CS4025: Grammars

- Grammars for English
- Features
- Definite Clause Grammars

See J&M Chapter 9 in 1st ed, 12 in 2nd, Prolog textbooks (for DCGs)

Grammar: Definition

The surface structure (syntax) of sentences is usually described by some kind of *grammar*.

- A grammar defines syntactically legal sentences.
  - » John ate an apple (syn legal)
  - » John ate apple (not syn legal)
  - » John ate a building (syn legal)
- More importantly, a grammar provides a description of the structure of a legal sentence

Facts about Grammars

- Nobody has ever constructed a complete and correct grammar for any natural language
- There is no single “correct” grammar for any natural language (though certain terminology has emerged that is generally useful)
- Linguists even dispute exactly what formalism should be used for stating a grammar
- Grammars describe subsets of NLs. They all have their strengths and weaknesses.

Sentence Constituents

- Grammar rules mention *constituents* (e.g. NP – noun phrase) as well as single word types (noun)
- A constituent can occur in many rules
  - » NP also after a preposition (*near the dog*), in possessives (*the cat's paws),...*
- These capture to some extent ideas such as:
  - » Where NP occurs in a rule, any phrase of this class can appear
  - » NP's have something in common in terms of their meaning
  - » Two phrases of the same kind, e.g. NP, can be joined together by the word "and" to yield a larger phrase of the same kind.

Common Constituents

- Some common constituents
  - » Noun phrase: *the dog, an apple, the man I saw yesterday*
  - » Adjective phrase: *happy, very happy*
  - » Verb group: *eat, ate, am eating, have been eating.*
  - » Verb phrase: *eating an apple, ran into the store*
  - » Prepositional phrase*: in the car*
- Constituents named "X phrase" in general have a most important word, the ***head***, of category X (underlined above)

Phrase-Structure Grammar

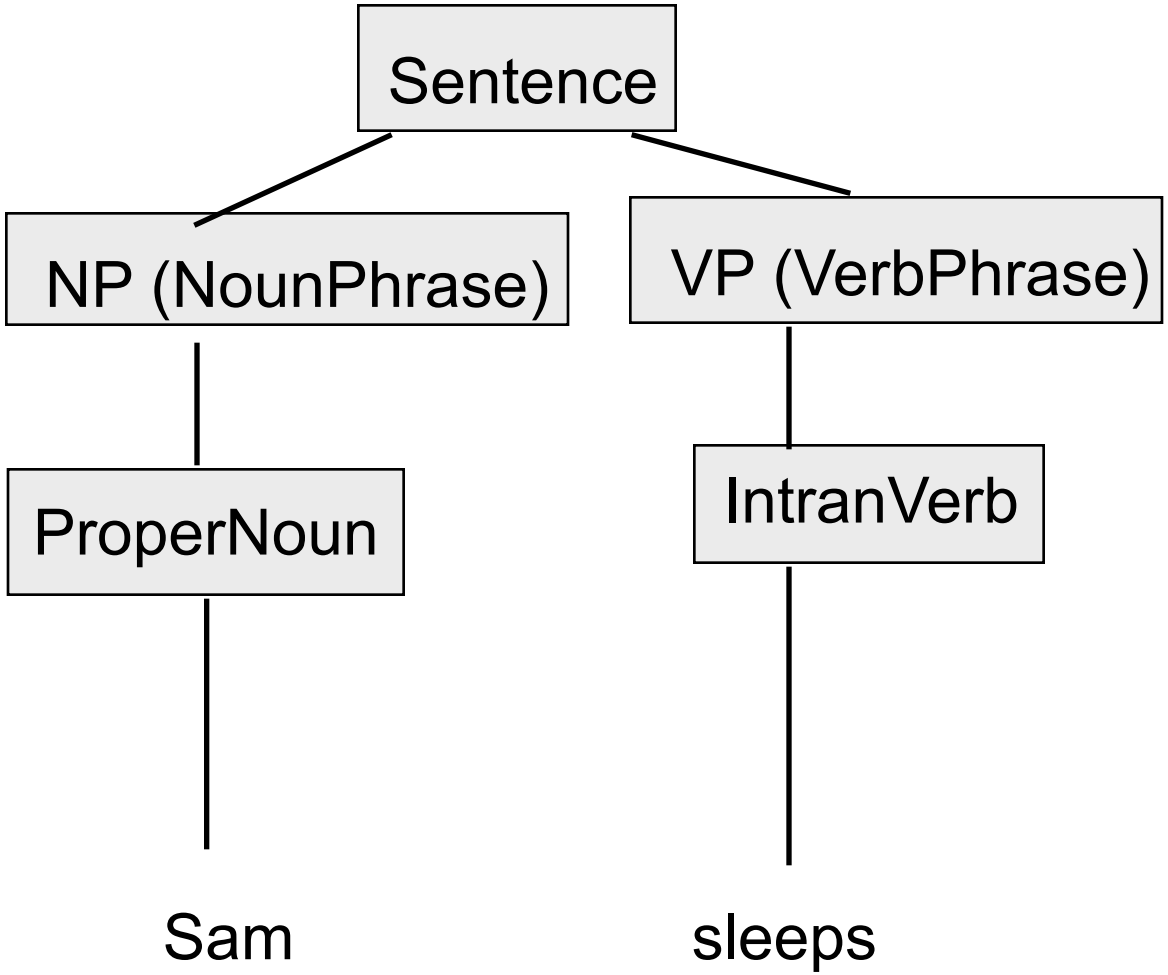
- PS grammars use context-free rules to define constituents and sentences
  - » S = NP Verb NP
- Similar to rules used to define syntax of programming languages.
  - » if-statement = **if** condition **then** statement
- Also called context-free grammar (CFG)
- Adequate for (almost all of?) English
  - » see J&M, chap 13

Another simple grammar

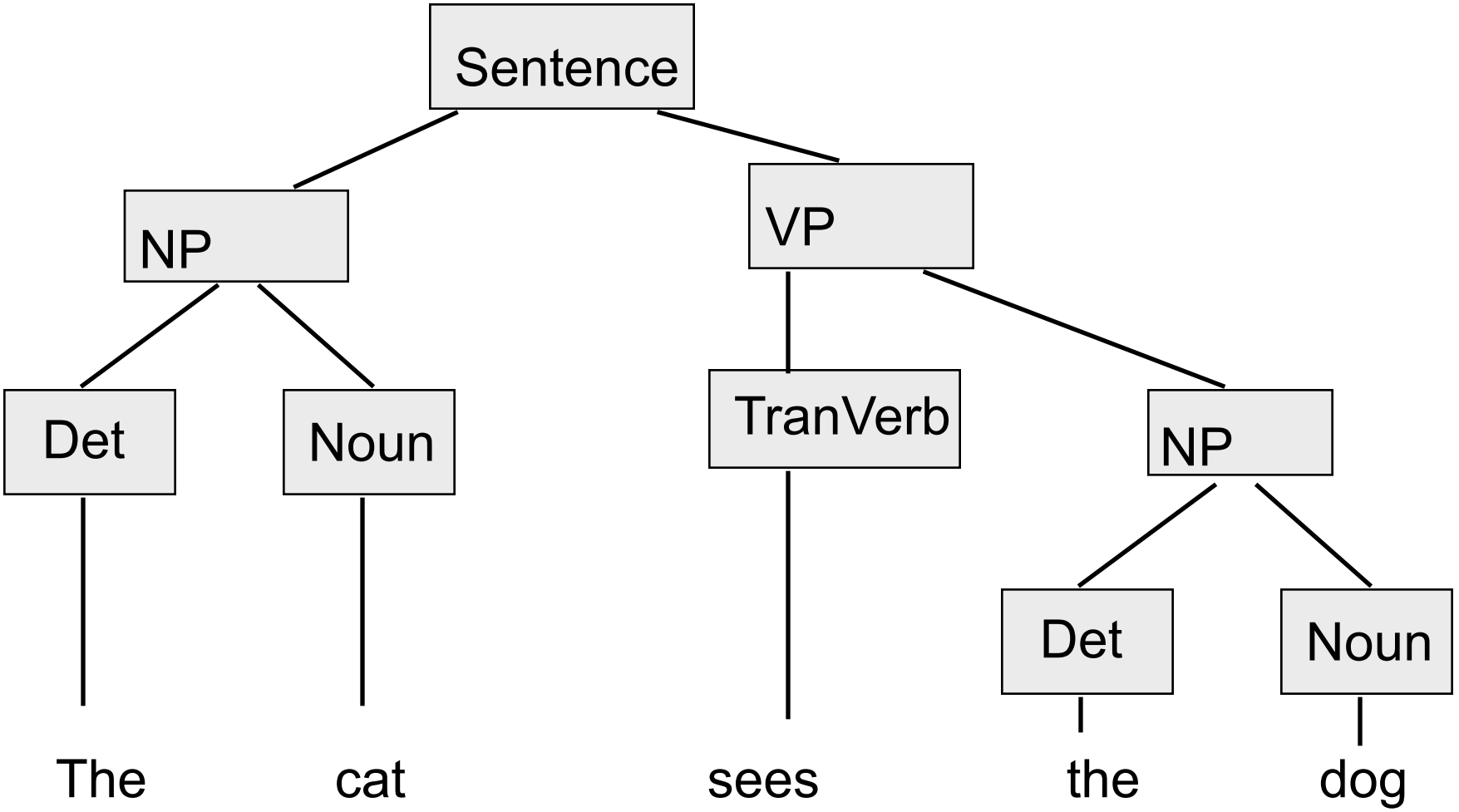
- S = NP VP
- NP = Det Adj* Noun
- NP = ProperNoun
- VP = IntranVerb
- VP = TranVerb NP
- Det: *a , the*
- Adj: *big , black*
- Noun: *dog , cat*
- ProperNoun: *Fido , Misty, Sam*
- IntranVerb: *runs, sleeps*
- TranVerb: *chases, sees, saw*

How does this assign structure to simple sentences?

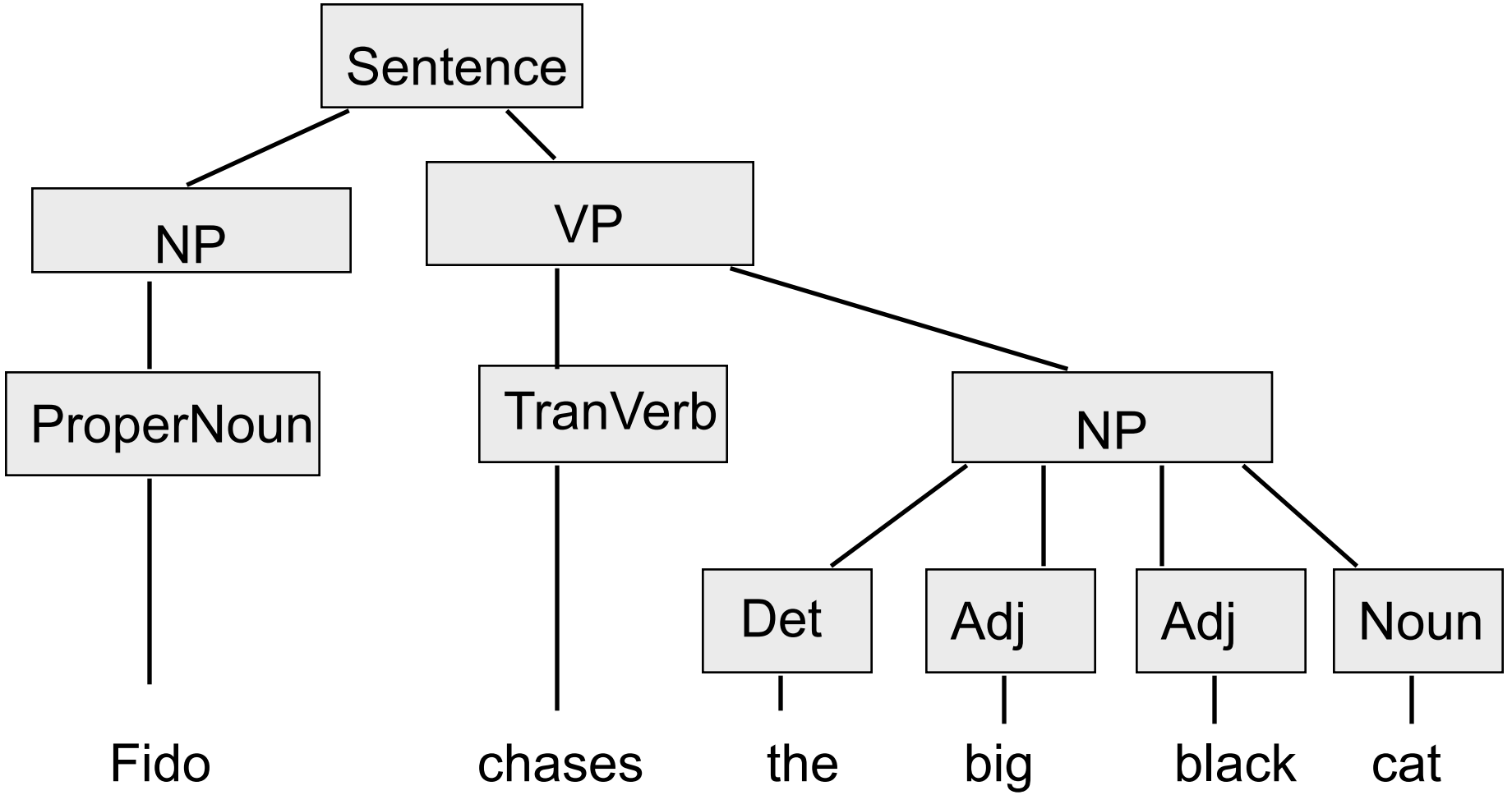
Example: Sam sleeps



Ex: The cat sees the dog



Fido chases the big black cat



Recursion

- NL Grammars allow recursion

  NP = Det Adj* CommonNoun PP*

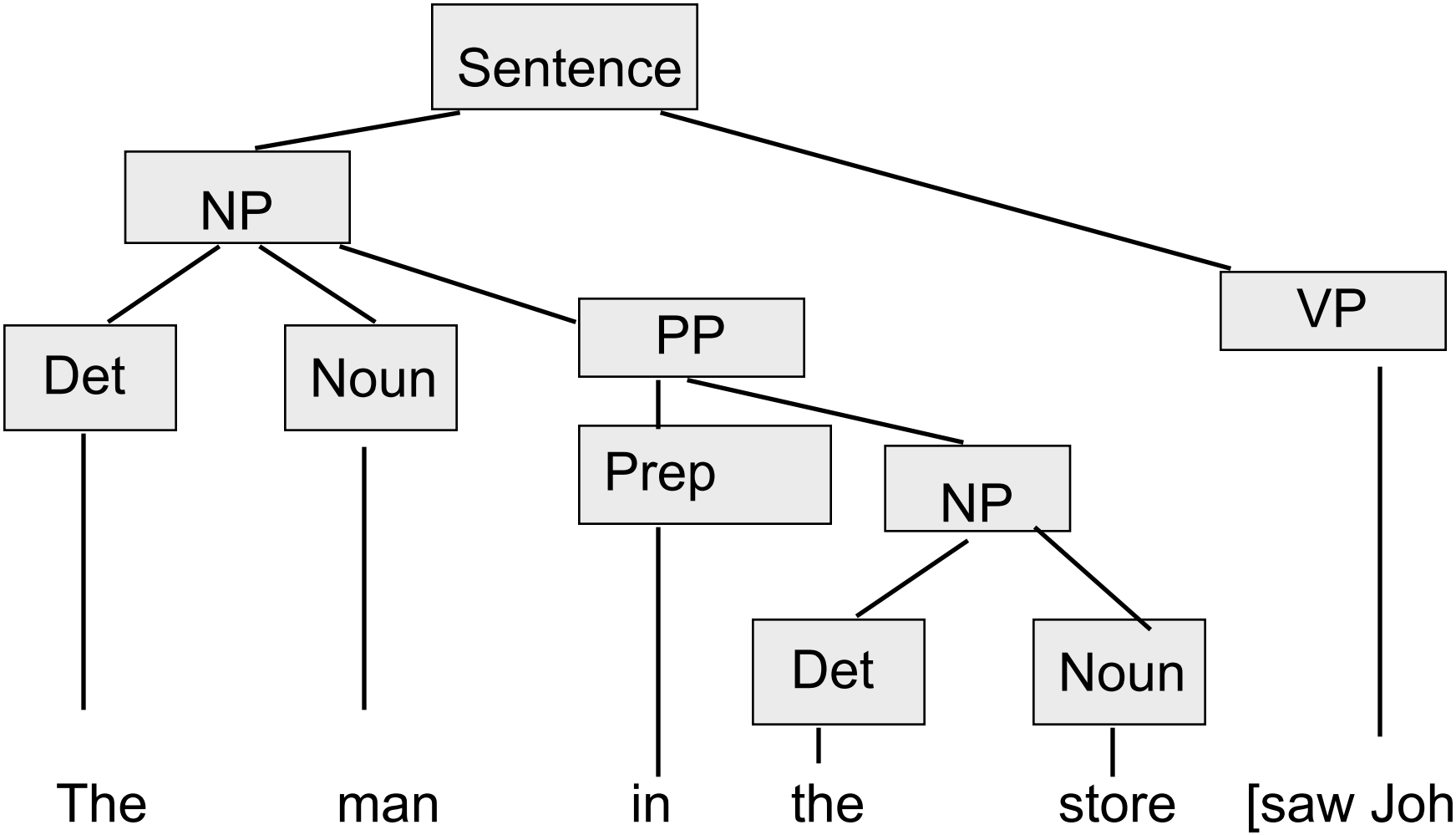
  PP = Prep NP

- For example:

  The man in the store saw John

  I fixed the intake of the engine of BA's first jet.

The man in the store saw John



Grammar is not everything

- Sentences may be grammatically OK but not acceptable (acceptability?)
  - » The sleepy table eats the green idea.
  - » Fido chases the black big cat.
- Sentences may be understandable even if grammatically flawed.
  - » Fido chases a cat small.

BUT:

- Grammar terminology may be useful for describing sentences even if they are flawed (e.g. in tutoring systems)
- Grammars may be useful for analysing fragments within unrestricted and badly-formed text.

Features

- Features are annotations on words and consituents
- Grammar rules use unification (as in Prolog) to manipulate features.
- Allow grammars to be smaller and simpler
- See J&M, chap 11

Ex: Number

- One feature is [number:N], where N is *singular* or *plural*.
  - » A noun's number comes from morphology
    - – *cat* - singular noun
    - – *cats* - plural noun
  - » An NP's number comes from its head noun
  - » A VP's number comes from its verb (aux verb?)
    - – *is happy* - singular VP
    - – *are happy* - plural VP
  - » NP and VP must agree in number:
    - – The rat has an injury
    - – *The rat have an injury

Modified Grammar

- S = NP[num:N] VP[num:N]
- NP[num:N] = Det[num:N] Adj* Noun[num:N] PP*
- NP[num:singular] = ProperNoun
- VP[num:N] = IntranVerb[num:N]
- VP[num:N] = TranVerb[num:N] NP
- PP = Prep NP

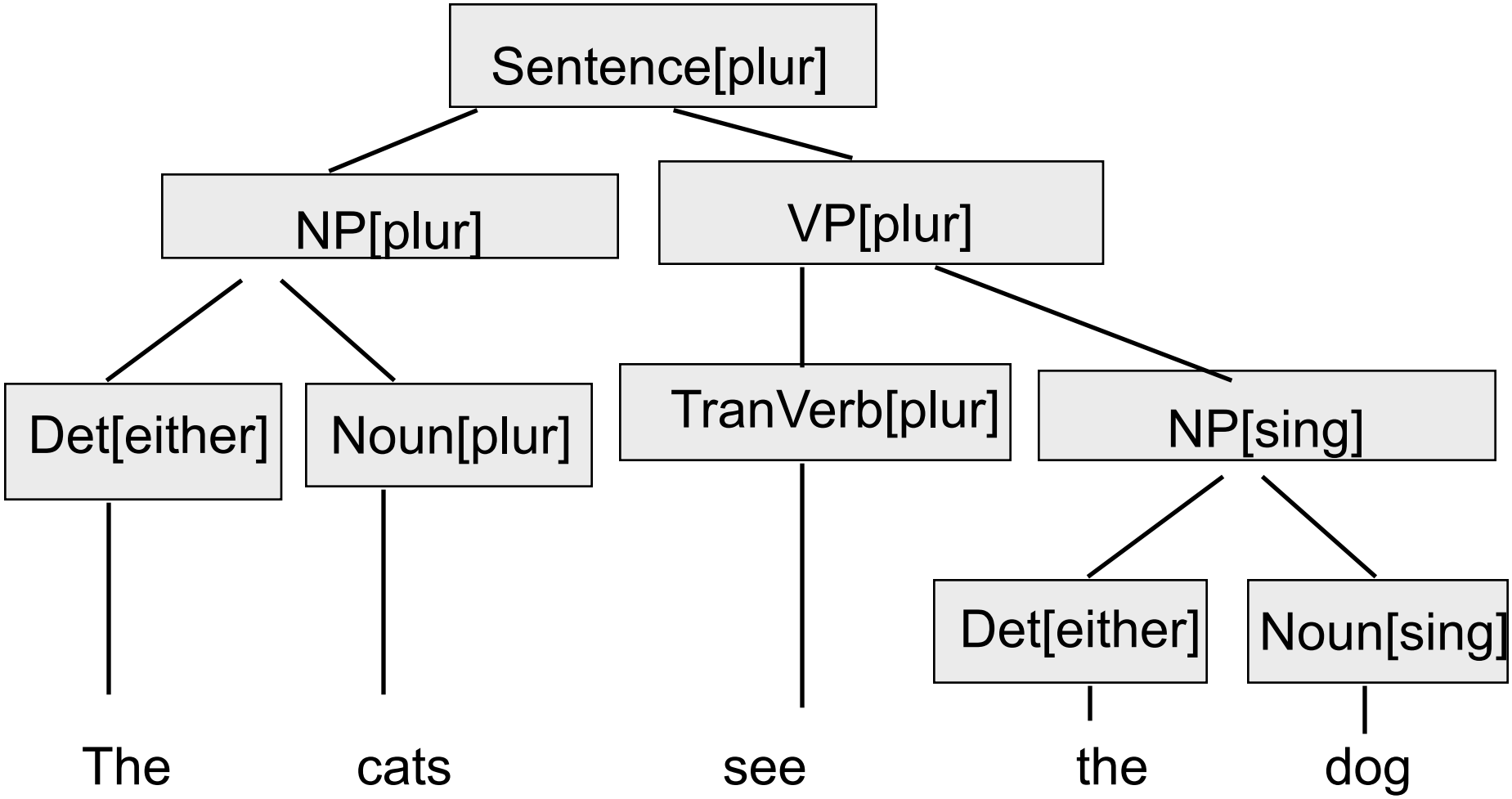
- Noun[num:sing] = dog
- Noun[num:plur] = dogs etc.

Example rule

NP[num:N] = Det[num:N] Adj* Noun[num:N]

- An NP's determiner and noun (but not its adjectives) must agree in number.
  - *this dog* is OK
  - *these dogs* is OK
  - *this dogs* is not OK (det must agree)
  - *these dog* is not OK (det must agree)
  - *the big red dog* is OK
  - *the big red dogs* is OK (adjs don't agree)
- [number of NP] is [number of Noun]

Ex: The cats see the dog



Other features

- Person (I, you, him)
- Verb form (past, present, base, past participle, pres participle)
- Gender (masc, fem, neuter)
- Polarity (positive, negative)
- etc, etc, etc

Definite Clause Grammars

- A definite clause grammar (DCG) is a CFG with added feature information
- The syntax is also Prolog-like, and DCGs are usually implemented in Prolog
- You don’t need to know Prolog to use DCGs
- The formalism was (partially) invented in Scotland

DCG Notation

Rule notation	*DCG notation*
S = NP VP	s ---> np, vp.
S[num:N] = NP[num:N] VP[num:N]	s(Num) ---> np(Num), vp(Num).
N = *dog*	n ---> [dog].
N = *New York*	n ---> [new, york].
NP = <name>	np ---> [X], {isname(X)}.

NB the number of "-" in the "--->" varies according to the implementation

DCG Notation (2)

- Use ---> between rule LHS and RHS.
- Non-terminal symbols (eg, np) written as Prolog constants (start with lower case letters).
- Features (eg, Num) are arguments in ()
  - » Features are shown by position, not name
  - » Prolog variables (start with upper case letters) indicate unknown values
- Actual words are lists, in []
- Prolog tests can be attached in {}
- (Optional) use *distinguished* predicate to define the goal symbol (eg, s).

An example DCG

```
distinguished(s).
s ---> np(Num), vp(Num).
np(Num) ---> det(Num), noun(Num).
np(sing) ---> name.
vp(Num) ---> intranverb(Num).
vp(Num) ---> tranverb(Num), np(_).
det(_) ---> [the].
noun(sing) ---> [dog].
name ---> [sam].
intranverb(sing) ---> [sleeps].
tranverb(plur) ---> [chase].
```

DCG's and Prolog

- DCG's are translated into Prolog in a fairly straightforward fashion
  - » See Shapiro, *The Art of Prolog*
- In fact, Prolog was originally designed as a language for NLP, as part of a machine-translation project.
- Prolog has evolved since, but is still well-suited to many NLP operations.

Edinburgh Package

- We will use a parsing system for DCGs developed (for teaching purposes) by Holt and Ritchie from Edinburgh University.
- Use SWI Prolog

Using the package

- Statements always end with “.”
- Prompt is ?-
- Type code into a file, not directly into the interpreter.
- Load files with [file].
  - .pl = prolog source
- By default, type “n.” to backtrack request

Using the DCG package

?- prp([sam, chases, the, dog *parse and pretty print*

?- prp(np, [the, dog]). *parse and print an np*

?- parse([sam, chases, the, dog], X). *returns parse in X*

?- parse(np, [the, dog], X). *parse a constituent (np)*

For all the above, after each parse is printed, you will be asked if you want to look for additional parses

NB for other DCG implementations, the grammar writer explicitly builds parse structures in the grammar rules.

Other Grammar Formalisms

- Many other grammar formalisms
  - » GPSG, HPSG, LFG, TAG, ATN, ...
- No consensus on which is best, although many strong opinions (like C vs Pascal vs Lisp)
- Like programming languages, formalism used often determined by experience of developers, and available support tools