Symbolic AI Coursework - Grammars, Parsing and Logic

2019.05.07

Haorui Chen

1. Exercise 1 (Building a CFG)

Assume we have the following corpus of individual sentences:

Steel is an alloy. Steel contains carbon.

1.1 Read the first section on Context Free Grammars (CFGs) here.

s -> n vp

np -> det n

vp -> v np

vp -> v n

det -> an

n -> steel

n -> alloy

n -> carbon

v -> is

v -> contains

1.2 Write down a Context Free Grammar (CFG) which is able to recognize these sentences. Start by listing the POS tags and the phrasal nodes which you will use. Then write down the grammar.

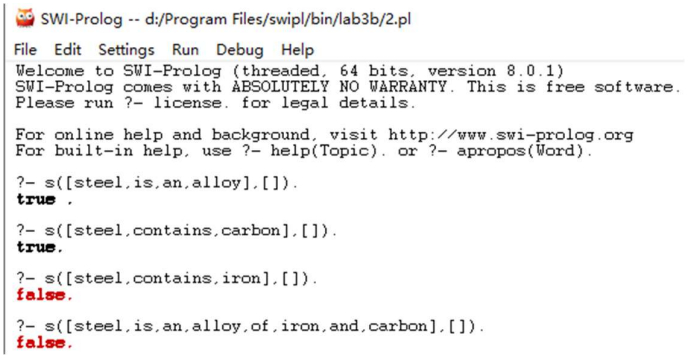
s n steel vp v is np det an n alloy; s n steel v contains n carbon

2. Exercise 2 (Building a CFG in Prolog (using Difference Lists))

Now that you have your grammar specfied, let's build a Prolog parser to recognise and generate sentences using this grammar.

2.3 Check if the grammar is able to recognize the four sentences.

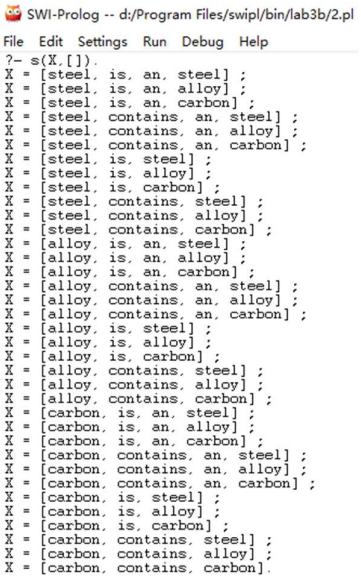
From pic 2-1 we can see that all 4 queries can get us judgements (whether true or false), meaning that the grammar has recognize them.



2-1 results of running these 4 queries in prolog

2.4 Generate all the sentences which can be recognized by this grammar.

Pic 2-2 shows all sentences that can be generated by prolog.



2-2 sentences generated from grammar

2.5 Include your program as a properly referenced appendix into your report.

```
s(X,Z):-    n(X,Y), vp(Y,Z).
np(X,Z):-   det(X,Y),n(Y,Z).
vp(X,Z):-   v(X,Y), np(Y,Z).
vp(X,Z):-   v(X,Y),  n(Y,Z).
det([an|W],W).
n([steel|W],W).
n([alloy|W],W).
n([carbon|W],W).
v([is|W],W).
```

```
v([contains|W],W).
```

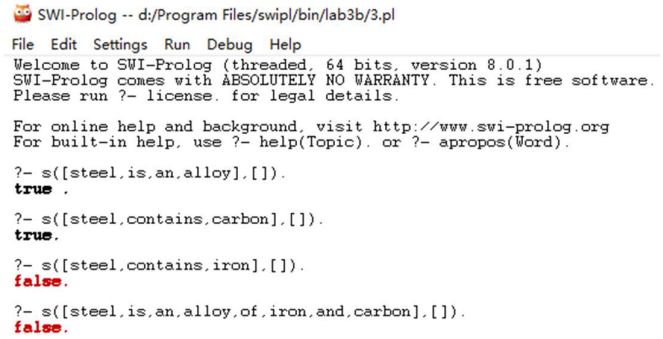
3. Exercise 3 (DFGs)

Prolog provides a convenient way to represent CFGs, using Definite Clause Grammars (DFGs).

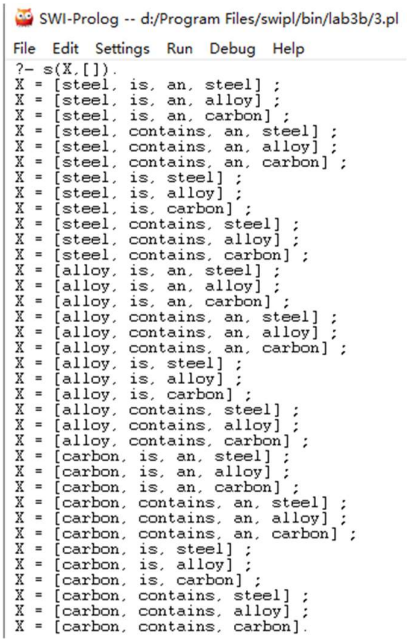
3.2 Rephrase the previous grammar into a DFG.

```
s  -->  n,vp.
np  -->  det,n.
vp - ->  v,np.
vp --> v,n.
det  --> [an].
n  --> [steel].
n  --> [alloy].
n  --> [carbon].
v  --> [is].
v  --> [contains].
```

3.3 Check if you can recognize and generate the same sentences.



3-1 results of running 4 queries in prolog after rephrasing



3-2 sentences generated from grammar

The results in pic 3-1 and 3-2 are all the same as those in previous questions, showing that it can recognize and generate the same sentences.

3.4 Include your program as a properly referenced appendix into your report.

```
s  -->  n, vp.
np  -->  det, n.
vp -->  v, np.
vp --> v, n.
det  --> [an].
n  --> [steel].
n  --> [alloy].
n  --> [carbon].
v  --> [is].
v  --> [contains].
```

4. Exercise 4 (CCGs)

Now we will look into an alternative way to encode grammars using Combinatory Categorical Grammars (CCGs). Given the following sets of natural language statements and questions:

Statements:

Steel is an alloy. Steel contains carbon.

Questions:

Does Ferrite have high hardness? Which material has the lowest tensile strength?

4.2 Create a CCG Grammar which recognizes these sentences. Focus on the syntactic dimension of the grammar (do not include the lambda calculus features).

Steel is an alloy.

Steel := NP
is: := S\NP/NP
an: := NP/N
alloy := N

Steel contains carbon.

Steel := NP
Contains := S\NP/NP
Carbon := NP

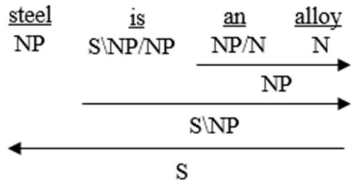
Does Ferrite have high hardness?

Does := S/S
Ferrite := NP
have := S\NP/NP
high := NP/N
hardness := N

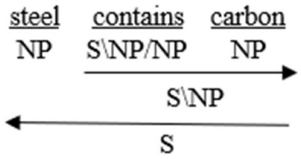
Which material has the lowest tensile strength?

Which := (S/(S\NP))/N
material := N
has := S\NP/NP
the lowest := NP/N
tensile strength := N

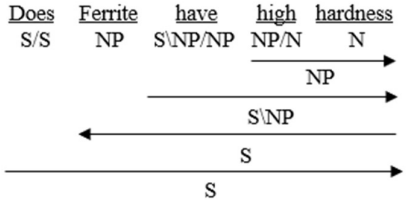
4.3 Draw the derivation trees for each sentence.



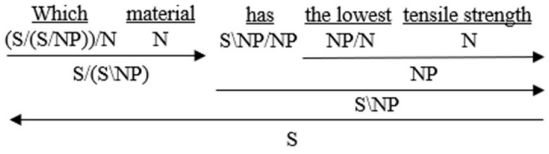
4-1 parse tree for 'Steel is an alloy'



4-2 parse tree for 'Steel contains carbon'



4-3 parse tree for 'Does Ferrite have high hardness'

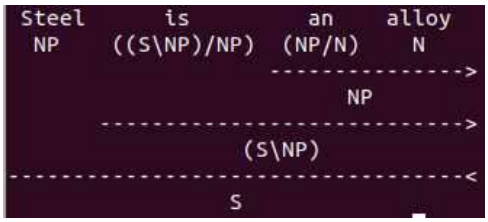


4-4 parse tree for'Which material has the lowest tensile strength'

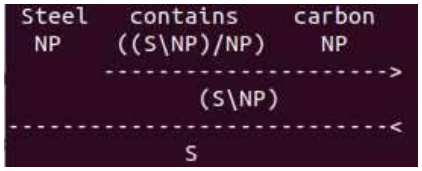
5 Exercise 5 (CCG Syntactic Parser)

NLTK is a lovely Python library for playing with natural language. We will use it to implement a syntactic parser for the grammar that you designed in the previous exercise.

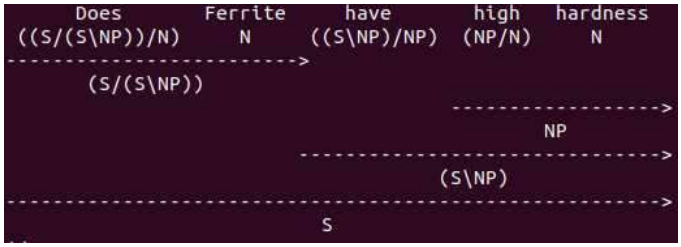
5.3 Print the derivation tree for the four sentences.



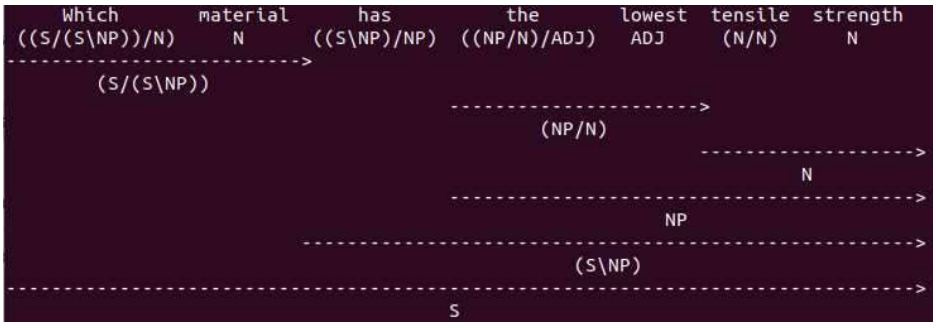
5-1 parse tree for 'Steel is an alloy'



5-2 parse tree for 'Steel contains carbon'



5-3 parse tree for 'Does Ferrite have high hardness'



5-4 parse tree for'Which material has the lowest tensile strength'

5.4 Include your program as a properly referenced appendix into your report.

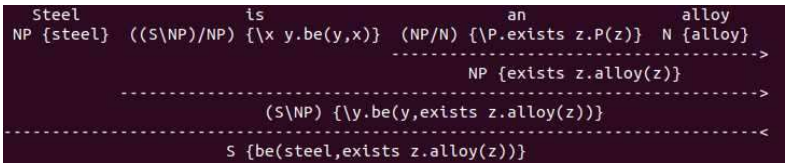
```
from nltk.ccg import chart, lexicon
from nltk.ccg.chart import printCCGDerivation
lex = lexicon.fromstring('''
:- S, NP, N
Steel => NP
is => (S\\NP)/NP
an => NP/N
alloy => N
''', False)
parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Steel is an alloy".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])
lex = lexicon.fromstring('''
:- S, NP, N
```

```
Steel => NP
contains => (S\\NP)/NP
carbon => NP
''', False)
parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Steel contains carbon".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])
lex = lexicon.fromstring('''
:- S, NP, N
Does => (S/(S\\NP))/N
Ferrite => N
have => (S\\NP)/NP
high => NP/N
hardness =>N
''', False)
parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Does Ferrite have high hardness".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])
lex = lexicon.fromstring('''
:-S, NP, N, ADJ
Which => (S/(S\\NP))/N
material => N
has => (S\\NP)/NP
the => (NP/N)/ADJ
lowest => ADJ
tensile => N/N
strength => N
''', False)
parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Which material has the lowest tensile strength".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])
```

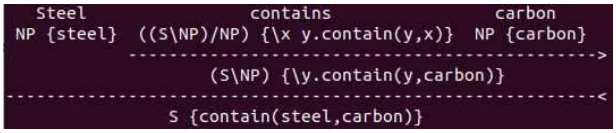
6 Exercise 6 (CCG Semantic Parser)

Now we will proceed to the next step and add a semantic layer to our grammar.

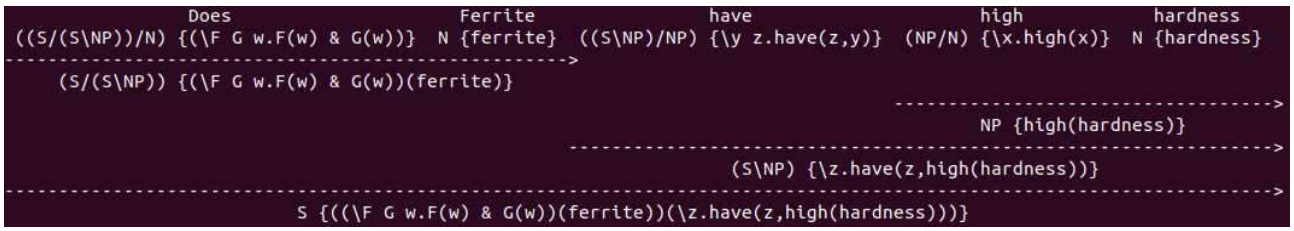
6.3 Print the derivation tree for the sentences in the corpus using the semantic features.



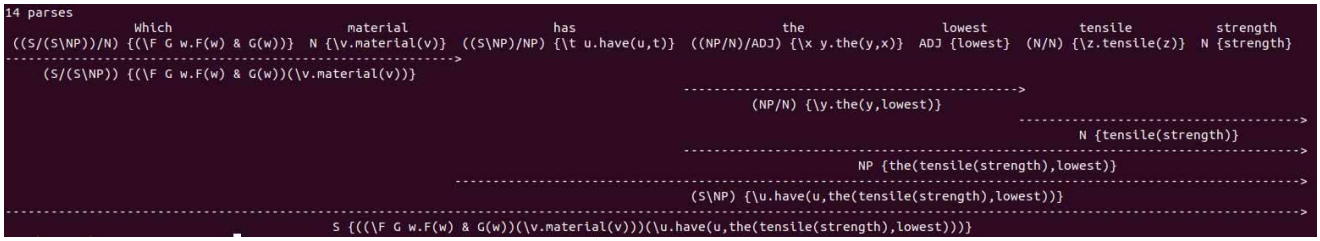
6-1 parse tree for 'Steel is an alloy' with semantic



6-2 parse tree for 'Steel contains carbon' with semantic



6-3 parse tree for 'Does Ferrite have high hardness' with semantic



6-4 parse tree for'Which material has the lowest tensile strength' with semantic

6.4 relevant code

```
from nltk.ccg import chart, lexicon
from nltk.ccg.chart import printCCGDerivation
lex = lexicon.fromstring('''
:- S, NP, N
Steel => NP {steel}
is => (S\\NP)/NP {\\x y.be(y, x)}
an => NP/N {\\P.exists z.P(z)}
alloy => N {alloy}
''', True)
parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Steel is an alloy".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])
lex = lexicon.fromstring('''
:- S, NP, N
Steel => NP {steel}
```

```
contains => (S\\NP)/NP {\\x y.contain(y, x)}
carbon => NP {carbon}
''', True)
parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Steel contains carbon".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])
lex = lexicon.fromstring('''
:- S, NP, N
Does => (S/(S\\NP))/N {\\F G w.F(w) & G(w)}
Ferrite => N {ferrite}
have => (S\\NP)/NP {\\y z.have(z,y)}
high => NP/N {\\x.high(x)}
hardness => N {hardness}
''', True)
parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Does Ferrite have high hardness".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])
lex = lexicon.fromstring('''
:- S, NP, N, ADJ
Which => (S/(S\\NP))/N {\\F G w.F(w) & G(w)}
material => N {\\v.material(v)}
has => (S\\NP)/NP {\\t u.have(u,t)}
the => (NP/N)/ADJ {\\x y.the(y,x)}
lowest => ADJ {lowest}
tensile => N/N {\\z.tensile(z)}
strength => N {strength}
''', True)
parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Which material has the lowest tensile strength".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])
```

7 Exercise 7 (Question Answering)

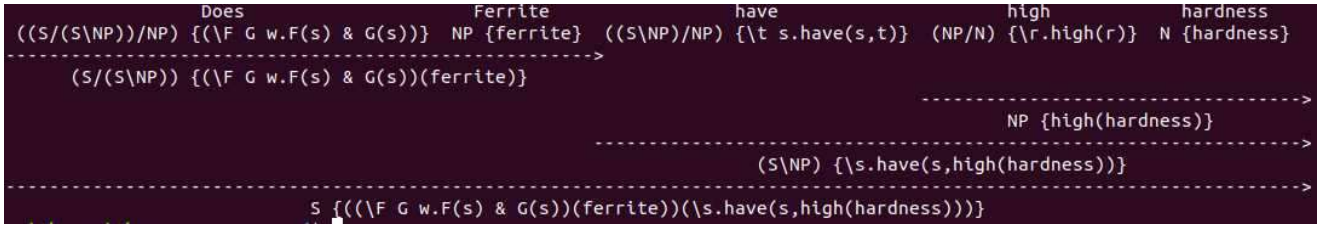
Now we will connect the questions to a knowledge base. Consider the following KB of facts:
Ferrite has 0.1 tensile strength. Ferrite has low hardness.
Perlite has 0.3 tensile strength. Perlite has medium hardness.
Austenite has 0.4 tensile strength. Austenite has medium hardness.
Cementite has 0.7 tensile strength. Cementite has high hardness.
and the questions: Does Ferrite have high hardness? Which material has the lowest tensile strength?

7.1 Manually convert the facts into a NLTK-style lambda-calculus logical form.

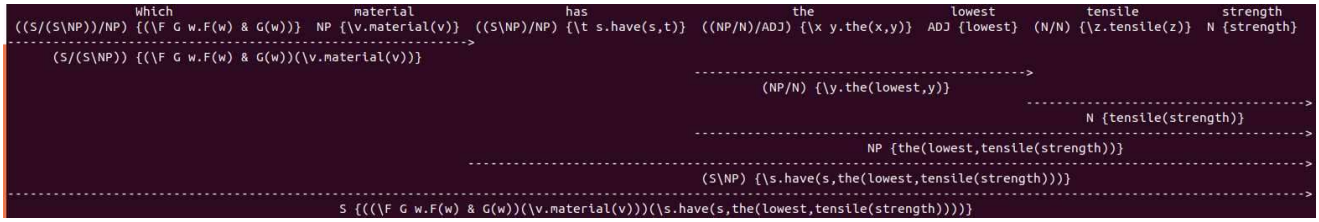
Ferrite has 0.1 tensile strength. ➔ have(ferrite, 0_1(tensile(strength)))
Ferrite has low hardness. ➔ have(ferrite, low(hardness))
Perlite has 0.3 tensile strength. ➔ have(perlite, 0_3(tensile(strength)))
Perlite has medium hardness. ➔ have(perlite, medium(hardness))
Austenite has 0.4 tensile strength. ➔ have(austenite, 0_4(tensile(strength)))
Austenite has medium hardness. ➔ have(austenite, medium(hardness))
Cementite has 0.7 tensile strength. ➔ have(cementite, 0_7(tensile(strength)))
Cementite has high hardness. ➔ have(cementite, high(hardness))

7.3 Run the parser and print the parse trees for the two questions.

Minor adjustments have been added to syntax of words in these two questions, basically changing N to NP.



7-1 parse tree for 'Does Ferrite have high hardness' with semantic



7-2 parse tree for'Which material has the lowest tensile strength' with semantic

7.4 How would you use the parser output to compute an answer for the query over the KB?

Basically, we are doing text mining in the whole lab, so when we read paragraphs into parser, we get results in the form of parse tree. From the parse tree we need to judge by ourselves the result of the query. For example, for the first query, we can read the facts from parse tree to know ferrite has low and know the first query is wrong; for the second query, we get tensile strength of all 4 materials, by comparing them we know ferrite have the lowest.

7.5 relevant code

```
from nltk.ccg import chart, lexicon
from nltk.ccg.chart import printCCGDerivation

lex = lexicon.fromstring('''
:- S, N, NP, ADJ
```

```
Which => (S/(S\\NP))/NP {\\F G w.F(w) & G(w)}
material => NP {\\v.material(v)}
has => (S\\NP)/NP {\\t s.have(s,t)}
have => (S\\NP)/NP {\\t s.have(s,t)}
the => (NP/N)/ADJ {\\x y.the(x,y)}
lowest => ADJ {lowest}
tensile => N/N {\\z.tensile(z)}
strength => N {strength}
Does => (S/(S\\NP))/NP {\\F G w.F(s) & G(s)}
Ferrite => NP {ferrite}
Perlite => NP {perlite}
Austenite => NP {austenite}
Cementite => NP {cemenite}
high => NP/N {\\r.high(r)}
medium => NP/N {\\r.medium(r)}
low => NP/N {\\r.low(r)}
hardness => N {hardness}
0_1 => NP/N {\\q.0_1(q)}
0_3 => NP/N {\\q.0_3(q)}
0_4 => NP/N {\\q.0_4(q)}
0_7 => NP/N {\\q.0_7(q)}
''', True)
parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Ferrite has 0_1 tensile strength".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])

parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Perlite has 0_3 tensile strength".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])

parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Austenite has 0_4 tensile strength".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])

parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Cementite has 0_7 tensile strength".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])

parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Ferrite has low hardness".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])

parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Perlite has medium hardness".split()))
```

```
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])

parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Austenite has medium hardness".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])

parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Which material has the lowest tensile strength".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])

parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
parses = list(parser.parse("Does Ferrite have high hardness".split()))
print(str(len(parses)) + " parses")
printCCGDerivation(parses[0])
```