sentence --> noun_phrase, verb_phrase.

noun_phrase --> proper_noun.
noun_phrase --> article, adjective, noun.
noun_phrase --> article, noun.

verb_phrase --> intransitive_verb.
verb_phrase --> transitive_verb, noun_phrase.

article --> [the].

adjective --> [lazy].
adjective --> [rapid].

proper_noun --> [achilles].
noun --> [turtle].
intransitive_verb --> [sleeps].
transitive_verb --> [beats].
Parse tree
Exercise 7.1
Exercise 7.2 (1)
Exercise 7.2 (2)
Difference lists in grammar rules

sentence(NP1-VP2):-
  noun_phrase(NP1-VP1),
  verb_phrase(VP1-VP2)
Meta-level vs. object-level

**GRAMMAR**

- \( s \rightarrow np, vp \)

**PARSING**

- \(?-\text{phrase}(s,L)\)

- \( s(L,L0) :- np(L,L1), \)
  \( \text{vp}(L1,L0) \)

- \(?-s(L,[])\)
sentence --> noun_phrase(N), verb_phrase(N).
noun_phrase --> article(N), noun(N).
verb_phrase --> intransitive_verb(N).

article(singular) --> [a].
article(singular) --> [the].
article(plural) --> [the].
noun(singular) --> [turtle].
noun(plural) --> [turtles].
intransitive_verb(singular) --> [sleeps].
intransitive_verb(plural) --> [sleep].

Non-terminals with arguments
Constructing parse trees
numeral(N) --> n1_999(N).
numeral(N) --> n1_9(N1), [thousand], n1_999(N2),
    {N is N1*1000+N2}.
numeral(N) --> n1_999(N).
numeral(N) --> n1_9(N1), [hundred], n1_99(N2),
    {N is N1*100+N2}.
numeral(N) --> n1_99(N).
numeral(N) --> n10_19(N).
numeral(N) --> n20_90(N).
numeral(N) --> n20_90(N1), n1_9(N2), {N is N1+N2}.
numeral(0) --> [].
n0_9(N) --> n1_9(N).
n0_9(N) --> n10_19(N).
n1_9(1) --> [one].
n1_9(2) --> [two].
n10_19(10) --> [ten].
n10_19(11) --> [eleven].
n20_90(20) --> [twenty].
n20_90(30) --> [thirty].

?-phrase(numeral(2211), N).
N = [two, thousand, two, hundred, eleven]
The meaning of the **proper noun** ‘Socrates’ is **the term socrates**

\[
\text{proper\_noun}(\text{socrates}) \rightarrow [\text{socrates}].
\]

The meaning of the **property** ‘mortal’ is **a mapping from terms to literals containing the unary predicate mortal**

\[
\text{property}(X \rightarrow \text{mortal}(X)) \rightarrow [\text{mortal}].
\]

The meaning of a **proper noun - verb phrase sentence** is **a clause with empty body and head obtained by applying the meaning of the verb phrase to the meaning of the proper noun**

\[
\text{sentence}((L:-true)) \rightarrow \text{proper\_noun}(X),\text{verb\_phrase}(X \rightarrow L).
\]

?-\text{phrase}(\text{sentence}(C),[\text{socrates},\text{is},\text{mortal}].
\]

\[
C = (\text{mortal}(\text{socrates}):-true)
\]
A transitive verb is a **binary mapping** from a pair of terms to literals

\[
\text{transitive}_{\text{verb}}(Y \Rightarrow X \Rightarrow \text{likes}(X, Y)) \rightarrow [\text{likes}].
\]

A proper noun instantiates **one of the arguments**, returning a **unary mapping**

\[
\text{verb}_{\text{phrase}}(M) \rightarrow \text{transitive}_{\text{verb}}(Y \Rightarrow M), \text{proper}_{\text{noun}}(Y).
\]

Exercise 7.4
sentence((L:-true))  -->  proper_noun(X), verb_phrase(X => L).

sentence((H:-B))  -->  [every], noun(X => B), verb_phrase(X => H).

% NB. separate ‘determiner’ rule removed, see later

verb_phrase(M)  -->  [is], property(M).

property(M)  -->  [a], noun(M).

property(X => mortal(X))  -->  [mortal].

proper_noun(socrates)  -->  [socrates].

noun(X => human(X))  -->  [human].
?-phrase(sentence(C),S).

C = human(X):-human(X)  
S = [every,human,is,a,human];

C = mortal(X):-human(X)  
S = [every,human,is,mortal];

C = human(socrates):-true  
S = [socrates,is,a,human];

C = mortal(socrates):-true  
S = [socrates,is,mortal];
‘Determiner’ sentences have the form ‘every/some [noun] [verb-phrase]’ (NB. meanings of ‘some’ sentences require 2 clauses)

sentence(Cs) --> determiner(M1,M2,Cs),noun(M1),verb_phrase(M2).
determiner(X=>B, X=>H, [(H:-B)]) --> [every].
determiner(sk=>H1, sk=>H2, [(H1:-true), (H1:-true)] --> [some].

?-phrase(sentence(Cs), [D, human, is, mortal]).

D = every, Cs = [(mortal(X):-human(X))];
D = some, Cs = [(human(sk):-true), (mortal(sk):-true)]
question(Q) --> [who], [is], property(X => Q).

question(Q) --> [is], proper_noun(X), property(X => Q).

question(Q1, Q2) --> [is], [some], noun(sk => Q1),
property(sk => Q2).
handle_input(\texttt{Question,Rulebase}) :-
phrase(\texttt{question(Query,\textit{Question})}), \% question
prove_rb(\texttt{Query,Rulebase}),!, \% it can be solved
transform(\texttt{Query,Clauses}), \% transform to
phrase(\texttt{sentence(Clauses,Answer)}), \% answer
show_answer(\texttt{Answer}),
nl\_shell(\texttt{Rulebase}).