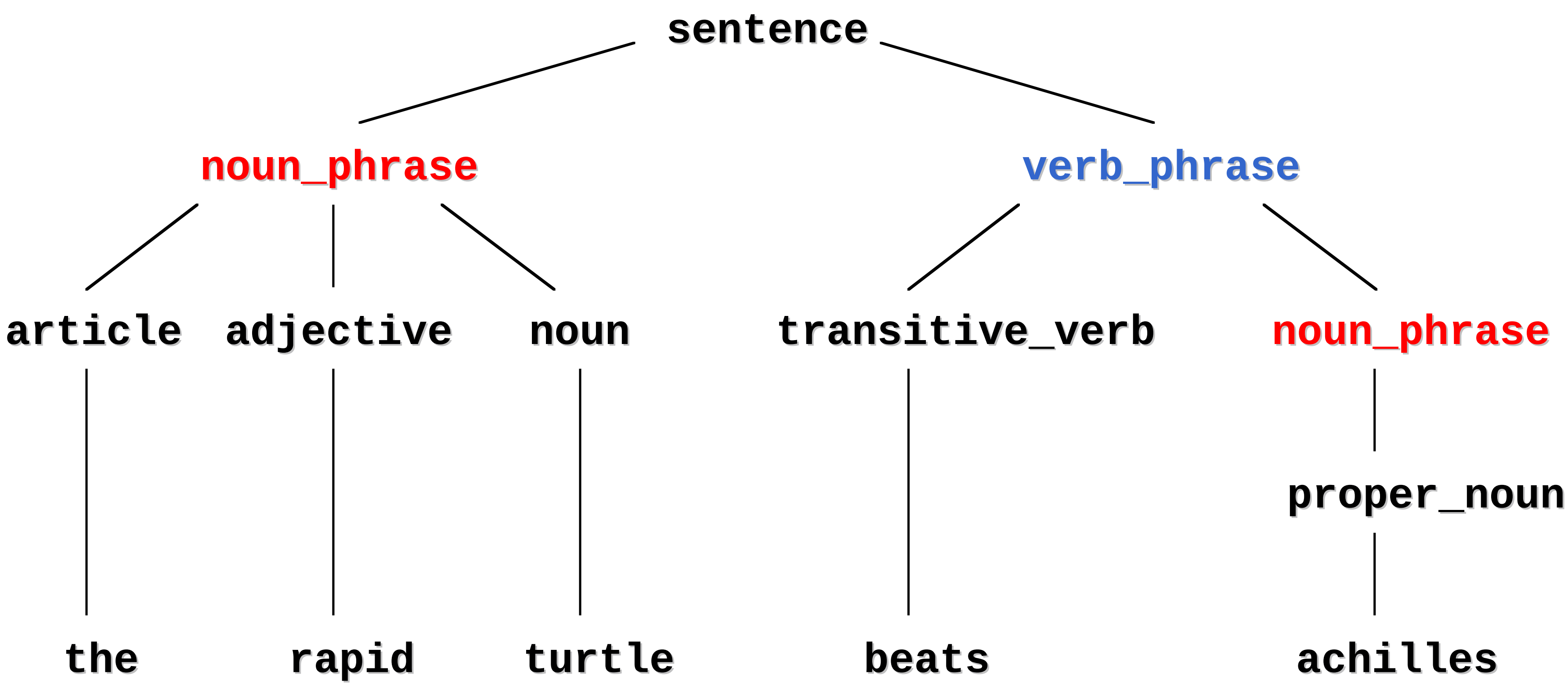


```
sentence            --> noun_phrase,verb_phrase.
noun_phrase         --> proper_noun.
noun_phrase         --> article,adjective,noun.
noun_phrase         --> article,noun.
verb_phrase         --> intransitive_verb.
verb_phrase         --> transitive_verb,noun_phrase.
article             --> [the].
adjective           --> [lazy].
adjective           --> [rapid].
proper_noun         --> [achilles].
noun                --> [turtle].
intransitive_verb --> [sleeps].
transitive_verb     --> [beats].
```

Context-free grammar

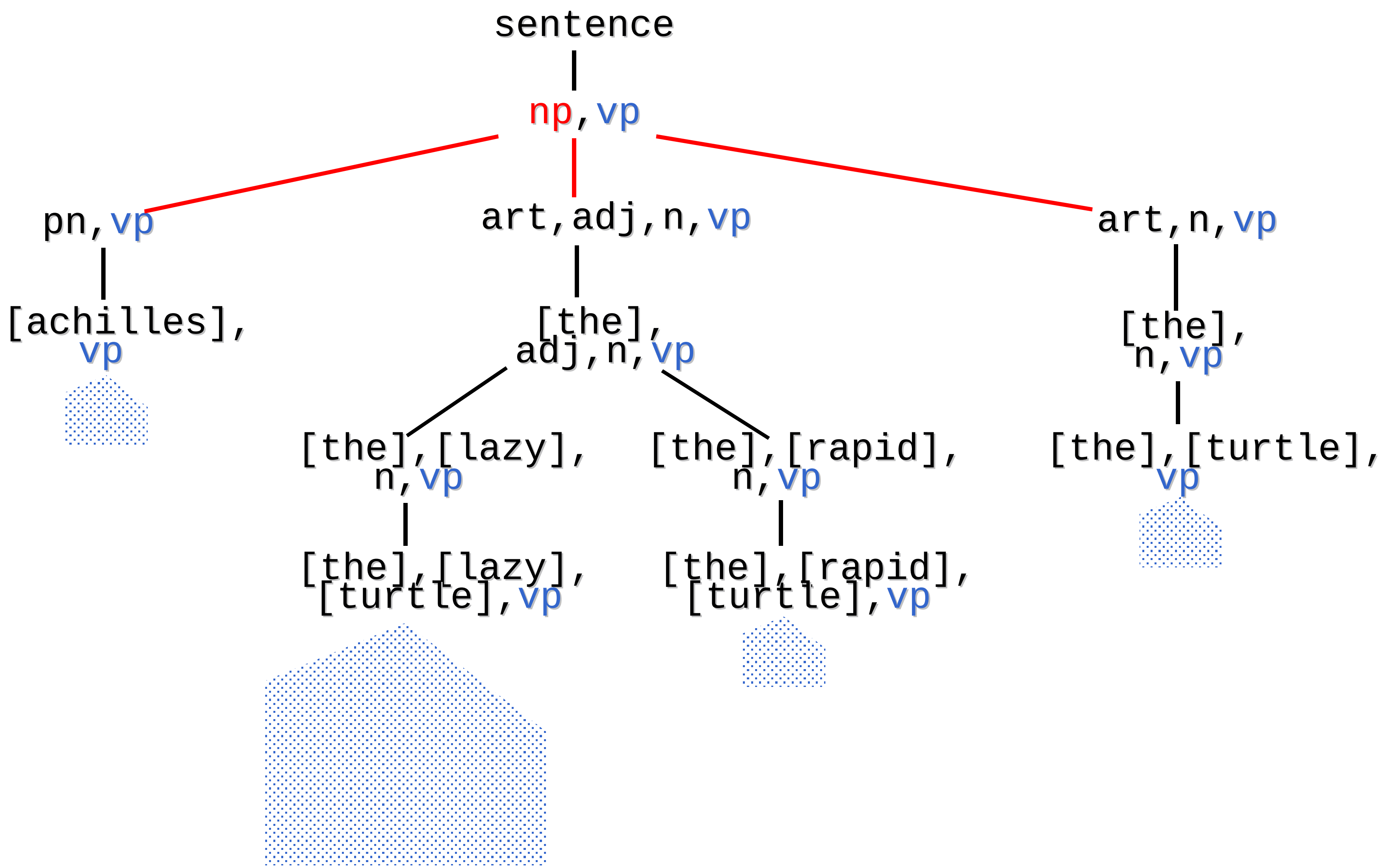

sentence
noun_phrase
verb_phrase
article
adjective
noun
transitive_verb
noun_phrase
proper_noun
the
rapid
turtle
beats
achilles

Parse tree

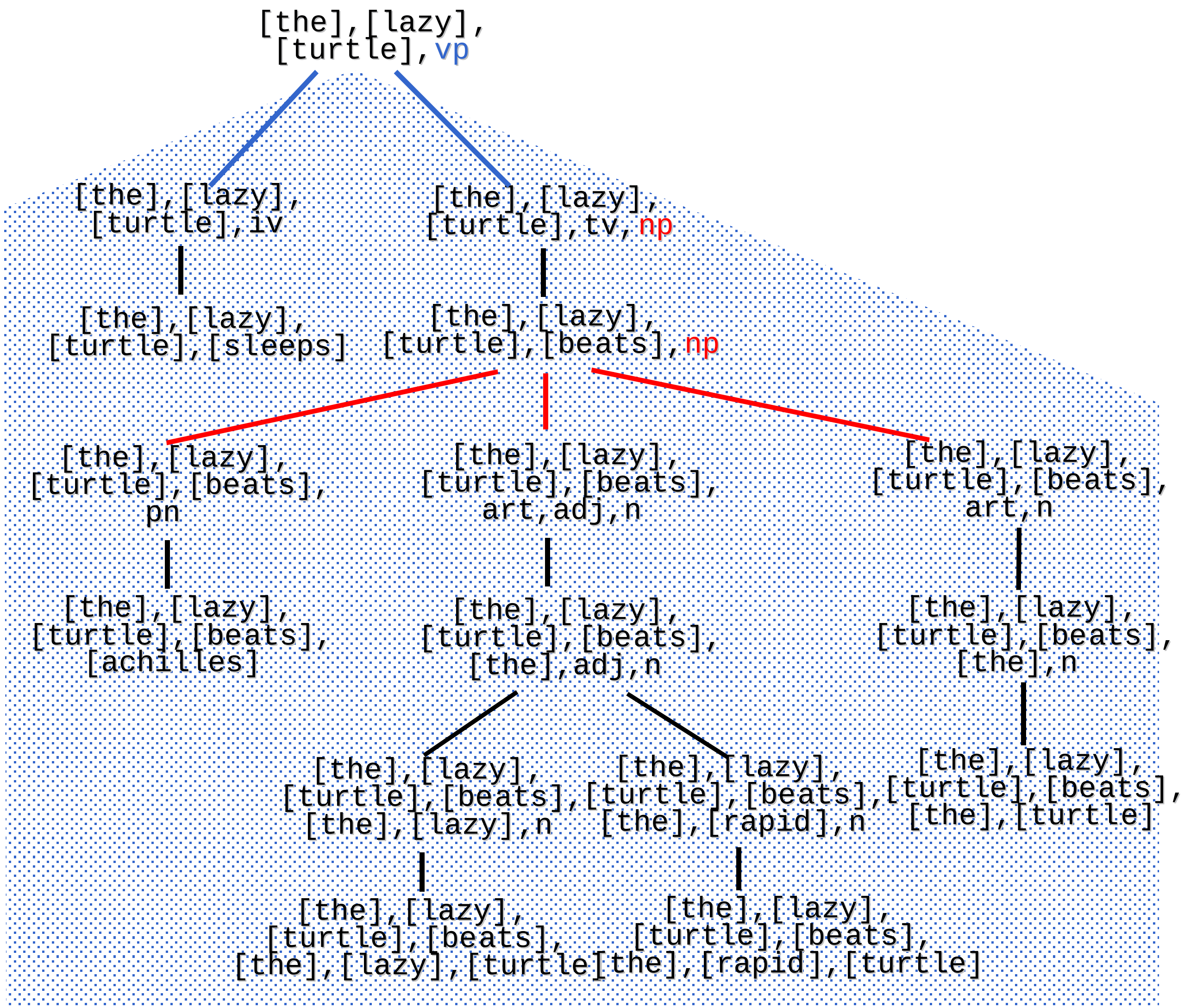
```
sentence
noun_phrase,verb_phrase
article,adjective,noun,verb_phrase
[the],adjective,noun,verb_phrase
[the],[rapid],noun,verb_phrase
[the],[rapid],[turtle],verb_phrase
[the],[rapid],[turtle],transitive_verb,noun_phrase
[the],[rapid],[turtle],[beats],noun_phrase
[the],[rapid],[turtle],[beats],proper_noun
[the],[rapid],[turtle],[beats],[achilles]
```

```
sentence --> noun_phrase,
             verb_phrase
noun_phrase --> article,
                adjective,
                noun
article --> [the]
adjective --> [rapid]
noun --> [turtle]
verb_phrase --> transitive_verb,
                noun_phrase
transitive_verb --> [beats]
noun_phrase --> proper_noun
proper_noun --> [achilles]
```

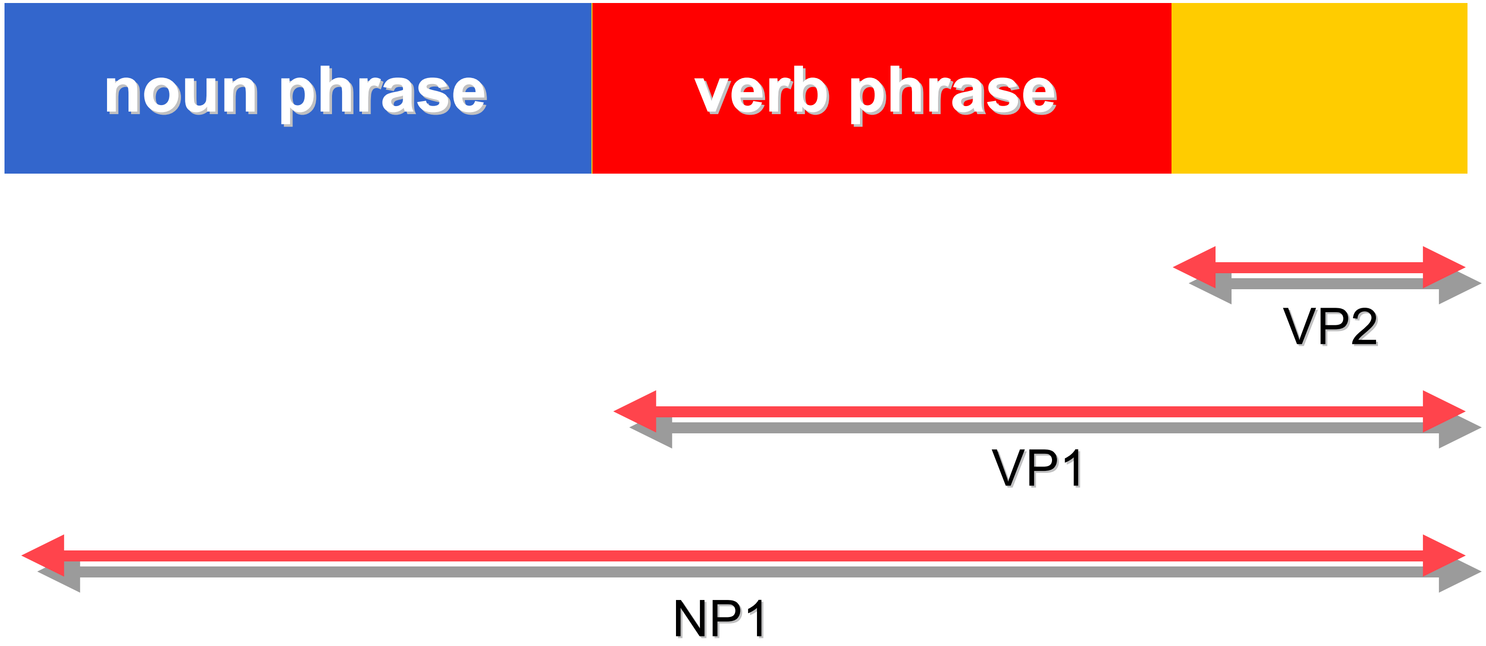
Exercise 7.1



Exercise 7.2 (1)



Exercise 7.2 (2)




```
sentence(NP1-VP2):-
        noun_phrase(NP1-VP1),
        verb_phrase(VP1-VP2)
```

Difference lists in grammar rules

	GRAMMAR	PARSING
META-LEVEL	`s --> np,vp`	`?-phrase(s,L)`

OBJECT-LEVEL	`s(L,L0):-`
	`np(L,L1),`
	`vp(L1,L0)`
		`?-s(L,[])`

Meta-level vs. object-level


```
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(pluralr)  --> [sleep].
```

Non-terminals with arguments

```
sentence(s(NP,VP))        --> noun_phrase(NP),verb_phrase(VP).
noun_phrase(np(N))        --> proper_noun(N).
noun_phrase(np(Art,Adj,N)) --> article(Art),adjective(Adj),
                               noun(N).
noun_phrase(np(Art,N))    --> article(Art),noun(N).
verb_phrase(vp(IV))       --> intransitive_verb(IV).
verb_phrase(vp(TV,NP))    --> transitive_verb(TV),
                              noun_phrase(NP).
article(art(the))         --> [the].
adjective(adj(lazy))      --> [lazy].
adjective(adj(rapid))     --> [rapid].
proper_noun(pn(achilles))  --> [achilles].
noun(n(turtle))           --> [turtle].
intransitive_verb(iv(sleeps)) --> [sleeps].
transitive_verb(tv(beats)) --> [beats].
```

```
?-phrase(sentence(T),[achilles,beats,the,lazy,turtle])
  T = s(np(pn(achilles)),
        vp(tv(beats),
           np(art(the),
              adj(lazy),
              n(turtle))))
```

Constructing parse trees


```
numeral(N)     --> n1_999(N).
numeralN)      --> n1_9(N1),[thousand],n1_999(N2),
                   {N is N1*1000+N2}.
n1_999(N)      --> n1_99(N).
n1_999(N)      --> n1_9(N1),[hundred],n1_99(N2),
                   {N is N1*100+N2}.
n1_99(N)       --> n0_9(N).
n1_99(N)       --> n10_19(N).
n1_99(N)       --> n20_90(N).
n1_99(N)       --> n20_90(N1),n1_9(N2),{N is N1+N2}.
n0_9(0)        --> [].
n0_9(N)        --> n1_9(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]
```

Prolog goals in grammar rules

- The meaning of the **proper noun 'Socrates'** is **the term `socrates`**

```
proper_noun(socrates)--> [socrates].
```

- The meaning of the **property 'mortal'** is **a mapping from terms to literals containing the unary predicate `mortal`**

```
property(X=>mortal(X))     --> [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**

```
sentence((L:-true)) --> proper_noun(X),verb_phrase(X=>L).
?-phrase(sentence(C),[socrates,is,mortal].
C = (mortal(socrates):-true)
```

Interpretation

- A transitive verb is a binary mapping from a pair of terms to literals

```
transitive_verb(Y=>X=>likes(X,Y)) --> [likes].
```

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

```
verb_phrase(M) --> transitive_verb(Y=>M),proper_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].
```

Interpretation (2)


```
?-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];
```

Interpretation (3)

- ‘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)]
```

Determiners


```
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).
```

Questions

```
handle_input(Question,Rulebase):-
phrase(question(Query),Question),    % question
prove_rb(Query,Rulebase),!,          % it can be solved
transform(Query,Clauses),            % transform to
phrase(sentence(Clauses),Answer),    % answer
show_answer(Answer),
nl_shell(Rulebase).
```

Querying a rulebase