From well-formed substring tables to active charts

- Well-formed substring tables: store complete analyses
- But: combination of complete sub-analyses is redone each time

  vp → *v-ditr np pp-to*
  vp → *v-ditr np np*

- 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 instead of category:
  - 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,`β`))`

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 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 **replace scanning by:** for each w_j in $w_1 \ldots w_n$
add ${}_{j-1}[w_j \rightarrow \bullet_j]$ to chart

Note: Different from scanning, this is done once for each word, i.e., it is not triggered by every new edge added to the chart.

Earley's algorithm

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 $w_1 \ldots w_n$
add ${}_{j-1}[w_j \rightarrow \bullet_j]$ to chart

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

A tiny example grammar

- s → np vp
- np → det n
- vp → left
- det → the
- n → man

An example run

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

The Earley algorithm in PROLOG
(earley.pl)

```
% Data structures: chart(From,To,Category)
:- dynamic chart/3.

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



```
% recognize(+WordList,?Startsymbol)
% top-level predicate for Earley recognizer

recognize(String) :-
   retractall(chart(_,_,_)),
   enter_edge(0,0,state(start,[s])),
   foreach(get_word(Word,JminOne,J,String),
           enter_edge(JminOne,J,state(Word,[]))),
   length(String,N),
   chart(0,N,state(start,[])).
```



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

```
% prediction(+StartPos,+DotPos,+State)
predict(_,J,state(_,[Cat|_])) :- !,
   foreach((Cat ---> RHS),
           enter_edge(J,J,state(Cat,RHS))).
predict(_,_,_).


% completion(+StartPos,+DotPos,+State)
complete(K,J,state(B,[])) :- !,
   foreach(chart(I,K,state(A,[B|Beta])),
           enter_edge(I,J,state(A,Beta))).
complete(_,_,_).
```



```
% get_word(-Element,-JminOne,J,+List)
get_word(X,0,1,[X|_]).
get_word(X,JminOne,J,[_|L]) :-
        get_word(X,_,JminOne,L),
        J is JminOne+1.


% foreach(+Goal1,+Goal2)
foreach(X,Y) :- X, Y, fail.
foreach(_,_).
```


The tiny example grammar
(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

```
| ?- recognize([the,boy,left]).
START:             1: 0-state(start,[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(start,[])-----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 such as $cat \rightarrow word$ or $lex(cat, word)$, one can
  - add a **passive item for each preterminal category** which can dominate the word instead of for the word itself.

Code change for preterminals as passive edges
(earley/passive_preterminals/earley_passive.pl)

In `recognize/1` change

```
foreach(get_word(Word,JminOne,J,String),
        enter_edge(JminOne,J,state(Word,[]))),
```

to take into account the preterminal category:

```
foreach((get_word(Word,JminOne,J,String),
         lex(Cat,Word)),
        enter_edge(JminOne,J,state(Cat,[]))),
```


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.

Top-down recognizer with agenda

```
% Data structures: chart(From,To,state(LHS,RHS-LIST))

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

% recognize(+WordList)
% top-level predicate for Earley recognizer

recognize(String) :-
   enter_string(String,0,N,Agenda),
   FullAgenda = [chart(0,0,state(start,[s])) | Agenda],
   fill_chart(FullAgenda,[],Chart),
   element(chart(0,N,state(start,[])),Chart).
```

```
enter_string([],N,N,[]).
enter_string([Word|RestString],JminOne,N,FullAgenda) :-
   J is JminOne + 1,
   findall(chart(JminOne,J,state(Cat,[])),
           lex(Cat,Word),
           FirstAgenda),
   enter_string(RestString,J,N,AgendaRest),
   append(FirstAgenda,AgendaRest,FullAgenda).
```



```
% enter_edges(+EdgeList,+ChartIn,-ChartOut)

enter_edges([],X,X).
enter_edges([Edge | Edges],ChartIn,ChartOut) :-
   enter_edge(Edge,ChartIn,ChartMid),
   enter_edges(Edges,ChartMid,ChartOut).

enter_edge(Edge,Chart,Chart) :-  member(Edge,Chart),!.
enter_edge(Edge,Chart,[Edge|Chart]).
```



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

fill_chart([],X,X).
fill_chart([Edge|RestAgenda],ChartIn,ChartOut) :-
   ChartMid = [Edge|ChartIn],
   %
   predict(Edge,PredictAgenda),
   complete(Edge,ChartMid,CompleteAgenda),
   %
   append(PredictAgenda,RestAgenda,RestPredAgenda),
   append(RestPredAgenda,CompleteAgenda,NewAgenda),
   %
   fill_chart(NewAgenda,ChartMid,ChartOut).
```



```
predict(chart(_,J,state(_,[B|_])),Agenda) :- !,
   findall(chart(J,J,state(B,Gamma)),
           (B ---> Gamma),
           Agenda).
predict(_,[]).


complete(chart(K,J,state(B,[])),Chart,Agenda) :- !,
   findall(chart(I,J,state(A,Beta)),
           element(chart(I,K,state(A,[B|Beta])), Chart),
           Agenda).
complete(_,_,[]).
```

```
% 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).
```