From first order terms to feature structures

- data structures:
  - compound terms (DCGs)
  - term unification
  - feature structures (PATR)
  - representing feature structures in Prolog
  - feature structure unification
- using feature structures:
  - grammar side: implementing a grammar in a typed feature structure based system (ALE) → project
  - algorithmic side: implementing a feature based parsing system in Prolog

Representation of feature descriptions in Prolog

- Feature names & atomic values represented by Prolog atoms

  `cat`, `verb`, `agr`, `num`, `sing`

- Paths built up of features separated by #

  `agr#num`, `subj#agr#num`

- Paths and values are separated by ===

  `cat===verb`, `agr#num===sing`,

- Path equality:

  `agr#num===subj#agr#num`

- Alternatively, it can be represented as structure sharing using a variable as value:

  ```
  agr#num===X
  ⋮
  subj#agr#num===X
  ```


Representation of feature structures in Prolog

- Feature names & atomic values represented by Prolog atoms:

  `cat`, `verb`, `agr`, `num`, `sing`

- Feature structures are represented as `feature-value` pairs, which are elements of a Prolog list with an open tail:

  ```
  [cat-verb|_]
  [cat-verb,agr-[num-sing|_]|_]
  ```

- Path equality (structure sharing) represented by variables:

  ```
  [agr-[num-X|_],subj-[agr-[num-X|_]|_]|_]
  ```

Grammar rules in a PATR-like system

```
:- op(500,xfy,-).
:- op(500,xfy,#).
:- op(500,xfy,:).

W ---> mary :-
   W:cat     === np,
   W:agr#per === third,
   W:agr#num === sing.

W ---> left :-
   W:cat     === vp,
   W:agr#per === third,
   W:agr#num === sing.
```

```
:- op(500,xfx,--->).
:- op(600,xfy,===).

S ---> [NP,VP] :-
   S:cat   === s,
   NP:cat  === np,
   VP:cat  === vp,
   NP:agr  === VP:agr.
```


Using “===” to map from descriptions to FSs

```
X === Y :-
  denotes(X,Z),
  denotes(Y,Z).

denotes(Var,Var) :-
   var(Var),!.
denotes(Atom,Atom) :-
   atomic(Atom),!.
denotes(FS:Path,Value) :-
   pathval(FS,Path,Value,_).
```


Mapping from descriptions to FSs

```
pathval(FS1,Feat#Path,Value,FSs) :-
   !,pathval(FS1,Feat,FS2,FSs),
   pathval(FS2,Path,Value,_).

pathval([Feat-Val1|Rest],Feat,Val2,Rest) :-
   !, fs_unify(Val1,Val2).

pathval([FS|Rest],Feat,Val,[FS|Rest2]) :-
   pathval(Rest,Feat,Val,Rest2).
```



```
fs_unify(FS,FS) :- !.
fs_unify([Path-Val|Rest1],FS) :-
   pathval(FS,Path,Val,Rest2),
   fs_unify(Rest1,Rest2).
```

A left-corner parser for CFG grammars in CNF

```
% ?- recognise(s,<list(word)>,[]).

recognise(Mother) --> Word,
                      {Cat ---> [Word]},
                      lc(Cat,Mother).

lc(Phrase,Phrase) --> [].

lc(SubPhrase,SuperPhrase) -->
    {Phrase ---> [SubPhrase,Right]},
    recognise(Right),
    lc(Phrase,SuperPhrase).
```


A left-corner parser for PATR grammars in CNF

```
recognise(MotherFs) --> [Word],
                        {CatFs ---> Word},
                        lc(CatFs,MotherFs).

lc(Fs1,Fs2) --> [],
                {fs_unify(Fs1,Fs2)}.

lc(SubPhraseFs,SuperPhraseFs) -->
    {PhraseFs ---> [SubPhraseFs,RightFs]},
    recognise(RightFs),
    lc(PhraseFs,SuperPhraseFs).
```