Remembering subresults:
From well-formed substring tables to active charts

Detmar Meurers: Intro to Computational Linguistics I
OSU, LING 684.01, 17., 19., 21. February 2003

Problem: Inefficiency of recomputing subresults

Two example sentences and their potential analysis:

(1) He [[gave [the young cat]] [to Bill]].
(2) He [[gave [the young cat]] [some milk]].

The corresponding grammar rules:

```
v_np ---> [v_ditrans, np].
vp   ---> [v_np, pp_to].
vp   ---> [v_np, np].
```

Solution: Memoization

- Store intermediate results:
  a) completely analyzed constituents:
     well-formed substring table or **(passive) chart**
  b) partial and complete analyses:
     (active) chart

- All intermediate results need to be stored for completeness.

- All possible solutions are explored in parallel.

CYK Parser

- Developed independently by Cocke, Younger, and Kasami
- Grammar has to be in Chomsky Normal Form (CNF), only
  - RHS with a single terminal: $A \rightarrow a$
  - RHS with two non-terminals: $A \rightarrow BC$
- Sentence representation showing position and word indices:

$$\cdot_0 \; w_1 \; \cdot_1 \; w_2 \; \cdot_2 \; w_3 \; \cdot_3 \; w_4 \; \cdot_4 \; w_5 \; \cdot_5 \; w_6 \; \cdot_6$$

  For example:

$$\cdot_0 \text{ the } \cdot_1 \text{ young } \cdot_2 \text{ boy } \cdot_3 \text{ saw } \cdot_4 \text{ the } \cdot_5 \text{ dragon } \cdot_6$$

The passive chart

- The well-formed substring table, henceforth (passive) chart, for a string of length n is an $n \times n$ matrix.
- An entry in a field (i, j) of the chart encodes the set of categories which spans the string from position i to j.
- More formally: chart(i,j) $= \{A \mid A \stackrel{*}{\Rightarrow} w_{i+1} \ldots w_j\}$

Coverage represented in the chart

An input sentence with 6 words:

$$\cdot_0 \; w_1 \; \cdot_1 \; w_2 \; \cdot_2 \; w_3 \; \cdot_3 \; w_4 \; \cdot_4 \; w_5 \; \cdot_5 \; w_6 \; \cdot_6$$

Coverage represented in the chart:

FROM: \ TO:	1	2	3	4	5	6
0	0–1	0–2	0–3	0–4	0–5	0–6
1		1–2	1–3	1–4	1–5	1–6
2			2–3	2–4	2–5	2–6
3				3–4	3–5	3–6
4					4–5	4–6
5						5–6

Example for coverage represented in chart

Example sentence:

$$\cdot_0 \text{ the } \cdot_1 \text{ young } \cdot_2 \text{ boy } \cdot_3 \text{ saw } \cdot_4 \text{ the } \cdot_5 \text{ dragon } \cdot_6$$

Coverage represented in chart:

	1	2	3	4	5	6
0	the	the young	the young boy	the young boy saw	the young boy saw the	the young boy saw the dragon
1		young	young boy	young boy saw	young boy saw the	young boy saw the dragon
2			boy	boy saw	boy saw the	boy saw the dragon
3				saw	saw the	saw the dragon
4					the	the dragon
5						dragon

An example for a filled-in chart

Input sentence:

$\cdot_0$ the $\cdot_1$ young $\cdot_2$ boy $\cdot_3$ saw $\cdot_4$ the $\cdot_5$ dragon $\cdot_6$

Chart:

	1	2	3	4	5	6
0	{Det}	{}	{NP}	{}	{}	{S}
1		{Adj}	{N}	{}	{}	{}
2			{N}	{}	{}	{}
3				{V}	{}	{VP}
4					{Det}	{NP}
5						{N}

Grammar:

S $\rightarrow$ NP VP
VP $\rightarrow$ Vt NP
NP $\rightarrow$ Det N
N $\rightarrow$ Adj N
Vt $\rightarrow$ saw
Det $\rightarrow$ the
Det $\rightarrow$ a
N $\rightarrow$ dragon
N $\rightarrow$ boy
Adj $\rightarrow$ young

Filling in the chart left-to-right, depth-first

	1	2	3	4	5	6
0	**1!**	**3**	**6**	**10**	**15**	**21**
1		**2!**	**5**	**9**	**14**	**20**
2			**4!**	**8**	**13**	**19**
3				**7!**	**12**	**18**
4					**11!**	**17**
5						**16!**

for $j := 1$ to length($string$)
    lexical_chart_fill$(j-1, j)$
    for $i := j-2$ down to 0
        syntactic_chart_fill(i, j)

lexical_chart_fill(j-1,j)

- Idea: Lexical lookup. Fill the field $(j-1, j)$ in the chart with the preterminal category dominating word j.

- Realized as:

$$chart(j-1, j) := \{\mathsf{X} \mid \mathsf{X} \rightarrow \mathsf{word}_j \in \mathsf{P}\}$$

syntactic_chart_fill(i,j)

- Idea: Perform all reduction step using syntactic rules such that the reduced symbol covers the string from i to j.

- Realized as:

$$chart(i,j) = \left\{ A \;\middle|\; \begin{array}{l} A \rightarrow BC \in P, \\ i < k < j, \\ B \in chart(i,k), \\ C \in chart(k,j) \end{array} \right\}$$

Explicit version of syntactic_chart_fill(i,j)

- Needed: version making explicit enumerations of
  - every possible value of k and
  - every context free rule

- Code:
  $chart(i, j) := \{\}$.
  for $k := i + 1$ to $j - 1$
      for every $A \rightarrow BC \in P$
          if $B \in chart(i, k)$ and $C \in chart(k, j)$ then
              $chart(i, j) := chart(i, j) \cup \{\mathsf{A}\}$.

Overview of the CYK algorithm

Input: start category S and input $string$

$n :=$ length$(string)$

for $j := 1$ to n
    lexical_chart_fill$(j-1, j)$
    for $i := j-2$ down to 0
        syntactic_chart_fill(i, j)

Output: if $S \in chart(0, n)$ then accept else reject

The complete CYK algorithm

Input: start category S and input $string$

$n :=$ length($string$)

for $j := 1$ to n
  $chart(j-1, j) := \{\mathsf{X} \mid \mathsf{X} \rightarrow \mathsf{word}_j \in \mathsf{P}\}$
  for $i := j - 2$ down to 0
    $chart(i, j) := \{\}$
    for $k := i + 1$ to $j - 1$
      for every $A \rightarrow BC \in P$
        if $B \in chart(i, k)$ and $C \in chart(k, j)$ then
          $chart(i, j) := chart(i, j) \cup \{\mathsf{A}\}$

Output: if $S \in chart(0, n)$ then accept else reject

Dynamic knowledge bases in PROLOG

- Declaration of a dynamic predicate: `dynamic/1` declaration, e.g:

  ```
  :- dynamic chart/3.
  ```

  to store facts of the form `chart(From,To,Category)`:

- Add a fact to the database: `assert/1`, e.g.:

  ```
  assert(chart(1,3,np)).
  ```

  Special versions `asserta/1`/`assertz/1` ensure adding facts first/last.

- Removing a fact from the database: `retract/1`, e.g.:

  ```
  retract(chart(1,_,np)).
  ```

  To remove all matching facts from the database use `retractall/1`

The CYK algorithm in PROLOG (parser/cky/cky.pl)

```
:- dynamic chart/3.                  % chart(From,To,Category)
:- op(1100,xfx,'--->').              % Operator for grammar rules


% recognize(+WordList,?Startsymbol): top-level of CYK recognizer

recognize(String,Cat) :-
   retractall(chart(_,_,_)),         % initialize chart
   length(String,N),                 % determine length of string
   fill_chart(String,0,N),           % call parser to fill the chart
   chart(0,N,Cat).                   % check whether parse successful
```

```
% fill_chart(+WordList,+Current minus one,+Last)
% J-LOOP from 1 to n

fill_chart([],N,N).
fill_chart([W|Ws],JminOne,N) :-
   J is JminOne + 1,
   lexical_chart_fill(W,JminOne,J),
   %
   I is J - 2,
   syntactic_chart_fill(I,J),
   %
   fill_chart(Ws,J,N).
```

```
% lexical_chart_fill(+Word,+JminOne,+J)
% fill diagonal with preterminals

lexical_chart_fill(W,JminOne,J) :-
   (Cat ---> [W]),
   add_to_chart(JminOne,J,Cat),
   fail
 ; true.
```

```
% syntactic_chart_fill(+I,+J)
% I-LOOP from J-2 downto 0

syntactic_chart_fill(-1,_) :- !.
syntactic_chart_fill(I,J) :-
   K is I+1,
   build_phrases_from_to(I,K,J),
   %
   IminOne is I-1,
   syntactic_chart_fill(IminOne,J).
```

```
% build_phrases_from_to(+I,+Current-K,+J)
% K-LOOP from I+1 to J-1

build_phrases_from_to(_,J,J) :- !.
build_phrases_from_to(I,K,J) :-
   chart(I,K,B),
   chart(K,J,C),
   (A ---> [B,C]),
   add_to_chart(I,J,A),
   fail
 ; KplusOne is K+1,
   build_phrases_from_to(I,KplusOne,J).
```

```
% add_to_chart(+Cat,+From,+To): add if not yet there
add_to_chart(From,To,Cat) :-
   chart(From,To,Cat),
   !.
add_to_chart(From,To,Cat) :-
   assertz(chart(From,To,Cat).
```

From well-formed substring tables to active charts

- 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:

    $${}_i[A \rightarrow \alpha \bullet_j \beta] \qquad \text{with } A \in N \text{ and } \alpha, \beta \in (\Sigma \cup N)^*$$

- active chart entry:
  `chart(i,j,state(A,`β`))` Note that α 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
$vp \longrightarrow \bullet$ *v-ditr np pp-to*	a *v-ditr*, a *np*, and a *pp-to*
vp $\longrightarrow$ *v-ditr* $\bullet$ *np pp-to*	a *np* and a *pp-to*
vp $\longrightarrow$ *v-ditr np* $\bullet$ *pp-to*	a *pp-to*
vp $\longrightarrow$ *v-ditr np pp-to* $\bullet$	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 $w_1 \ldots w_n$
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 $w_1 \ldots w_n$
  add ${}_{j-1}[w_j \rightarrow \bullet_j]$ to chart

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

A tiny example grammar

Lexicon:

vp $\longrightarrow$ left

det $\longrightarrow$ the

n $\longrightarrow$ boy

n $\longrightarrow$ girl

Syntactic rules:

s $\longrightarrow$ np vp

np $\longrightarrow$ det n

An example run

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