Chart parsing with non-atomic categories

Detmar Meurers: Intro to Computational Linguistics I
OSU, LING 684.01, 10. March 2003

Overview

Three options for chart parsing with grammars employing non-atomic categories:

1. Expand the grammar into a CFG with atomic categories

2. Parse using an atomic CFG backbone with reduced information

3. Incorporate special mechanisms into the parser

Idea 1: Expand the grammar into a CFG with atomic categories

- number of categories grows exponentially, e.g., 3^n is size of category set with n binary features (plus, minus, unspecified)
- leads to a potentially huge set of rules
- grammar size relevant for time and space efficiency of parsing

Idea 2: Parse using an atomic CFG backbone with reduced information

- idea:
  - parse using a property defined for all categories
  - use other properties to filter solutions from set of parses
- downside:
  - parsing with partial information can significantly enlarge the search space

Idea 3: Incorporate special mechanism into parser

- Equality check for atomic categories has to be replaced by **unification**.
- Every active and inactive edge in a chart may be used for different uses. So for each time an edge is used, a new **copy** needs to be made.
- Two effectiveness issues:
  - Use **subsumption** test to ensure general enough predictions
  - Using **restriction** to prevent prediction loops
- Two efficiency issues (not dealt with here):
  - intelligent **indexing** of edges in chart
  - **packing** of similar edges in chart (cf., Tomita parser)

Earley parser with atomic categories

Prediction:

for each ${}_i[A \rightarrow \alpha \bullet_j B \beta]$ in chart
  for each $B \rightarrow \gamma$ in rules
    add ${}_j[B \rightarrow \bullet_j \gamma]$ to chart

Scanning:

let $w_1 \ldots w_j \ldots w_n$ be the input string
for each ${}_i[A \rightarrow \alpha \bullet_{j-1} w_j \beta]$ in chart
  add ${}_i[A \rightarrow \alpha w_j \bullet_j \beta]$ to chart

Completion (fundamental rule of chart parsing):

for each ${}_i[A \rightarrow \alpha \bullet_k B \beta]$ and ${}_k[B \rightarrow \gamma \bullet_j]$ in chart
  add ${}_i[A \rightarrow \alpha B \bullet_j \beta]$ to chart

Earley parser with unification

Prediction:

for each ${}_i[A \rightarrow \alpha \bullet_j B\ \beta]$ in chart
  for each $B' \rightarrow \gamma$ in rules
    add ${}_j[\sigma(B \rightarrow \bullet_j \gamma)]$ with $\sigma = \text{mgu}(B, B')$ to chart

Completion (fundamental rule of chart parsing):

for each ${}_i[A \rightarrow \alpha \bullet_k B\ \beta]$ and ${}_k[B' \rightarrow \gamma \bullet_j]$ in chart
  add ${}_i[\sigma(A \rightarrow \alpha\ B \bullet_j \beta)]$ with $\sigma = \text{mgu}(B, B')$ to chart

Earley algorithm with lexical lookup marked up with unification for complex categories

```
:- dynamic chart/3.          % chart(From,To,state(Lhs,Rest_Rhs))
:- op(1200,xfx,'--->').      % operator for grammar rules


% recognize(+WordList,+Startsymbol): Earley recognizer toplevel

recognize(String,Startsymbol) :-
   retractall(chart(_,_,_)),
   enter_edge(0,0,state('S',[Startsymbol])),
   scan(String,0,N),
   chart(0,N,state('S',[])).
```

```
% enter_edge(+FromIndex,+ToIndex,+Contents)

% a) only add if it does not yet exist:
enter_edge(I,J,State) :-
   chart(I,J,State),
   !.

% b) add to chart and make try prediction/completion
enter_edge(I,J,State) :-
   assertz(chart(I,J,State)),
   predict(I,J,State),
   complete(I,J,State).
```

```
predict(_,J,State) :-
   State = state(_,[B|_]),      % active edge
   (B1 ---> Gamma),
   unify_terms(B,B1),
   enter_edge(J,J,state(B,Gamma)),
   fail
 ; true.

% ------------------------------------------------

complete(K,J,State) :-
   State = state(B,[]),         % passive edge
   chart(I,K,state(A,[B1|Beta])),
   unify_terms(B,B1),
   enter_edge(I,J,state(A,Beta)),
   fail
 ; true.
```

```
% Unification of first order terms can be handled by Prolog
% For feature structures and other data structures, a
% unification predicate would need to be defined here.

unify_terms(X,X).


scan([],N,N).
scan([W|Ws],JminOne,N) :-
   J is JminOne+1,
   ( lex(Cat,W),
     enter_edge(JminOne,J,state(Cat,[])),
     fail
   ; scan(Ws,J,N)).
```

The subsumption problem (based on Covington, 1994)

```
s  ---> [np,vp].
np ---> [det,n].
vp ---> [vbar(0)].
vp ---> [vbar(X),complements(X)].
vbar(X) ---> [v(X)].
vbar(X) ---> [adv,v(X)].
complements(1) ---> [np].
complements(2) ---> [np,np].

lex(det,the).
lex(n,dog).
lex(n,cat).
lex(adv,often).
lex(v(0),sings).                % intransitive verb
lex(v(1),chases).               % transitive verb
lex(v(2),gives).                % ditransitive verb
```

Example trace

```
| ?- recognize([the,dog,chases,the,cat],s).
START:                          1: 0--S -> * [s]---------0
PRED s in 1:                    2: 0--s -> * [np,vp]-----0
PRED np in 2:                   3: 0--np-> * [det,n]-----0
SCAN 1 (the):                   4: 0--det---------------1
COMP 3 + 4:                     5: 0--np-> * [n]---------1
SCAN 2 (dog):                   6: 1--n-----------------2
COMP 5 + 6:                     7: 0--np----------------2
COMP 2 + 7:                     8: 0--s -> * [vp]--------2
PRED vp in 8:                   9: 2--vp-> * [vbar(0)]---2
PRED vbar(0) in 9:             10: 2--vbar(0)-> * [v(0)]------2
PRED vbar(0) in 9:             11: 2--vbar(0)-> * [adv,v(0)]--2
PRED vp in 8:                  12: 2--vp-> * [vbar(_3377),complements(_3377)]2
PRED vbar(_3377) in 12:            =10
PRED vbar(_3377) in 12:            =11
SCAN 3 (chases):               13: 2--v(1)---------------3
SCAN 4 (the):                  14: 3--det----------------4
SCAN 5 (cat):                  15: 4--n-----------------5
no
```

Adding the subsumption check to the code

Change the first clause of `enter_edge/3` using unification

```
enter_edge(I,J,State) :-
   chart(I,J,State),
   !.
```

to a version that tests for subsumption

```
enter_edge(I,J,State) :-
   chart(I,J,ChartState),
   subsumes_chk(ChartState,State)
   !.
```

Check for subsumption among terms

(from Covington, 1994, based on O'Keefe's Edinburgh library)

```
% subsumes_chk(?T1,?T2)
%  Succeeds if term T1 subsumes T2, i.e.,
%  T1 and T2 can be unified without further
%  instantiating T2.

subsumes_chk(T1,T2) :-
   \+ ( numvars(T2), \+ (T1 = T2) ).


% numvars(+Term,-NewTerm)
%  Instantiates each variable in Term to a unique
%  term in the series vvv(0), vvv(1), vvv(2)...

numvars(Term) :-
   numvars_aux(Term,0,_).
```

```
numvars_aux(Term,N,N) :- atomic(Term), !.

numvars_aux(Term,N,NewN) :-
   var(Term), !,
   Term = vvv(N),
   NewN is N+1.

numvars_aux(Term,N,NewN) :-
   Term =.. List,
   numvars_list(List,N,NewN).

numvars_list([],N,N).
numvars_list([Term|Terms],N,NewN) :-
   numvars_aux(Term,N,NextN),
   numvars_list(Terms,NextN,NewN).
```

The restriction problem

Shieber et al. (1995): Grammar accepting ab^n with N being instantiated to the successor representation of n.

$$\begin{aligned}
&\mathbf{start} \rightarrow \mathbf{r}(0, N)\\
&\mathbf{r}(X, N) \rightarrow \mathbf{r}(s(X), N)\ \mathbf{b}\\
&\mathbf{r}(N, N) \rightarrow \mathbf{a}
\end{aligned}$$

Prediction step with unification will loop:

$$\begin{aligned}
&{}_0[\mathbf{start} \rightarrow\ \bullet\ {}_0\mathbf{r}(0, N)]\\
&{}_0[\mathbf{r}(0, N) \rightarrow\ \bullet\ {}_0\mathbf{r}(s(0), N)\ \mathbf{b}]\\
&{}_0[\mathbf{r}(s(0), N) \rightarrow\ \bullet\ {}_0\mathbf{r}(s(s(0)), N)\ \mathbf{b}]\\
&{}_0[\mathbf{r}(s(s(0)), N) \rightarrow\ \bullet\ {}_0\mathbf{r}(s(s(s(0))), N)\ \mathbf{b}]\\
&\ldots
\end{aligned}$$

Using restriction to prevent prediction loops

- Prediction terminates for grammars with atomic categories, since a new item is only added to the chart if not already there and there is a finite number of atomic categories.

- Moving beyond atomic categories, there can be an infinite number of non-atomic categories.

- Prediction loop on left-recursive rules can be problem again.

- Solution: use **restriction** on prediction substitution to limit to finite number of cases

Prediction with restriction

for each ${}_i[A \rightarrow \alpha \bullet_j B \beta]$ in chart
  for each $B' \rightarrow \gamma$ in rules
    add ${}_j[\sigma(B \rightarrow \bullet_j \gamma)]$ with $\sigma =$ restriction(mgu(B, B')) to chart

– restriction(mgu(B, B')) can be any operation reducing the number of possible substitutions to finite classes:

(a) depth bound on term complexity
(b) elimination of terms that are known to grow indefinitely
(c) use only of selected terms known not to grow indefinitely

– sound since predicted edge only step towards completion!

References

Covington, Michael A. (1994). *Natural Language Processing for Prolog Programmers*. Englewood Cliffs, NJ: Prentice-Hall.

Shieber, Stuart M., Yves Schabes and Fernando C. N. Pereira (1995). Principles and Implementation of Deductive Parsing. *Journal of Logic Programming* 24, 3–36.