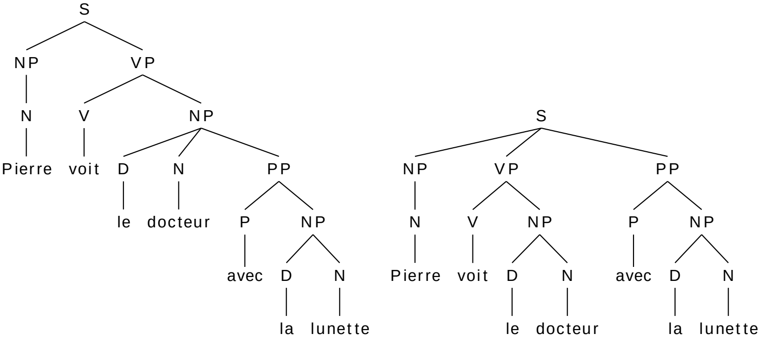
Linguistique LI6

Benoit Crabbe

2012-2013

The key problem for parsing natural languages:massive ambiguity



Problem:exponential number of parses

- Pierre voit le docteur avec la lunette et Jean voit le savant avec la lunette et...
- $2 \times 2 \times \ldots$ = (2^k) with k the number of coordinations (= repetitions of the ambiguous construction)

Parsing : goals

Goals (for this lesson):

- Generate every possible analysis of the sentence (manage ambiguity)
- Since the number of parses for a sentence is exponential, we need to find a way to pack the computations of all these parses.
- Use of data structures and algorithms allowing to share subparses common to several full parses.

Plan

1. Parsing context free grammars
   - Parsing as intersection
   - Classical parsing algorithms
     - CKY
     - The Earley algorithm
     - Left corner
2. Parsing TAGs

The Intersection theorem

Theorem

The intersection of a context free language with a regular language is again a context free language (Bar Hillel 1964)

The theorem's proof is constructive: given an FSA and a CFG it yields an **intersection grammar** which is itself a CFG

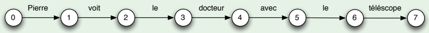
Illustration

CFG

S → NP VP
S → NP VP PP
VP → V NP
NP → N
NP → D N
NP → D N PP
PP → P NP

$\cap$

DFA



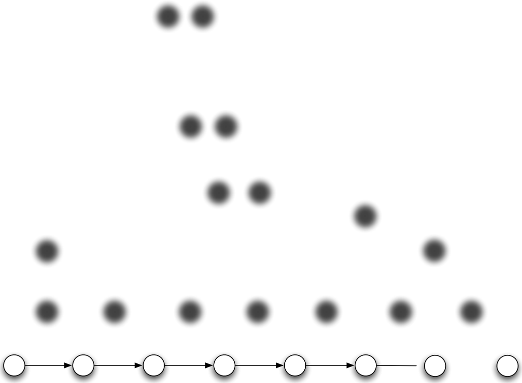
=

CFG$_\cap$

$S_{0,7} \rightarrow NP_{0,1}\ VP_{1,7}$
$S_{0,7} \rightarrow NP_{0,1}\ VP_{1,4}\ PP_{4,7}$
$VP_{1,4} \rightarrow V_{1,2}\ NP_{2,4}$
$VP_{1,7} \rightarrow V_{1,2}\ NP_{2,7}$
$NP_{0,1} \rightarrow N_{0,1}$
$NP_{2,4} \rightarrow D_{2,3}\ N_{3,4}$
$NP_{2,7} \rightarrow D_{2,3}\ N_{3,4}\ PP_{4,7}$
$NP_{5,7} \rightarrow D_{5,6}\ N_{6,7}$
$PP_{4,7} \rightarrow P_{4,5}\ NP_{5,7}$

The shared forest

The intersection grammar can also be interpreted as an and/or graph (the **shared forest**)



Read from top down with the dotted red circle denoting a disjunctive node, the shared forest encodes the two parses generated by the grammar for the sentence

Properties of the shared forest

- The shared forest naturally **packs** several subparses together
  - Observe that the $PP_{4;7}$ node or the $NP_{0;1}$ node are shared among the two parses.
  - In what follows we assume that the FSA given as input encodes just one string $w = a_1 \dots a_n$ with an initial state labelled 0 and a final state labelled n.
  - The non terminals of the intersection grammar noted $X_{i;j}$ are indexed by state positions ($0 \leq i < j \leq n$) where X is a non terminal from the input grammar. These triples are called parse **items**
  - The start symbol of the intersection grammar is $S_{0;n}$ where S is axiom of the input grammar

Construction of the intersection grammar

- Let $\mathcal{G}$ be the input CFG and $|w| = n$ be the length of the input string, the intersection algorithm proceeds as follows:

```
function INTERSECTION($\mathcal{G}, w$)
    $\mathcal{G}_\cap \leftarrow$ CFG with start symbol $S_{0,n}$ and empty set of rules
    for all rules $A \rightarrow X^1 \ldots X^m$ from $\mathcal{G}$ do
        for all sequences of positions $i_0, \ldots, i_m (0 \leq i_0 \leq i_m \leq n)$ do
            add the rule $A_{i_0,i_m} \rightarrow X^1_{i_0,i_1} \ldots X^m_{i_{m-1},i_m}$ to $\mathcal{G}_\cap$
        end for
    end for
    for all $i \quad (1 \leq i \leq n)$ do                    ▷ Terminals emitting rules
        add the rule $a^i_{i-1,i} \rightarrow a_i$ to $\mathcal{G}_\cap$
    end for
end function
```

Algorithms for computing generating and reachable symbols

- Here is an algorithm for computing generating symbols. Let denote the set of terminal symbols, and $*$ the set of strings of terminal symbols

```
function GENERATING($\mathcal{G}$)
    oldgen $\leftarrow \emptyset$
    gen $\leftarrow \{A|A \rightarrow v \quad (v \in \Sigma*)\}$
    while oldgen $\neq$ gen do
        oldgen $\leftarrow$ gen
        gen $\leftarrow \{A|A \rightarrow \alpha \quad (\alpha \in (\Sigma \cup \text{oldgen})*)\}$
    end while
    return gen
end function
```

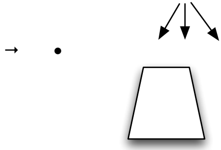
- For computing reachable symbols, one can use graph reachability
- Observe that generating proceeds **bottom-up** while reachability proceeds **top down.**

Combining intersection with Generating

- Instead of running $\textsc{Intersection}(\mathcal{G}, w)$,$\textsc{Generating}(\mathcal{G})$ and reachability sequentially, running them simultaneously saves time and space
- Divided in two steps:
  1. Compute a set of generating nonterminals (without explicitly building the intersection rules) = recognition
  2. Explicit construction of the intersection rules = parsing

Dotted items

- To this end, we make use of **dotted items** of the form $\langle A \rightarrow \alpha \bullet \beta, i, j \rangle$ where $A \rightarrow \alpha\beta \in \mathcal{G}$ and $\alpha \neq \epsilon$. α is the prefix of the item, β the suffix.
  - The prefix is a sequence of generating nonterminals generating from state i to state j on the input DFA



  - Dotted items can be seen as a partial result towards discovering a complete result : here $\langle X \rightarrow AB \bullet C, i, j \rangle$ can be seen as a partial result towards completing $\langle X \rightarrow ABC \bullet, i, j + k \rangle$ (also abbreviated as $X_{i;j+k}$)
- In what follows we will gather dotted items with complete items in a set called GEN.

Agenda

- The parsing algorithm maintains an **agenda** storing newly obtained items that are still to be put in GEN.
- New items are combined with existing old items to derive additional items.
- The agenda is a data structure storing new items waiting to be processed

Generating intersection algorithm

```
  = $a_1 \dots a_n$
function GeneratingIntersection($\mathcal{G}$,  )
    Gen $\leftarrow \emptyset$
    Agenda $\leftarrow a^i_{i-1,i} | 1 \leq i \leq n\}$                          ▷ Insert terminals
    while Agenda $\neq \emptyset$ do
        remove some Item from Agenda
        if Item $\notin$Gen then                                                  ▷ Chart insertion
            Gen $\leftarrow$ Gen $\cup$ Item
            if Item = $X_{j,k}$ then
                for all $\langle A \rightarrow \alpha \bullet X\beta, i, j\rangle$ $\in$Gen do      ▷ Complete (Left)
                    Agenda $\leftarrow$ Agenda $\cup\langle A \rightarrow \alpha X \bullet \beta, i, k\rangle$
                end for
                for all $\langle A \rightarrow X\beta, i, j\rangle$ $\in$Gen do                   ▷ Complete (Left)
                    Agenda $\leftarrow$ Agenda $\cup\langle A \rightarrow X \bullet \beta, i, k\rangle$
                end for
            end if
            if Item = $\langle A \rightarrow \alpha \bullet X\beta, i, j\rangle$ then               ▷ Complete (Right)
                for all $X_{j,k} \in$ Gen do
                    Agenda $\leftarrow$ Agenda $\cup\langle A \rightarrow \alpha X \bullet \beta, i, k\rangle$
                end for
            end if
            if Item = $\langle A \rightarrow \alpha \bullet, i, j\rangle$ then
                Gen $\leftarrow$ Gen $\cup\{A_{i,j}\}$
            end if
        end if
    end while
    return Gen
end function
```

Note: Gen is often called a **chart**

Exercises

- Does this algorithm terminates ? *consider the infinitely ambiguous grammar :* $\mathcal{G}$ with rules
  - $A \to BC$
  - $B \to B \,|\, b$
  - $C \to c$

  and axiom A. Does $\textsc{GeneratingIntersection}(\mathcal{G}, bc)$ terminates ?
- What is the time complexity of the algorithm ? (worst case, as a function of n)

Exercises

- Does this algorithm terminates ? *consider the infinitely ambiguous grammar :* $\mathcal{G}$ with rules
    - $A \rightarrow BC$
    - $B \rightarrow B \,|\, b$
    - $C \rightarrow c$

  and axiom A. Does $\textsc{GeneratingIntersection}(\mathcal{G}, bc)$ terminates ?
- What is the time complexity of the algorithm ? (worst case, as a function of n)

- **Solution 1:** the key to termination is chart insertion, the algorithm cannot process two times the same item
- **Solution 2:** The combinatorics lies in the complete sections i, j, k can all span over the n input positions, hence $\mathcal{O}(n^3)$

Forest Generation

- $\textsc{GenerateIntersection}(\mathcal{G}, w)$ builds a set of items bottom up, not the intersection grammar. There is no guarantee that items in $\textsc{Gen}$ are reachable from the axiom.
- The algorithm $\textsc{IntersectionFiltered}(S_{0,n})$ processes the items starting from the axiom in a top down manner in order to guarantee reachability (and also builds the actual intersection grammar)
- We keep a set $\textsc{Done}$ initally empty storing all the rules already built, to prevent the addition of several identical rules by iterative calls to $\textsc{IntersectionFiltered}$ (i, X, j)

Intersection Filtered and shared forest construction

```
function $\textsc{IntersectionFiltered}(X_{i,j})$
    if $X_{i,j} \notin \textsc{Done}$ then                                    ▷ Blocks useless recursions
        $\textsc{Done} \leftarrow \textsc{Done} \cup \{X_{i,j}\}$
        if $X$ is terminal $a$ then
            add $a_{i,j} \rightarrow a$ to $\mathcal{G}_\cap$
        else $X$ is non terminal $a$
            for all $A \rightarrow X_1 \dots X_m (m > 0)$ and sequences
   $\langle A \rightarrow X_1 \dots X_{m-1}, X_m \bullet, i_0, i_m\rangle, X_{m(i_{m-1},i_m)}$,
   $\langle A \rightarrow X_1 \dots X_{m-1}, \bullet X_m, i_0, i_{m-1}\rangle, X_{m-1(i_{m-2},i_{m-1})}$,
   $\dots$
   $\langle A \rightarrow X_1 \bullet \dots X_{m-1}, X_m, i_0, i_1\rangle, X_{1(i_0,i_1)} \in \textsc{Gen}$
   where $i_0 = i$ and $i_m = j$ do
                add $A_{(i_0,i_m)} \rightarrow X_{1(i_0,i_1)} \dots X_{m(i_{m-1},i_m)}$ to $\mathcal{G}_\cap$
                for all $k$ $(1 \leq k \leq m)$ do
                    $\textsc{IntersectionFiltered}(X_{k(i_{k-1},i_k)})$       ▷ Recursive call for each $X_k$
                end for
            end for
        end if
    end if
end function
```

Parsing as a deductive system

- Instead of writing the full blown pseudo-code, parsing algorithms ($\textsc{GenerateIntersection}(\mathcal{G}, w)$) are often presented as deduction systems
- Inference rules have the following form :

$$\frac{\text{antecedent}}{\text{consequent}} \quad \{\text{side conditions}$$

- **Antecedents** stand for items already derived, **consequents** for items produced by the rules, and **side conditions** are additional conditions for the rule to trigger

GENERATEINTERSECTION as a deductive system

- SCAN ($w = a_1 \dots a_n$):

$$\frac{}{a^i_{i-1,i}}\{1 \leq i \leq n$$

- COMPLETE 1 (init rule):

$$\frac{X_{j,k}}{\langle A \rightarrow X \bullet \beta, j, k\rangle}\{A \rightarrow X\,\beta \in \mathcal{G}$$

- COMPLETE 2 (advance dot):

$$\frac{\langle A \rightarrow \alpha \bullet X\beta, i, j\rangle \qquad X_{j,k}}{\langle A \rightarrow \alpha X \bullet \beta, i, k\rangle}$$

- FINISH ITEM:

$$\frac{A \rightarrow \alpha\bullet, i, j}{A_{i,j}}$$

As a rule of thumb, an easy way to find out the time complexity (as a function of input length) of a parsing algorithm amounts to consider the most expressive rule (here COMPLETE 2*) and count the the number of indices allowed to span the input, here three indices ($O(n^3)$)*

Exercise

- Let w = La belle porte le voile
- and G
    - S $\rightarrow$ NP VP
    - NP $\rightarrow$ D N $|$ D A N
    - VP $\rightarrow$ V NP $|$ Cl V
    - D $\rightarrow$ le $|$ la
    - Cl $\rightarrow$ le
    - N $\rightarrow$ belle $|$ porte $|$ voile
    - V $\rightarrow$ porte $|$ voile

  with axiom S.
- Run the recognition algorithm, keep track of the items successively produced.

Solution

- $la_{0,1}, belle_{1,2}, porte_{2,3}, le_{3,4}, voile_{4,5}$ (Scan)
- $\langle D \rightarrow la\bullet, 0, 1\rangle$, $\langle A \rightarrow belle\bullet, 1, 2\rangle$, $\langle N \rightarrow belle\bullet, 1, 2\rangle$, $\langle N \rightarrow porte\bullet, 2, 3\rangle$, $\langle V \rightarrow porte\bullet, 2, 3\rangle$, $\langle Cl \rightarrow le\bullet, 3, 4\rangle$, $\langle D \rightarrow le\bullet, 3, 4\rangle$, $\langle N \rightarrow voile\bullet, 4, 5\rangle$, $\langle V \rightarrow voile\bullet, 4, 5\rangle$ (init rule)
- $\langle NP \rightarrow D \bullet N, 0, 1\rangle$, $\langle NP \rightarrow D \bullet A\, N, 0, 1\rangle$, $\langle VP \rightarrow Cl \bullet V, 3, 4\rangle$, $\langle VP \rightarrow V \bullet NP, 2, 3\rangle$, $\langle NP \rightarrow D \bullet N, 3, 4\rangle$ (init rule)
- $\langle NP \rightarrow DN\bullet, 0, 2\rangle$, $\langle NP \rightarrow D\, A \bullet N, 0, 2\rangle$, $\langle VP \rightarrow Cl\, V\bullet, 3, 5\rangle$, $\langle NP \rightarrow D\, A\, N\bullet, 0, 3\rangle$, $\langle NP \rightarrow D\, N\bullet, 3, 5\rangle$, $\langle VP \rightarrow V\, NP\bullet, 2, 5\rangle$,(advance dot)
- $\langle S \rightarrow NP \bullet VP, 0, 3\rangle$, $\langle S \rightarrow NP \bullet VP, 0, 2\rangle$ (init rule)
- $\langle S \rightarrow NP\, VP\bullet, 0, 5\rangle$, $\langle S \rightarrow NP\, VP\bullet, 0, 5\rangle$ (advance dot)

Skipping up the finish rule everywhere, inference rule ordering while processing is indicative (except for scan)

Exercise : top down recognition

- A top down parser starts with the axiom S and successively replaces lefthand sides of productions with righthandsides
- It replaces the nonterminals in an order from left to right
- Design the inference rules for a top down recognition algorithm for the ambiguous case where S is the axiom of the grammar, and $w = a_1 \dots a_n$ the string to be parsed.
  - Items of your parser will be of the form $\langle \alpha, i \rangle$ where α is a sequence of symbols (terminals or non terminals) and i the length of the string already recognized. $\textsc{TopDown}(S, 1)$
  - Does it terminate ? what happens if the grammar is left recursive (rules of the form $A \rightarrow A\alpha$ with $\alpha \in NT*$) ?
  - Provide an example run with $w = aaabbb$ and the grammar $S \rightarrow aSb, S \rightarrow ab$

Top down recognizer (deductive version)

- Scan:

$$\frac{< a\alpha, i >}{< \alpha, i + 1 >} \qquad w_{i-1} = a$$

- Predict :

$$\frac{< A\alpha, i >}{< \gamma\alpha, i >} \quad A \rightarrow \gamma \in P$$

- Axiom :

$$\frac{}{< S, 0 >}$$

- Goal (sentence of length n):

$$< \epsilon, n >$$

Comment

Note that the items $< \alpha, i >$ do encode the current state of the stack, and the amount of the string already parsed.

Plan

1. Parsing context free grammars
   - Parsing as intersection
   - Classical parsing algorithms
     - CKY
     - The Earley algorithm
     - Left corner
2. Parsing TAGs

Classical parsing algorithms used in NLP

NAME	AUTHOR	MAIN USAGE
CKY	Cocke Kasami Younger (65-67)	mostly weighted parsing
Earley algorithm	Earley 70	symbolic
Left Corner Parser	Rosenkrantz and Lewis 1970	symbolic

Plan

1. Parsing context free grammars
   - Parsing as intersection
   - Classical parsing algorithms
     - CKY
     - The Earley algorithm
     - Left corner
2. Parsing TAGs

CKY

- CKY is used with grammars in Chomsky Normal Form (binary rules)
- CKY is a bottom-up parser : starts with the terminals in the input string and subsequently computes recognized parse trees by reducing already recognized RHS of productions to the non terminal of the lefthand side (invariant: all produced items are ntseess(arly)]TJ/F3810.9
-

Chomsky normal form

- For each CFL L there is a CFG G in Chomsky normal form with $L = L(G)$
- Construction of an equivalent CFG in CNF for a given CFG
  1. For each terminal a : introduce new non terminal C_a, replace a with C_a in all right hand sides of length > 1 and add production $C_a \rightarrow a$
  2. For each production $A \rightarrow B_0 \ldots B_n$ introduce new non terminals $\mathcal{B}_1 \ldots \mathcal{B}_{n-1}$ and replace production with productions :
     - $A \rightarrow B_0 \mathcal{B}_1$
     - $\mathcal{B}_1 \rightarrow B_1 \mathcal{B}_2$
     - $\mathcal{B}_2 \rightarrow B_2 \mathcal{B}_3$
     - $\ldots$
     - $\mathcal{B}_{n-1} \rightarrow B_{n-1} B_n$

CYK in a parsing as deduction framework

CYK can be framed in the parsing as deduction framework as follows

- We consider that the algorithm yields parsing items of the form $< A, i, \ell >$, meaning that the preterminal A covers a span in the input starting at index i of length ℓ
- Inference rules ($w = a_1 \ldots a_n$):
  - Goal :

    $$< S, 0, n >$$

  - Scan :

    $$\frac{}{\langle A, i-1, 1\rangle}\; A \rightarrow w_i \in P$$

  - Complete :

    $$\frac{\langle B, i, \ell_1\rangle \quad \langle C, i+\ell_1, \ell_2\rangle,}{\langle A, i, \ell_1+\ell_2\rangle} \quad A \rightarrow BC \in P$$

CYK parsing scheme

Classical implementations of CKY rely on a fixed inference order. Items are produced from left to right by order of increasing length.

```
function $\textsc{CykParse}(\mathcal{G}, w)$
    $\frac{}{\langle A,i-1,1\rangle} A \rightarrow w_i \in \mathcal{G}_P \quad 1 \leq i \leq n$        ▷ Init
    for $2 \leq \ell \leq n$ do        ▷ $\ell$ is the length of the span
        for $0 \leq i < n$ do
            for $0 < k < \ell$ do        ▷ Completer
                $\frac{\langle B,i,k\rangle \quad \langle C,i+k,\ell-k\rangle}{\langle A,i,\ell\rangle} A \rightarrow B\,C \in \mathcal{G}_P$
            end for
        end for
    end for
end function
```

Illustration

ℓ						
5	S					
4						
3	NP		VP			
2	NP	NP1		NP|VP		
1	D	A|N	N|V	D|Cl	V|N	
	0	1	2	3	4	i
	la	belle	porte	le	voile	

S → NP VP

NP → D N | D NP1

VP → V NP | CL V

NP1 → A N

D → la | le

A → belle

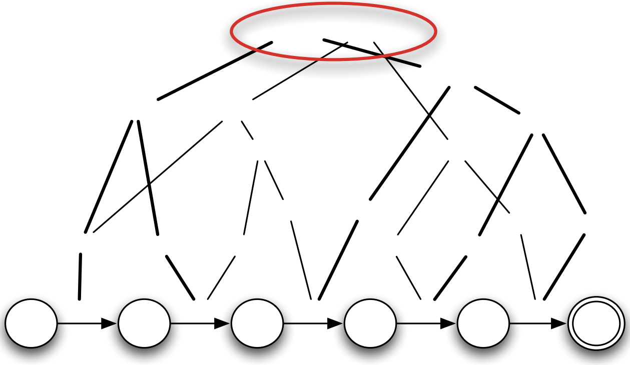
V → porte|voile

N → belle|porte|voile

CL → le

Shared forest as an hypergraph

- The shared forest can be seen as an hypergraph:



Important observation for parsing as a search CKY builds the hypervertices following a topological order on this hypergraph

Plan

1. Parsing context free grammars
    - Parsing as intersection
    - Classical parsing algorithms
        - CKY
        - The Earley algorithm
        - Left corner
2. Parsing TAGs

Earley parser

- During parsing, items are partly bottom-up recognized (left of the dot)
- and top down predicted

$$< A \rightarrow B_1 \dots B_i \bullet B_{i+1} \dots B_n, i, j >$$

Comments

the bullet indicates up to which point the production has been recognized,the indices, indicates the span $w_i \dots w_j$ of the already recognized substring

Algorithm

$w = a_1 \ldots a_n$

- Axioms: (start by predicting from the axiom)

$$\frac{}{< S \rightarrow \bullet\alpha, 0, 0 >} \quad S \rightarrow \alpha \in P$$

- Predict :

$$\frac{< A \rightarrow \alpha \bullet B\beta, i, j >}{< B \rightarrow \bullet\gamma, j, j >} \quad B \rightarrow \gamma \in P$$

- Scan :

$$\frac{< A \rightarrow \alpha \bullet a_j\beta, i, j-1 >}{< A \rightarrow \alpha a \bullet \beta, i, j >} \quad a_j \in w$$

- Complete:

$$\frac{< A \rightarrow \alpha \bullet B\beta, i, j >< B \rightarrow \gamma\bullet, j, k >}{< A \rightarrow \alpha B \bullet \beta, i, k >}$$

- Goal :

$$< S \rightarrow \alpha\bullet, 0, n > \quad \exists S \rightarrow \alpha \in P$$

Exercise

- Parse : Je vois
- with the grammar :

$$
\begin{array}{l}
\mathrm{S} \rightarrow \mathrm{NP}\ \mathrm{VP} \\
\mathrm{NP} \rightarrow \mathrm{Pro} \\
\mathrm{NP} \rightarrow \mathrm{Det}\ \mathrm{N} \\
\mathrm{VP} \rightarrow \mathrm{V} \\
\mathrm{VP} \rightarrow \mathrm{V}\ \mathrm{S} \\
\mathrm{VP} \rightarrow \mathrm{V}\ \mathrm{NP} \\
\mathrm{Pro} \rightarrow \mathrm{je} \\
\mathrm{V} \rightarrow \mathrm{vois}
\end{array}
$$

Exercise (solution)

$< S \rightarrow \bullet NP\, VP, 0, 0 >$ (predict from axiom)

$< NP \rightarrow \bullet Pro, 0, 0 >$ (predict)

$< NP \rightarrow \bullet De \;\; N, 0, 0 >$ (predict)

$< Pro \rightarrow \bullet je, 0, 0 >$ (predict)

$< Pro \rightarrow je \bullet, 0, 1 >$ (scan)

$< NP \rightarrow Pro \bullet, 0, 1 >$ (complete)

$< S \rightarrow NP \bullet \; VP, 0, 1 >$ (complete)

$< VP \rightarrow \bullet V, 1, 1 >$ (predict)

$< VP \rightarrow \bullet V \; S, 1, 1 >$ (predict)

$< VP \rightarrow \bullet V \; NP, 1, 1 >$ (predict)

$< V \rightarrow \bullet pense, 1, 1 >$ (predict)

$< V \rightarrow pense \bullet, 1, 2 >$ (scan)

$< VP \rightarrow V \bullet, 1, 2 >$ (complete)

$< VP \rightarrow V \; \bullet S, 1, 2 >$ (complete)

$< VP \rightarrow V \; \bullet NP, 1, 2 >$ (complete)

$< S \rightarrow NP\, VP \bullet, 0, 2 >$ (complete)

$< S \rightarrow \bullet NP\; PP, 2, 2 >$ (predict)

$< NP \rightarrow \bullet Pro, 2, 2 >$ (predict)

$< NP \rightarrow \bullet De \;\; N, 2, 2 >$ (predict)

$< Pro \rightarrow \bullet je, 2, 2 >$ (predict)

Invariant

Invariant (To advance the dot, recognize the input):

$$< A \to \alpha \bullet \beta, i, j >$$

iff

$$S \overset{*}{\Rightarrow} w_1 \dots w_i A\gamma \Rightarrow w_1 \dots w_i \alpha\beta\gamma \overset{*}{\Rightarrow} w_1 \dots w_i w_{i+1} \dots w_j \beta\gamma$$

for some $\gamma \in (N \cup T*)$

Prefix valid