On the use of complex data structures

- What do non-atomic data structures represent?
- Combining non-atomic data structures
- Term unification
- Representing feature structures
- Feature structure unification

What do non-atomic data structures represent?

```
s --> np(Per,Num), vp(Per,Num).
```

The rule represents a set of ground instances, one for each possible **substitution** of a variable with a value.

```
s --> np(first,sing),  vp(first,sing).
s --> np(second,sing), vp(second,sing).
s --> np(third,sing),  vp(third,sing).
s --> np(first,plur),  vp(first,plur).
s --> np(second,plur), vp(second,plur).
s --> np(third,plur),  vp(third,plur).
```


Combining compound terms

```
s --> np(Per,Num), vp(Per,Num).

np(third,sing) --> [he].
np(third,plur) --> [they].

vp(third,sing) --> [walks].
vp(third,plur) --> [walk].
```

Two possible substitutions:
`third/Per, sing/Num` or `third/Per, plur/Num`

Most general unifiers

```
termUnify(f(X,h(Y,e),g(d(Z),e)),
          f(Y,h(d(Z),e),g(Y,e))).
```

Which substitution should one report?

- `d(a)/X` or `d(a)/X`
- `d(b)/X` or `d(b)/X`
- `d(h(a,b))/X` and `d(h(a,b))/X`

Pick the most general unifier (mgu):

- `d(Z)/X` and `d(Z)/X`

Infinite trees

- What happens when one unifies
  - `f(X)` with `X` ?
  - `f(a,g(X,b))` with `X` ?
- Add an **occurs check** to test whether the variable occurs in the term.
- In practice too costly.

Term unification

- A variable unifies with any term it does not occur in.
- An atom (functor or constant) unifies only with an identical atom.
- Compound terms unify if their functors are identical and their arguments unify pairwise, and the substitutions obtained as a result of each of these unifications are compatible.

Explicit term unification in Prolog

```
termUnify(X,Y) :-
   var(X),!,
   X = Y.

termUnify(X,Y) :-
   var(Y),!,
   X = Y.
```



```
termUnify(X,Y) :-
   X =.. [XFunctor|XArgs],
   Y =.. [YFunctor|YArgs],
   XFunctor=YFunctor,
   unifyArgs(XArgs,YArgs).

unifyArgs([],[]).
unifyArgs([X|Xs],[Y|Ys]) :-
   termUnify(X,Y),
   unifyArgs(Xs,Ys).
```

Representing feature structures in Prolog

- Feature names and atomic values represented by Prolog atoms
- The operator ":" is defined to separate `feature:value`
  `?- op(500,xfy,:).`
- Prolog variables encode structure sharing.
- Feature structures represented as Prolog lists with open tails:
  - `[person:third, num:sing|_]`
  - `[person:X, num:sing, head:subj:person:X|_]`

Unifying feature structures in Prolog

```
unify(Dag,Dag) :- !.

unify([Path:Val | Rest1], Dag) :-
   pathval(Dag,Path,Val,Rest2),
   unify(Rest1,Rest2)

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

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


Towards grammar rules in PATR

```
rule(S,[NP,VP]) :-
  pathval(S,cat,s,_),
  pathval(NP,cat,np,_),
  pathval(VP,cat,vp,_),
  pathval(NP,per,X,_),
  pathval(VP,per,X,_),
  pathval(NP,num,Y,_),
  pathval(VP,num,Y,_).
```


Grammar rules in PATR

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

S ---> [NP,VP]) :-
  S:cat  === s,
  NP:cat === np,
  VP:cat === vp,
  NP:per === X,
  VP:per === X,
  NP:num === Y,
  VP:num === Y.
```

Assigning a general meaning to “===”

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

denotes(Var,Var) :-
   var(Var),!.

denotes(Atom,Atom) :-
   atomic(Atom),!.
```



```
denotes(Dag:Path,Value) :-
   pathval(Dag,Path,Value).

pathval(Dag1,Feature:Path,Value,Dags) :-
   !,pathval(Dag1,Feature,Dag2,Dags).
   pathval(Dag2,Path,Value).
```


Two notes on ALE

- Testing for unifiability: `mgsat/1`
- Stepwise addition of edges to chart: `interp/0`