Exercises on Semantic Parsing and Information Extraction

Ion Androutsopoulos, 2022–23

There are no exercises to be handed in as assignments in this part.

1. Represent in First-Order Predicate Logic (FOPL) the meaning of the following sentences.

(i) John is a student.
(ii) Every student is clever.
(iii) Every student who has passed the AI course is clever.
(iv) John has passed at least one course.
(v) John has passed exactly one course.
(vi) Every student who has passed at least one course is clever.

Answer:
(i) $Student(John)$
(ii) $\forall x \left(Student(x) \rightarrow Clever(x)\right)$
(iii) $\forall x \left((Student(x) \wedge Passed(x, AI)) \rightarrow Clever(x)\right)$
(iv) $\exists x \, (Course(x) \wedge Passed(John, x))$
(v) $\exists x \, (Course(x) \wedge Passed(John, x) \; \wedge \forall y \, ((Course(y) \wedge Passed(John, y)) \; \rightarrow (x = y)) \,)$
(vi) $\forall x \left((Student(x) \wedge \exists y \, (Course(y) \; \wedge Passed(x, y))) \rightarrow Clever(x)\right)$

2. Represent in FOPL the meaning of the following sentences.

i. Milos and Suzos are dogs.
ii. Psita and Rana are cats.
iii. Psita likes Milos.
iv. Rana was bitten by Milos or Suzos.
v. Some dog bit some cat.
vi. No cat likes Suzos.
vii. If some dog bit some cat, then no cat likes that dog.

Answer:
(i) $(Dog(Milos) \wedge Dog(Suzos))$
(ii) $(Cat(Psita) \wedge Cat(Rana))$
(iii) $Likes(Psita, Milos)$
(iv) $(Bite(Milos, Rana) \vee Bite(Suzos, Rana))$
(v) $\exists x \, \exists y \, (Dog(x) \wedge Cat(y) \wedge Bite(x, y))$
(vi) $\forall y \, (Cat(y) \Rightarrow \neg Likes(y, Suzos))$
(vii) $\forall x \, ((Dog(x) \wedge \; \exists y \, (Cat(y) \wedge Bite(x, y))) \Rightarrow \forall z \, (Cat(z) \Rightarrow \neg Likes(z, x)))$

3. Add a few adjectives (e.g., “cheap”, “expensive”, “kind”, “loyal”) to the grammar of slides 7–12, to allow it to handle (and translate to FOPL) sentences like “A kind loyal customer wants a cheap flight”.

4. (Optional) Implement in Keras/Tensorflow or PyTorch (e.g., using LSTMs or BERT) your own (a) named entity recognizer (NER) and/or (b) relation extractor, for one of the corresponding datasets of Papers with Code.[1]

[1] See https://paperswithcode.com/datasets?task=named-entity-recognition-ner and https://paperswithcode.com/datasets?q=relation+extraction.

The following exercises refer to optional material of the slides.

5. (Optional) Convert the grammar of exercise 2(c) of the previous section (syntactic parsing) to a DCG, so that the new grammar will also compute the length of the input sequence, as in the following examples.

```
?- phrase( s(Length), [a, a, b, b, b, c, c]).
Length = 7
?- phrase( s(Length), [a, a, b, b, b, c]).
no
```

Answer:

```
s(NewValue) --> [a], p(Value), [c], {NewValue is Value + 2}.
p(NewValue) --> [a], p(Value), [c], {NewValue is Value + 2}.
p(Value) --> q(Value).
q(1) --> b.
q(NewValue) --> b, q(Value), {NewValue is Value + 1}.
```

6. (Optional) Convert the following CFG to a DCG:

S → NP VP	Det → the
NP → Det N	N → dog | cat
VP → V NP	V → chased | bit

and extend it, so that if the DCG is given to Prolog with the following goal:

phrase(s(Action, Agent, Object), [ο, σκύλος, κυνήγησε, τη, γάτα]).

Prolog will respond:

Action = chase, Agent = dog, Object = cat,

and if the DCG is given to Prolog with the following goal:

phrase(s(Action, Agent, Object), [ο, σκύλος, δάγκωσε, τη, γάτα]).

Prolog will respond:

Action = bite, Agent = dog, Object = cat.

Answer:

```
s(Action, Agent, Object) --> np(Agent), vp(Action, Object).
np(Entity) --> det, n(Entity).
vp(Action, Object) --> v(Action), np(Object).
Det --> the.
n(dog) --> [dog].
n(cat) --> [cat].
v(chase) --> [chased].
v(bite) --> [bit].
```

7. (Optional) Modify the DCG of slide 33 (semantic parsing for the arithmetic language) to allow using constants from zero to 9999. As a simplification, write 10 as [one, ten], 12 as [one, ten, two], 23 as [two, ten, three], 100 as [one, hundred], 110 as [one, hundred, one, ten], 123 as [one, hundred, two, ten, three], 1000 as [one, thousand], 1123 as [one, thousand, one, hundred, two, ten, three] etc. For example, the following Prolog goal:

?-phrase(expression(V), [open, two, thousand, three, hundred, five, ten, two, plus, two, ten, one, close]).

should return:

V = 2373

and the following goal:

?-phrase(expression(V), [open, open, two, thousand, three, hundred, five, ten, two, plus, two, ten, one, close, minus, one, thousand, seven, ten, close]).

should return:

V = 1303

Answer:

```
digit(0) --> [zero].        digit(1) --> [one].
digit(2) --> [two].         digit(3) --> [three].
digit(4) --> [four].        digit(5) --> [five].
digit(6) --> [six].         digit(7) --> [seven].
digit(8) --> [eight].       digit(9) --> [nine].

number(Value) --> digit(Value).
number(NewValue) --> digit(Value1), [ten], number(Value2),
        {Value2 < 10, NewValue is Value1 * 10 + Value2}.
number(NewValue) --> digit(Value1), [ten], {NewValue is Value1 * 10}.
number(NewValue) --> digit(Value1), [hundred], number(Value2),
        {Value2 < 100, NewValue is Value1 * 100 + Value2}.
number(NewValue) --> digit(Value1), [hundred], {NewValue is Value1 * 100}.
number(NewValue) --> digit(Value1), [thousand], number(Value2),
        {Value2 < 1000, NewValue is Value1 * 1000 + Value2}.
number(NewValue) --> digit(Value1), [thousand],  {NewValue is Value1 * 1000}.

expression(Value) --> number(Value).
expression(NewValue) --> [open], expression(Value1), [plus], expression(Value2), [close],
        {NewValue is Value1 + Value2}.
expression(NewValue) --> [open], expression(Value1), [minus], expression(Value2) , [close],
        {NewValue is Value1 - Value2}.
```

8. (Optional) Write a DCG that will allow sentences like:

[open,the,windows,of,the,living,room], [close,the,small,window,of,the,living,room], [switch,off,the,large,light,of,the,small,kitchen], [switch,on,the,lights,of,the,small,bathrooms]

The grammar should allow using “switch on” and “switch off” only with lights, not windows; and it should allow using “open” and “close” only with windows, not lights. For example, sentences like the following should not be allowed:

X [switch,off,the,small,window,of,the,living,room],
X [open,the,large,light,of,the,small,kitchen]

For the allowed sentences, it should be possible to use the grammar as illustrated below to obtain semantic representations:

```
?- phrase(s(Action, Type1, Size1, Number1, Type2, Size2, Number2,
[open,the,large,windows,of,the,small,living,room]).
Action = open, Type1 = window, Size1 = large, Number1 = plural,
Type2 = livingroom, Size2=small, Number2 = singular

?- phrase(s(Action, Type1, Size1, Number1, Type2, Size2, Number2,
[switch,on,the,lights,of,the,small,bathrooms]).
Action = on, Type1 = light, Size1 = unknown, Number1 = plural,
Type2 = bathroom, Size2=small, Number2 = plural

?- phrase(s(Action, Type1, Size1, Number1, Type2, Size2, Number2,
[close,the,windows]).
Action = close, Type1 = window, Size1 = unknown, Number1 = plural,
Type2 = unknown, Size2=unknown, Number2 = unknown
```

Answer:

```
s(Action, Type1, Size1, Number1, Type2, Size2, Number2) -->
        vp(Action, Type1, Size1, Number1, Type2, Size2, Number2).

vp(Action, Type, Size, Number, uknown, uknown, unknown) -->
        v(Action, Type), np(Number, Type, Size).

vp(Action, Type1, Size1, Number1, Type2, Size2, Number2) -->
        v(Action, Type1), np(Number1, Type1, Size1), pp(Number2, Type2, Size2).

np(Number, Type, Size) -->
        det, nominal(Number, Type, Size).

nominal(Number, Type, Size) -->
        adj(Size), n(Number, Type).

nominal(Number, Type, unknown) -->
        n(Number, Type).

pp(Number, Type, Size) -->
        prep, np(Number, Type, Size).

v(on, light) --> [switch, on].
v(open, window) --> [open].
v(off, light) --> [switch, off].
v(close, window) --> [close].

det --> [the].

prep --> [of].

n(singular, window) --> [window].
n(plural, window) --> [windows].
n(singular, light) --> [light].
n(plural, light) --> [lights].
n(singular, livingroom) --> [living, room].
n(plural, livingroom) --> [riving, rooms].
```

n(singular, bathroom) --> [bathroom].
n(plural, bathroom) --> [bathrooms].
n(singular, kitchen) --> [kitchen].
n(plural, kitchen) --> [kitchens].

adj(small) --> [small].
adj(large) --> [large].

9. (Optional) Consider the following DCG for English:

```
v(event(saw, Who, What, How)) --> [saw].

det --> [the].

n(entity(scientist, Specifier)) --> [scientist].
n(entity(telescope, Specifier)) --> [telescope].

pn(paris) --> [paris].

prep(with) --> [with].              prep(from) --> [from].

pp(specifier(PSem, NPSem)) --> prep(PSem), np(NPSem).

pron(we) --> [we].

nominal(NSem)--> n(NSem).
nominal(entity(Type, Specifier)) -->
  n(entity(Type, Specifier)), pp(Specifier).

np(NominalSem) --> det, nominal(NominalSem).
np(PNSem) --> pn(PNSem).
np(PronSem) --> pron(PronSem).

vp(event(Type, Who, NPSem, How)) -->
  v(event(Type, Who, NPSem, How)), np(NPSem).
vp(event(Type, Who, NPSem, How)) -->
  v(event(Type, Who, NPSem, How)), np(NPSem), pp(How).

s(event(Type, Who, What, How)) -->
  np(Who), vp(event(Type, Who, What, How)).
```

or the following very similar DCG for Greek:

```
v(event(saw, we, What, How)) --> [είδαμε].

det --> [τον].                      det --> [το].

n(entity(scientist, Specifier)) --> [επιστήμονα].
n(entity(telescope, Specifier)) --> [τηλεσκόπιο].

pn(paris) --> [παρίσι].

prep(with) --> [με].                prep(from) --> [από].

pp(specifier(PSem, NPSem)) --> prep(PSem), np(NPSem).
```

```
nominal(NSem)--> n(NSem).
nominal(entity(Type, Specifier)) -->
  n(entity(Type, Specifier)), pp(Specifier).

np(NominalSem) --> det, nominal(NominalSem).
np(PNSem) --> det, pn(PNSem).

vp(event(Type, Who, NPSem, How)) -->
  v(event(Type, Who, NPSem, How)), np(NPSem).
vp(event(Type, Who, NPSem, How)) -->
  v(event(Type, Who, NPSem, How)), np(NPSem), pp(How).

s(VPSem) --> vp(VPSem).
```

Write the values that Prolog will return for Sem, when given the following goal for the English grammar:

```
phrase(S(Sem), [we, saw, the, scientist, with, the,
telescope]).
```

or the following goal for the Greek grammar:

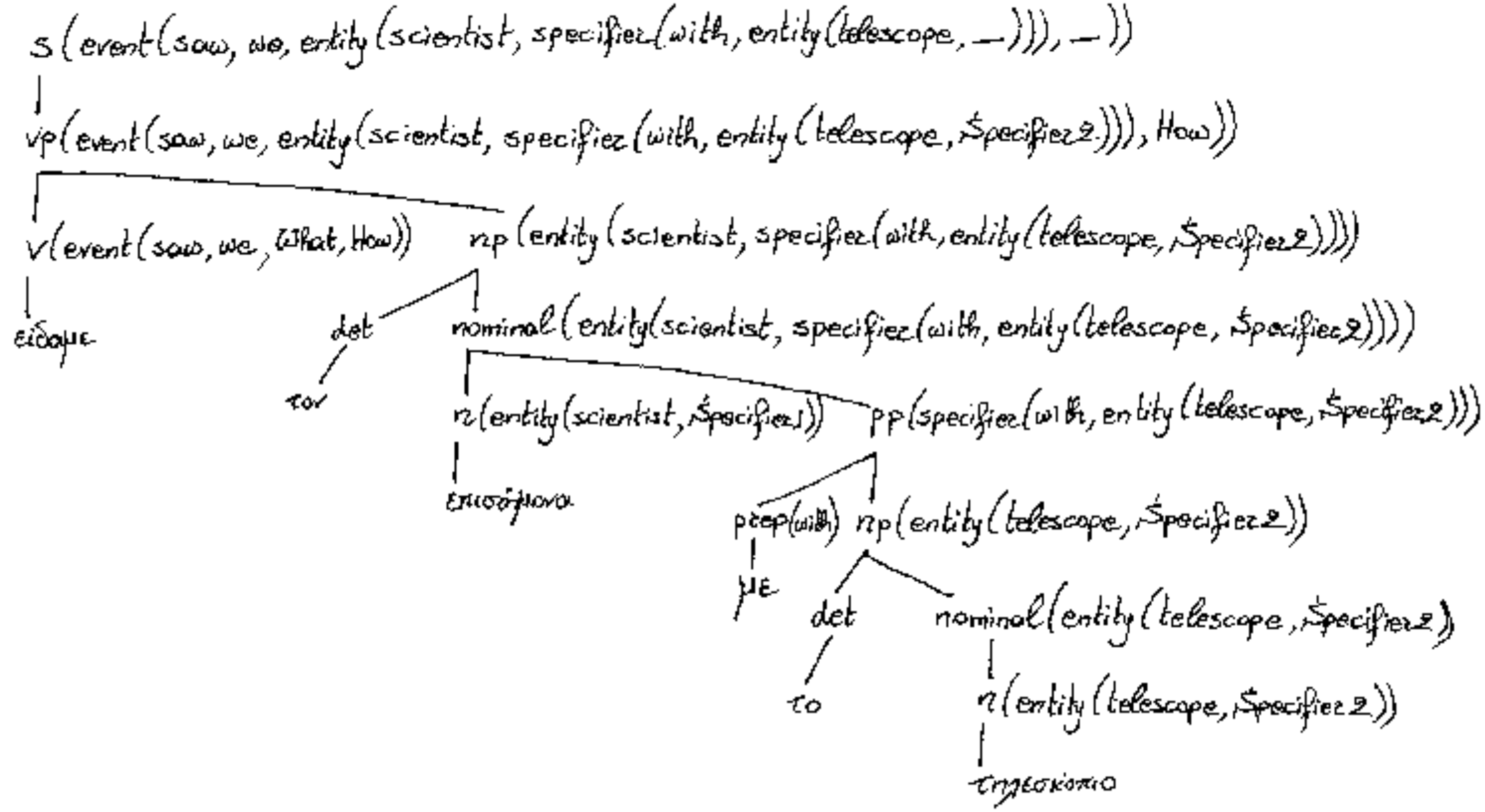
```
phrase(S(Sem), [είδαμε, τον, επιστήμονα, με, το, τηλεσκόπιο]).
```

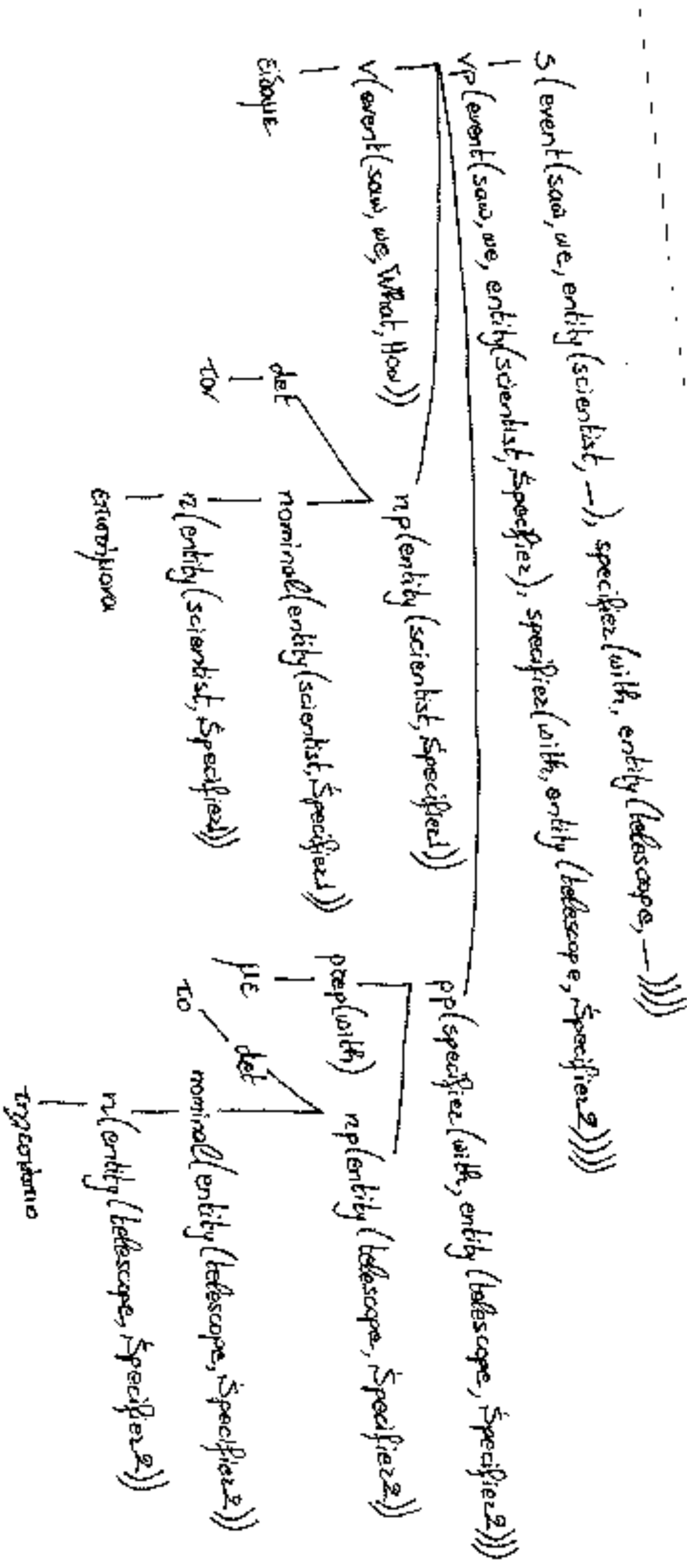
Use "_" to denote variables with no values. Explain how you these values are obtained.

Answer: We obtain two parse trees and two responses for Sem, as illustrated below for the Greek sentences (the English trees are very similar). The two values of Sem are:

Sem = event(saw, we, entity(scientist, specifier(with, entity(telescope, _))), _)

Sem = event(saw, we, entity(scientist, _), specifier(with, entity(telescope, _)))


s(event(saw, we, entity(scientist, specifier(with, entity(telescope, _))), _))
vp(event(saw, we, entity(scientist, specifier(with, entity(telescope, Specifier2))), How))
v(event(saw, we, What, How))
np(entity(scientist, specifier(with, entity(telescope, Specifier2))))
είδαμε
det
nominal(entity(scientist, specifier(with, entity(telescope, Specifier2))))
τον
n(entity(scientist, Specifier1))
pp(specifier(with, entity(telescope, Specifier2)))
επιστήμονα
prep(with)
np(entity(telescope, Specifier2))
με
det
nominal(entity(telescope, Specifier2))
το
n(entity(telescope, Specifier2))
τηλεσκόπιο



If you are curious, you can also try:

```
phrase(S(Sem), [we, saw, the, scientist, with, the, telescope,
from, paris]).
```