that (5-a) is represented as (5-b) in standard FOL, or equivalently as (5-b) in NLTK-style LF.

(5) a. Suzie shows John a dog.
b. $\exists x(dog(x) \wedge show(suzie, x, john))$
c. `some x.((dog x) and (show j x suzie))`

Let's also assume that the LF for *shows John a dog* is derived by combining *shows John* with *a dog*.

Now, let's figure out the lambda terms by working top-down from the root of the sentence. On the left-hand side of the following tables we show the (type-raised) functor NP, and on the right-hand side, the argument. So here's how the subject combines with the VP:

| Suzie | shows John a dog |
|---|---|
| `\P.(P suzie)` | `\y.some x.((dog x) and (show john x y))` |

Down at the VP level, we need to combine *a dog* with *shows John*. So we need to pull out the quantifier phrase and make it a functor:

| a dog | shows John |
|---|---|
| `\P.some x.((dog x) and (P x))` | `\X y.(X \x.(show john x y))` |

And down another level, we pull out the NP argument *John*, and make it a functor. So the expression on the right-hand side below is the representation we want for the verb *shows*:

| John | shows |
|---|---|
| `\P.(P j)` | `\Z X y.(X \x. (Z \z.(show z x y)))` |

There are two main ways of expressing the syntactic rule for ditransitives. If we stick to using `app(a, b)` as the semantic form of the mother, then we can only use binary syntactic rules:

```
# binary branching ditransitive VP
VP[num=?n,sem=<app(?dtvp,?obj)>] -> DTVP[num=?n,sem=?dtvp] NP[sem=?obj]
DTVP[num=?n,sem=<app(?v,?obj)>] -> DTV[num=?n,sem=?v] NP[sem=?obj]

DTV[num=sg,sem=<\Z X y.(X \x. (Z \z.(show z x y)))>,tns=pres] -> 'shows'
DTV[num=pl,sem=<\Z X y.(X \x. (Z \z.(show z x y)))>,tns=pres] -> 'show'
```

However, we are allowed to use more complex values in the semantics of the mother, such as the following:

```
# ternary branching ditransitive VP
VP[num=?n,sem=<((?dtvp ?obj1) ?obj2>] -> DTV[num=?n,sem=?dtvp] NP[sem=?obj1] NP[sem=?obj2]
```

Here is a model, with an evaluation of the sentence:

```
from nltk.sem import *
val = Valuation({
'suzie': 's',
'john': 'j',
'dog': {'f': True},
'show': {'j': {'f': {'s': True}}}
})

dom = val.domain
m = Model(dom, val)
g = Assignment(dom)

e = 'some x.((dog x) and ((show john x) suzie))'
print m.evaluate(e, g)
# True
```

## Exercise 6

Here are LFs for the two readings:

(6) Every person needs a doctor.

a. $\exists y(doctor(y) \wedge \forall x(person(x) \rightarrow need(x, y)))$

b. $\forall x(person(x) \rightarrow \exists y(doctor(y) \wedge need(x, y)))$

Let's consider the models $M_1$, $M_2$ defined as follows: $M_1 = \langle D, V_1 \rangle$ where:

- $D = \{p_1, p_2, d_1, d_2\}$
- $V_1(person) = \{p_1, p_2\}$
  $V_1(doctor) = \{d_1, d_2\}$
  $V_1(need) = \{\langle p_1, d_1 \rangle, \langle p_2, d_1 \rangle\}$

$M_2 = \langle D, V_2 \rangle$ where:

- $D = \{p_1, p_2, d_1, d_2\}$
- $V_2(person) = \{p_1, p_2\}$
  $V_2(doctor) = \{d_1, d_2\}$
  $V_2(need) = \{\langle p_1, d_1 \rangle, \langle p_2, d_2 \rangle\}$

(We have assumed, for simplicity, that doctors and people are disjoint!)

Now $M_1 \models$ (6-a), since $d_1$ is a doctor such that every person (i.e., $p_1$, $p_2$) needs him/her. Moreover $M_1 \models$ (6-b). By contrast $M_2 \models$ (6-b) but $M_2 \not\models$ (6-a). So $M_1$ satisfies both readings, and $M_2$ satisfies only one reading, (6-b). So $\forall M$, if $M \models$ (6-a) then $M \models$ (6-b), but not conversely. So we can conclude that (6-a) entails (6-b).

## Exercise 7

Tableaux expansion rules:

$$T_{nor}: \quad \begin{array}{c} T(\phi \ nor \ \psi) \\ \hline F\phi \\ F\psi \end{array}$$

$$F_{nor}: \quad \begin{array}{c|c} \multicolumn{2}{c}{F(\phi \ nor \ \psi)} \\ \hline T\phi & T\psi \end{array}$$

Proof of $\neg(p \vee q) \leftrightarrow (pnorq)$. We do it in two stages.

$\rightarrow$ direction:

$$\begin{array}{lll} 1. & F\neg(p \vee r) \rightarrow (p \ nor \ q) & \surd \\ 2. & T\neg(p \vee r) & 1, F_{\rightarrow}, \surd \\ 3. & F(p \ nor \ q)) & 1, F_{\rightarrow}, \surd \\ 4. & F(p \vee r) & 2, T_{\neg}, \surd \\ 5. & Fp & 4, F_{\vee} \\ 6. & Fq & 4, F_{\vee} \end{array}$$

$$\begin{array}{l|r} \hline 7. \quad Tp \quad 3, F_{nor} & Tq \quad 3, F_{nor} \end{array}$$

$\leftarrow$ direction:

$$\begin{array}{lll} 1. & F(p \ nor \ q) \rightarrow \neg(p \vee r) & \surd \\ 2. & T(p \ nor \ q)) & 1, F_{\rightarrow}, \surd \\ 3. & F\neg(p \vee r) & 1, F_{\rightarrow}, \surd \\ 4. & Fp & 2, T_{nor} \\ 5. & Fq & 2, T_{nor} \\ 6. & T(p \vee r) & 3, F_{\neg}, \surd \end{array}$$

$$\begin{array}{l|r} \hline 7. \quad Tp \quad 6, T_{\vee} & Tq \quad 6, T_{\vee} \end{array}$$

You can combine these two conditionals into a single tableau by using the following rule for biconditionals:

$$F_{\leftrightarrow}: \quad \begin{array}{c|c} \multicolumn{2}{c}{F(\phi \ nor \ \psi)} \\ \hline T\phi & F\phi \\ F\psi & T\psi \end{array}$$

## Exercise 8

In the following grammar, the event variable $e$ gets passed through the rules by successive $\lambda$-abstraction and $\beta$-conversion. In general — if we ignore type-raising over *NP* arguments — the basic type of intransitive verbs become IND $\rightarrow$ (IND $\rightarrow$ BOOL), i.e., binary relations over individuals and events. As a consequence, we change the type of *NP*s from (IND $\rightarrow$ BOOL) $\rightarrow$ BOOL to ((IND $\rightarrow$ (IND $\rightarrow$ BOOL)) $\rightarrow$ (IND $\rightarrow$BOOL), i.e., a function from binary relations to sets. This means that the event variable of a *VP* that contains a quantified *NP* object will still be available for adverbial modification. Finally, the semantic representation of an *S*, e.g., the value of `(?subj ?vp)`, will be of the form `\e. phi`. Existential quantification of the event variable takes place as a supplementary step in the semantics for building *S* from *NP* and *VP*.

```
% start S
S[sem = <some e.((?subj ?vp) e)>] -> NP[num=?n,sem=?subj] VP[num=?n,sem=?vp]

NP[num=?n,sem=<app(?det,?nom)> ] -> Det[num=?n,sem=?det]  Nom[num=?n,sem=?nom]
NP[loc=?l,num=?n,sem=?np] -> PropN[loc=?l,num=?n,sem=?np]
```

```
Nom[num=?n,sem=?nom] -> N[num=?n,sem=?nom]
Nom[num=?n,sem=<app(?pp,?nom)>] -> N[num=?n,sem=?nom] PP[sem=?pp]

VP[num=?n,sem=<app(?v,?obj)>] -> TV[num=?n,sem=?v] NP[sem=?obj]
VP[num=?n,sem=?v] -> IV[num=?n,sem=?v]

VP[num=?n,sem=<app(?pp,?vp)>] -> VP[num=?n,sem=?vp] PP[sem=?pp]
VP[num=?n,sem=<app(?adv,?vp)>] -> VP[num=?n,sem=?vp] Adv[sem=?adv]

PropN[-loc,num=sg,sem=<\R e. (R john e)>] -> 'John'
PropN[-loc,num=sg,sem=<\R e. (R mary e)>] -> 'Mary'
PropN[-loc,num=sg,sem=<\R e. (R suzie e)>] -> 'Suzie'
PropN[-loc,num=sg,sem=<\R e. (R fido e)>] -> 'Fido'
PropN[+loc, num=sg,sem=<\P.(P noosa)>] -> 'Noosa'

NP[-loc, num=sg, sem=<\P.\x.(P x)>] -> 'who'

Det[num=sg,sem=<\P R e. all x. ((P x) implies (R x e))>] -> 'every'
Det[num=pl,sem=<\P R e. all x. ((P x) implies (R x e))>] -> 'all'
Det[sem=<\P R e. some x. ((P x) and (R x e))>] -> 'some'
Det[num=sg,sem=<\P R e. some x. ((P x) and (R x e))>] -> 'a'

N[num=sg,sem=<boy>] -> 'boy'
N[num=pl,sem=<boy>] -> 'boys'
N[num=sg,sem=<girl>] -> 'girl'
N[num=pl,sem=<girl>] -> 'girls'
N[num=sg,sem=<dog>] -> 'dog'
N[num=pl,sem=<dog>] -> 'dogs'

TV[num=sg,sem=<\X y. (X \x e. ((agent y e) and ((chase e) and (patient x e))))>,tns=pres] -> 'chases'
TV[num=pl,sem=<\X y. (X \x e. ((agent y e) and ((chase e) and (patient x e))))>,tns=pres] -> 'chase'
TV[num=sg,sem=<\X y. (X \x e. ((agent y e) and ((see e) and (patient x e))))>,tns=pres] -> 'sees'
TV[num=pl,sem=<\X y. (X \x e. ((agent y e) and ((see e) and (patient x e))))>,tns=pres] -> 'see'
IV[num=sg,sem=<\x e. ((agent x e) and (bark e))>,tns=pres] -> 'barks'
IV[num=pl,sem=<\x e. ((agent x e) and (bark e))>,tns=pres] -> 'bark'
IV[num=sg,sem=<\x e. ((agent x e) and (walk e))>,tns=pres] -> 'walks'
IV[num=pl,sem=<\x e. ((agent x e) and (walk e))>,tns=pres] -> 'walk'

Adv[sem=<\R x e. ((slow e) and (R x e))>] -> 'slowly'
Adv[sem=<\R x e. ((thoughtful e) and (R x e))>] -> 'thoughtfully'
```

Since the NLTK `LogicParser` is set up to only recognize `x, y, z` as individual variables, we can modify the *S* rule along the following lines to ensure that the resulting LFs can be parsed:

```
S[sem = <some x.((event x) and ((?subj ?vp) e))>] -> NP[num=?n,sem=?subj] VP[num=?n,sem=?vp]
```

Sample model:

```
from nltk.sem import *
v = """
suzie => s
john => j
dog => {f}
girl => {s}
boy => {j}
event => {e1, e2, e3, e4, e5, e6}
```

```
walk => {e1, e2, e3}
slow => {e2}
see => {e4, e5, e6}
agent => {(e1, f), (e2, j), (e3, s), (e4, s), (e5, j), (e6, f)}
patient => {(e4, j), (e5, f), (e6, s)}
"""
val = parse_valuation(v)
dom = val.domain
m = Model(dom, val)
g = Assignment(dom)

expr = '\ z. some x. ((event z) and ((boy x) and ((agent suzie z) and ((see z) and (patient x z))

print m.evaluate(expr, g)
#True
```