# From well-formed substring tables to active charts

Detmar Meurers: Intro to Computational Linguistics I
OSU, LING 684.01

## Overview

- CKY algorithm:
  - explores all analyses in parallel
  - bottom-up
  - stores complete subresults
- desiderata:
  - add top-down guidance (to only use rules derivable from start-symbol), but avoid left-recursion problem of top-down parsing
  - store partial analyses (useful for rules right-hand sides longer than 2)
- Idea: also store partial results, so that the chart contains
  - passive items: complete results
  - active items: partial results

## Representing active chart items

- well-formed substring entry:
  `chart(i,j,A)`: from `i` to `j` there is a constituent of category `A`
- More elaborate data structure needed to store partial results:
  - rule considered + how far processing has succeeded
  - dotted rule:
    $_i[A \rightarrow \alpha \bullet_j \beta]$ with $A \in N$ and $\alpha, \beta \in (\Sigma \cup N)^*$
- active chart entry:
  `chart(i,j,state(A,`$\beta$`))` Note that $\alpha$ is not represented.

## Dotted rule examples

- A dotted rule represents a state in processing a rule.
- Each dotted rule is a hypothesis:

| | We found a *vp* if we still find |
|---|---|
| *vp* → • *v-ditr np pp-to* | a *v-ditr*, a *np*, and a *pp-to* |
| *vp* → *v-ditr* • *np pp-to* | a *np* and a *pp-to* |
| *vp* → *v-ditr np* • *pp-to* | a *pp-to* |
| *vp* → *v-ditr np pp-to* • | nothing |

The first three are examples of **active items** (or **active edges**)

The last one is a **passive item/edge**.

## The three actions in Earley's algorithm

In $_i[A \rightarrow \alpha \bullet_j B\beta]$ we call $B$ the *active constituent*.

- **Prediction:** Search all rules realizing the active constituent.
- **Scanning**: Scan over each word in the input string.
- **Completion:** Combine an active edge with each passive edge covering its active constituent.

## A closer look at the three actions

**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
```

## Eliminating scanning

**Scanning:** 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:** 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

**Observation:** Scanning = completion + words as passive edges. One can thus simplify scanning to adding a passive edge for each word:

```
for each $w_j$ in $w_1 \dots w_n$
  add ${}_{j-1}[w_j \rightarrow \bullet_j]$ to chart
```

## Earley's algorithm without scanning

**General setup:**
apply prediction and completion to every item added to chart

**Start:** add ${}_0[start \rightarrow \bullet_0\, s]$ to chart

```
for each $w_j$ in $w_1 \dots w_n$
  add ${}_{j-1}[w_j \rightarrow \bullet_j]$ to chart
```

**Success state:** ${}_0[start \rightarrow s \bullet_n]$

## A tiny example grammar

Lexicon:

| | | |
|---|---|---|
| vp | → | left |
| det | → | the |
| n | → | boy |
| n | → | girl |

Syntactic rules:

| | | |
|---|---|---|
| s | → | np vp |
| np | → | det n |

## An example run

| | |
|---|---|
| start | 1. ${}_0$[start → $\bullet_0$ s] |
| predict from 1 | 2. ${}_0$[s → $\bullet_0$ np vp] |
| predict from 2 | 3. ${}_0$[np → $\bullet_0$ det n] |
| predict from 3 | 4. ${}_0$[det → $\bullet_0$ the] |
| scan "the" | 5. ${}_0$[the → $\bullet_1$] |
| complete 4 with 5 | 6. ${}_0$[det → the $\bullet_1$] |
| complete 3 with 6 | 7. ${}_0$[np → det $\bullet_1$ n ] |
| predict from 7 | 8. ${}_1$[n → $\bullet_1$ boy ] |
| predict from 7 | 9. ${}_1$[n → $\bullet_1$ girl ] |
| scan "boy" | 10. ${}_1$[boy → $\bullet_2$] |
| complete 8 with 10 | 11. ${}_1$[n → boy $\bullet_2$] |
| complete 7 with 11 | 12. ${}_0$[np → det n $\bullet_2$] |
| complete 2 with 12 | 13. ${}_0$[s → np $\bullet_2$ vp] |
| predict from 13 | 14. ${}_2$[vp → $\bullet_2$ left] |
| scan "left" | 15. ${}_2$[left → $\bullet_3$] |
| complete 14 with 15 | 16. ${}_2$[vp → left $\bullet_3$] |
| complete 13 with 16 | 17. ${}_0$[s → np vp $\bullet_3$] |
| complete 1 with 17 | 18. ${}_0$[start → s$\bullet_3$] |

## The Earley algorithm in Prolog

**(parser/earley/earley.pl)**

```
:- 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
   (B ---> Gamma),
   enter_edge(J,J,state(B,Gamma)),
   fail
 ; true.

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

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

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

## The tiny example grammar

**(parser/earley/earley_grammar.pl)**

```
% lexicon:
vp  ---> [left].
det ---> [the].
n   ---> [boy].
n   ---> [girl].

% syntactic rules:
s  ---> [np, vp].
np ---> [det, n].
```

## The example run in Prolog

**(parser** parser/earley/earley_trace.pl, **grammar:** parser/earley/earley_grammar.pl**)**

```
| ?- recognize([the,boy,left]).
START:             1: 0-state(S,[s])--------0
PRED s in 1:       2: 0-state(s,[np,vp])----0
PRED np in 2:      3: 0-state(np,[det,n])---0
PRED det in 3:     4: 0-state(det,[the])----0
SCAN 1 (the):      5: 0-state(the,[])-------1
COMP 4 + 5:        6: 0-state(det,[])-------1
COMP 3 + 6:        7: 0-state(np,[n])-------1
PRED n in 7:       8: 1-state(n,[boy])------1
PRED n in 7:       9: 1-state(n,[girl])-----1
SCAN 2 (boy):     10: 1-state(boy,[])-------2
COMP 8 + 10:      11: 1-state(n,[])---------2
COMP 7 + 11:      12: 0-state(np,[])--------2
COMP 2 + 12:      13: 0-state(s,[vp])-------2
PRED vp in 13:    14: 2-state(vp,[left])----2
SCAN 3 (left):    15: 2-state(left,[])------3
COMP 14 + 15:     16: 2-state(vp,[])--------3
COMP 13 + 16:     17: 0-state(s,[])---------3
COMP 1 + 17:      18: 0-state(S,[])---------3
SUCCESS: 18
```

## Improving the efficiency of lexical access

- In the setup just described
  - words are stored as passive items so that
  - prediction is used for preterminal categories. The set of predicted words for a preterminal can be huge.
- If each word in the grammar is introduced by a preterminal rule $cat \rightarrow word$ one can add a **passive item for each preterminal category** which can dominate the word instead of for the word itself.
- What needs to be done:
  - syntactically distinguish syntactic rules (`--->`/2) from rules with preterminals on the left-hand side, i.e. lexical entries (`lex`/2).
  - modify scanning to take lexical entries into account

## Code change for preterminals as passive edges

**(parser/earley/preterminals/earley.pl)**

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

is changed to

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

## The tiny example grammar in the modified format

(parser/earley/preterminals/grammar1.pl)

```
% lexicon:
lex(vp,left).
lex(det,the).
lex(n,boy).
lex(n,girl).

% syntactic rules:
s  ---> [np, vp].
np ---> [det, n].
```

## The improved example run

(parser parser/earley/preterminals/earley_trace.pl, grammar: parser/earley/preterminals/grammar1.pl)

```
| ?- recognize([the,boy,left],s).
START:              1: 0--state(S,[s])-------0
PRED s in 1:        2: 0--state(s,[np,vp])---0
PRED np in 2:       3: 0--state(np,[det,n])--0
SCAN 1 (the):       4: 0--state(det,[])------1
COMP 3 + 4:         5: 0--state(np,[n])------1
SCAN 2 (boy):       6: 1--state(n,[])--------2
COMP 5 + 6:         7: 0--state(np,[])-------2
COMP 2 + 7:         8: 0--state(s,[vp])------2
SCAN 3 (left):      9: 2--state(vp,[])-------3
COMP 8 + 9:        10: 0--state(s,[])--------3
COMP 1 + 10:       11: 0--state(S,[])--------3
SUCCESS: 11
```

## Towards more flexible control

The algorithms, we saw

- use the Prolog database to store the chart and
- Prolog backtracking on edges in chart instead of an explicit agenda.

Alternatively, one can

- explicitly introduce an **agenda**
- to store and work off edges in any order one likes.

## Earley-recognizer with explicit agenda and chart

(parser/earley/agenda/earley.pl)

```
:- op(1200,xfx,'--->').  % Operator for grammar rules

% Data structures: chart(From,To,Category)
% ------------------------------------------------------
% recognize(+WordList)
% top-level predicate for Earley recognizer

recognize(String,Startsymbol) :-
   StartAgenda=[chart(0,0,state('S',[Startsymbol]))],
   process_agenda(StartAgenda,[],Chart0),
   scan(String,0,N,Chart0,Chart),
   element(chart(0,N,state('S',[])),Chart).
```

```
% process_agenda(+Agenda,+ChartIn,-ChartOut)

process_agenda([],X,X).
process_agenda([Edge|Agenda0],Chart0,Chart) :-
   element(Edge,Chart0), !,
   process_agenda(Agenda0,Chart0,Chart).
process_agenda([Edge|Agenda0],Chart0,Chart) :-
   Chart1=[Edge|Chart0],
   %
   predict(Edge,PAgenda),
   append(PAgenda,Agenda0,Agenda1),
   %
   complete(Edge,Chart1,CAgenda),
   append(CAgenda,Agenda1,NewAgenda),
   process_agenda(NewAgenda,Chart1,Chart).
```

```
scan([],N,N,Chart,Chart).
scan([W|Ws],JminOne,N,Chart0,Chart) :-
   J is JminOne+1,
   setof(chart(JminOne,J,state(Cat,[])),
         lex(Cat,W),
         Agenda),
   process_agenda(Agenda,Chart0,Chart1),
   scan(Ws,J,N,Chart1,Chart).
```

```
predict(chart(_,J,state(_,[B|_])),Agenda) :-
   setof(chart(J,J,state(B,Gamma)),
         (B ---> Gamma),
         Agenda), !.
predict(_,[]).   % is passive edge or no matching grammar rule


complete(chart(K,J,state(B,[])),Chart,Agenda) :-
   setof(chart(I,J,state(A,Beta)),
         element(chart(I,K,state(A,[B|Beta])), Chart),
         Agenda), !.
complete(_,_,[]). % is active edge or no matching chart edge
```

```
% ------------------------------------------------------
% element(?Element,+List)

element(X,[X|_]).
element(X,[_|L]) :-
   element(X,L).

% ------------------------------------------------------
% append(+List,?List,-List) or append(-List,?List,+List)

append([],L,L).
append([H|T],L,[H|R]) :-
   append(T,L,R).
```