Logical and Computational Structures for Linguistic Modeling
Part 2 – Parsing CFGs and beyond

Éric de la Clergerie
<Eric.De_La_Clergerie@inria.fr>

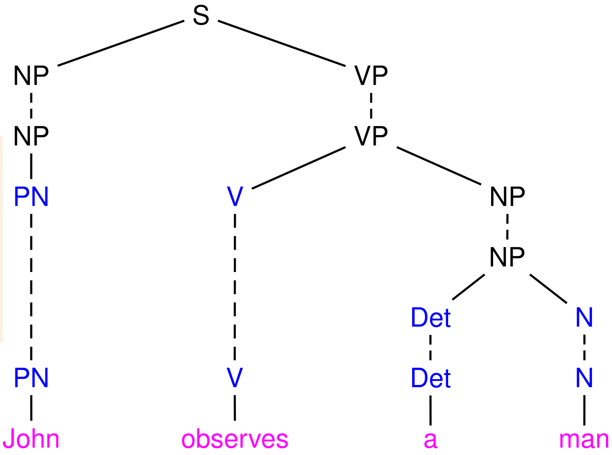
23 Septembre 2014

Part I

Parsing CFGs

Parsing as tree gluing

```
s   ---> np  vp
np  ---> pn
np  ---> det  n
np  ---> np  pp
vp  ---> v  np
vp  ---> vp  pp
pp  ---> prep  np
```



Parsing strategy

Essential to clearly distinguish

- Parsing strategy
- Control strategy

A parsing strategy describes the allowed steps to be tried during parsing

- **top-down** strategies (guided by goals, starting from the axiom)
- **bottom-up** strategies (guided by answers, starting from terminals)
- hybrid strategies (including Earley strategy)
- table-driven strategies (Left Corner, Head Corner, LR, . . .)

Control strategies

A control strategy specifies how to handle non-determinism, especially scheduling:

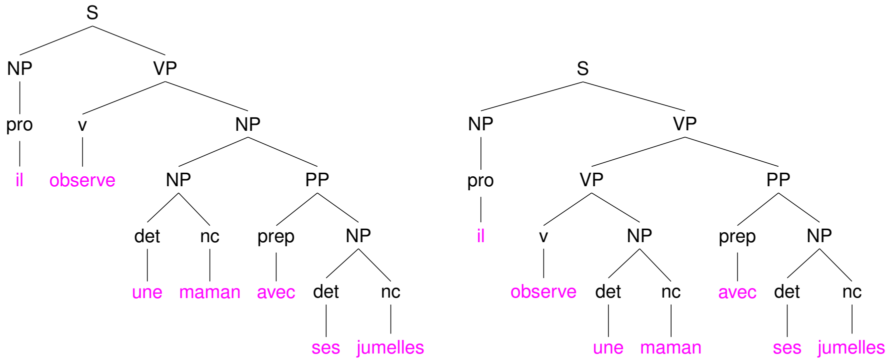
Scheduling In which order perform parsing steps ?

- depth first
- breadth first,
- (left-to-right) string scanning synchronization
- parallel, concurrent, . . .

Ambiguity How to handle ambiguities

- disambiguation (probabilities, heuristics, lookahead),
- backtracking,
- tabulation, . . .

Syntactic ambiguities on PP attachments



for a chain of *k* PPs, exponential number of syntactic trees wrt *k*

> **la Chambre des communes reprendra l'examen du$_1$ projet de$_2$ loi de$_3$ ratification du$_4$ traité de$_5$ Maastricht dès$_6$ la reprise de$_7$ la session du$_8$ soir dans$_9$ la salle principale du$_{10}$ batiment.**

Dynamic Programmig and tabulation

The principles of Dynamic Programming are

1. (recursively) break a problem into smaller ones
2. compute once the (best) solution(s) to the small problems
3. reuse the solution to (recursively) solve larger problems

$\leadsto$ tabulation of the solutions for reuse

Mostly found for optimization problems

- shortest path in a graph
- knapsack problem
- editing distance

But also a long tradition in parsing

DP and CFG parsing

A long story with many algorithms:

- CKY [Cocke-Kasami-Younger]
- Earley algorithm – Chart parsing [Kay]
- Generalized LR [Tomita]
- Stack automata / dynamic programming [Lang]

Outline

Cocke-Kasami-Younger algorithm [CKY]

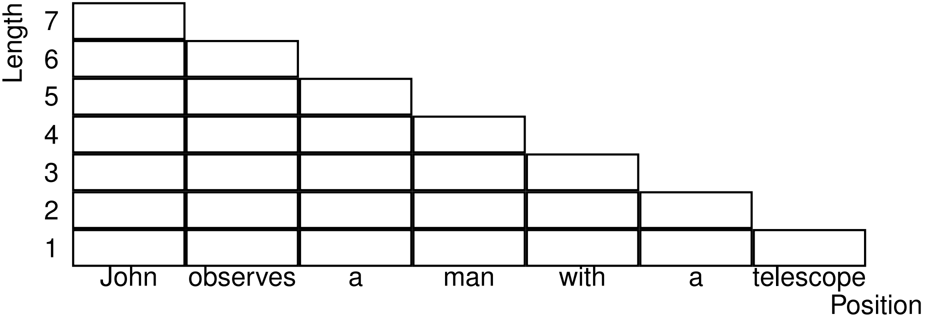
Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

> *If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, A_i present in (x_i, l_i) and $x_{i+1} = x_i + l_i$, then tabulate the non terminal A_0 in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).*

Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

> *If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, A_i present in (x_i, l_i) and $x_{i+1} = x_i + l_i$, then tabulate the non terminal A_0 in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).*

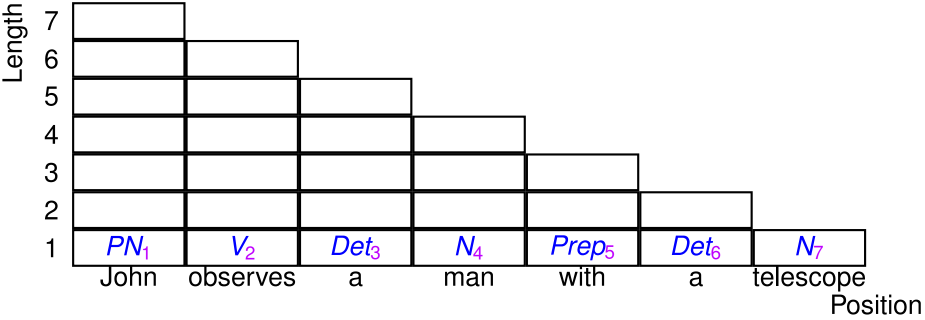


Constituents usually built by increasing length going left-to-right
but actually not mandatory!

Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

> *If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, A_i present in (x_i, l_i) and $x_{i+1} = x_i + l_i$, then tabulate the non terminal A_0 in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).*

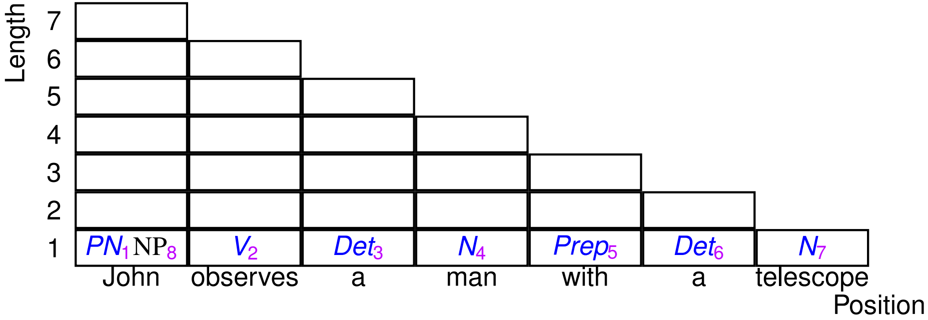


Constituents usually built by increasing length going left-to-right
but actually not mandatory!

Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

> *If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, A_i present in (x_i, l_i) and $x_{i+1} = x_i + l_i$, then tabulate the non terminal A_0 in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).*

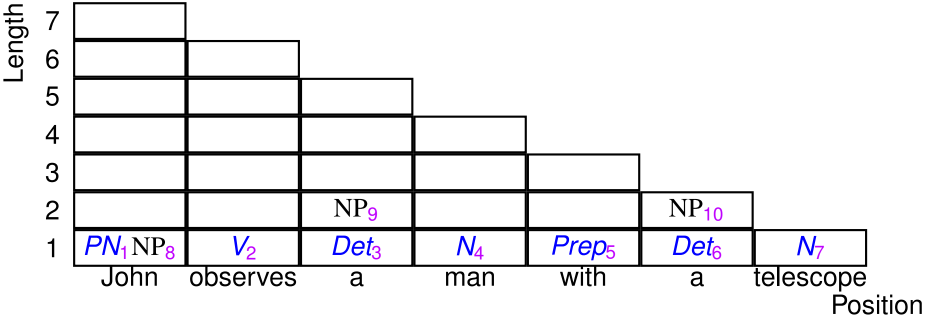


Constituents usually built by increasing length going left-to-right
but actually not mandatory!

Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

If there exists a production $A_0 \leftarrow A_1 \dots A_n$ with, for all $i > 0$, A_i present in (x_i, l_i) and $x_{i+1} = x_i + l_i$, then tabulate the non terminal A_0 in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).

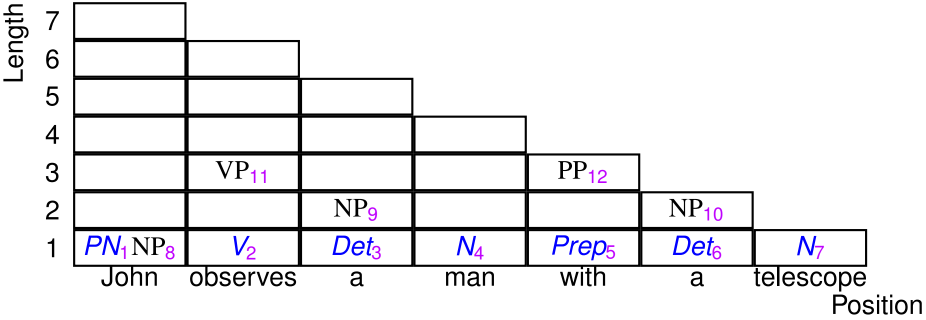


Constituents usually built by increasing length going left-to-right
but actually not mandatory!

Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

> *If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, A_i present in (x_i, l_i) and $x_{i+1} = x_i + l_i$, then tabulate the non terminal A_0 in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).*

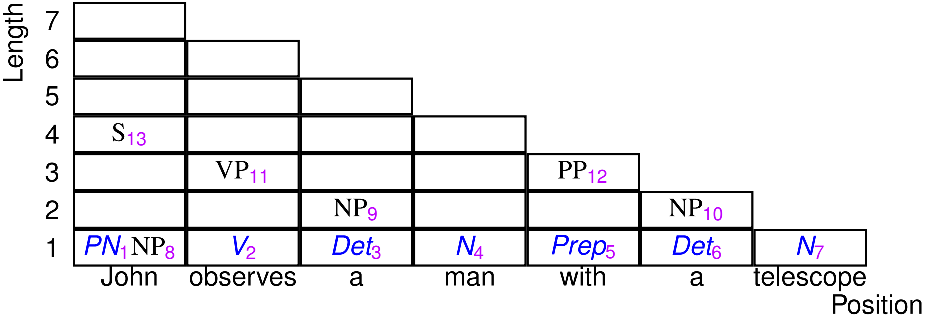


Constituents usually built by increasing length going left-to-right
but actually not mandatory!

Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

> *If there exists a production $A_0 \leftarrow A_1 \dots A_n$ with, for all $i > 0$, A_i present in (x_i, l_i) and $x_{i+1} = x_i + l_i$, then tabulate the non terminal A_0 in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).*



Constituents usually built by increasing length going left-to-right
but actually not mandatory!

Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, A_i present in (x_i, l_i) and $x_{i+1} = x_i + l_i$, then tabulate the non terminal A_0 in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).

Length							
7							
6							
5			NP_{14}				
4	S_{13}						
3		VP_{11}			PP_{12}		
2			NP_9			NP_{10}	
1	PN_1 NP_8	V_2	Det_3	N_4	$Prep_5$	Det_6	N_7
	John	observes	a	man	with	a	telescope

Position

Constituents usually built by increasing length going left-to-right
but actually not mandatory!

Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

> *If there exists a production $A_0 \leftarrow A_1 \dots A_n$ with, for all $i > 0$, A_i present in (x_i, l_i) and $x_{i+1} = x_i + l_i$, then tabulate the non terminal A_0 in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).*

Length							
7							
6		$VP_{15,[16]}$					
5			NP_{14}				
4	S_{13}						
3		VP_{11}			PP_{12}		
2			NP_9			NP_{10}	
1	PN_1 NP_8	V_2	Det_3	N_4	$Prep_5$	Det_6	N_7
Position	John	observes	a	man	with	a	telescope

Constituents usually built by increasing length going left-to-right
but actually not mandatory!

Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

> *If there exists a production $A_0 \leftarrow A_1 \dots A_n$ with, for all $i > 0$, A_i present in (x_i, l_i) and $x_{i+1} = x_i + l_i$, then tabulate the non terminal A_0 in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).*

Length							
7	S_{17}						
6		$VP_{15,[16]}$					
5			NP_{14}				
4	S_{13}						
3		VP_{11}			PP_{12}		
2			NP_9			NP_{10}	
1	PN_1 NP_8	V_2	Det_3	N_4	$Prep_5$	Det_6	N_7
	John	observes	a	man	with	a	telescope

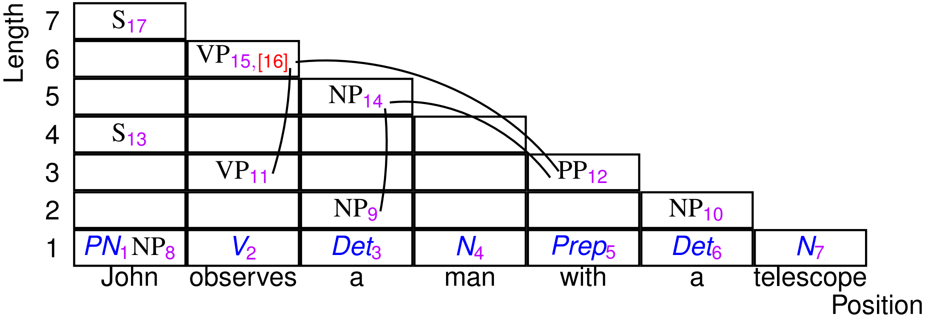
Position

Constituents usually built by increasing length going left-to-right
but actually not mandatory!

Cocke-Kasami-Younger algorithm [CKY]

Dynamic programming algorithm (1965)
Bottom-up parsing strategies with tabulation of constituents

> *If there exists a production $A_0 \leftarrow A_1 \ldots A_n$ with, for all $i > 0$, A_i present in (x_i, l_i) and $x_{i+1} = x_i + l_i$, then tabulate the non terminal A_0 in the entry $(x_1, \Sigma_i l_i)$ (unless already tabulated).*



Constituents usually built by increasing length going left-to-right
but actually not mandatory!

Algorithm and Complexity

```
table_initialize
for all positions x and lengths l
  for all productions A_0 <- A_1 ... A_v
    for all lengths l_1,...,l_{v-1} with Σ_{k=1..v-1} l_k < l
          l_v = l − Σ_{k=1..v-1} l_k
          x_j = x + l_1 + ··· + l_{j-1}
          if A_j ∈ T[x_j, l_j] for all j > 1
          then add A_0 in T[x, l]  (unless present)
```

Worst-case time complexity provided by nested iterations on x, l and l_j ($1 \leq j < v$) bounded by the input string length n.
$\Longrightarrow O(n^{v+1})$ where v is the length of the longest production

For a recognizer, worst-case space complexity given by the number of table cells and number of constituents per cell
$\Longrightarrow O(n^2)$

Chomsky normal form (binarization)

Complexity in $O(n^{v+1})$ reduced to $O(n^3)$ using **Chomsky normal form** (**binarization**).

Ternary rule VP --> V, NP, NP gives a $O(n^4)$ complexity
but may be replaced by following binary rules

```
VP ---> V, VP_ARGS.
VP_ARGS ---> NP, NP.
```

But involves grammar transformation
more elegant to manipulate **dotted rules**.

Worst-case $O(n^3)$ time and $O(n^2)$ space complexities (almost) optimal for CFGs
but CKY not (always) efficient!

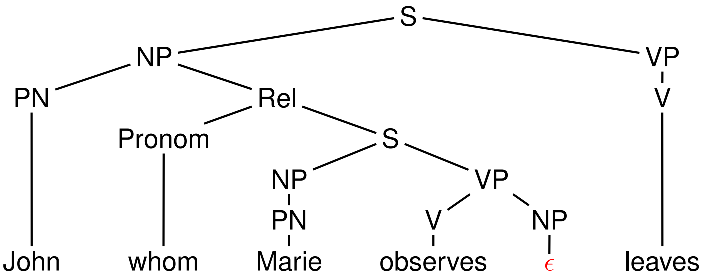
CKY drawbacks: useless computations

Useless constituants

John who looks [$_S$ Marie leaves]

In the model, the longer [$_S$ the word is the less frequent] it is

Trace hypotheses



Outline

Principle

Historically, motivated by the wish to

- use tabulation (for computation sharing)
- preserves optimal complexity $O(n^3)$ for CFGs
- introduce (top-down) prediction

$\leadsto$ development of generic techniques based on **charts**.

CKY as a passive chart algorithm

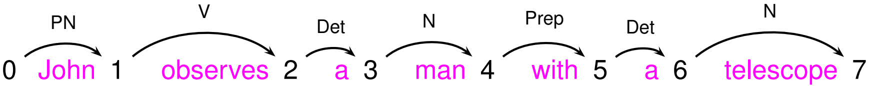
CKY table entries visually represented by edges and stored as **items** $\langle i, j, Cat\rangle$.

0 John 1 observes 2 a 3 man 4 with 5 a 6 telescope 7

Time complexity in $O(n^{v+1})$

CKY as a passive chart algorithm

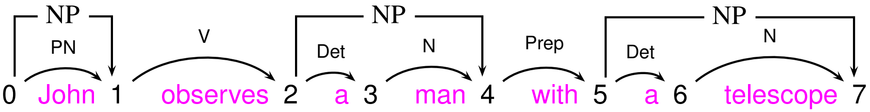
CKY table entries visually represented by edges and stored as **items** $\langle i, j, Cat \rangle$.



Time complexity in $O(n^{v+1})$

CKY as a passive chart algorithm

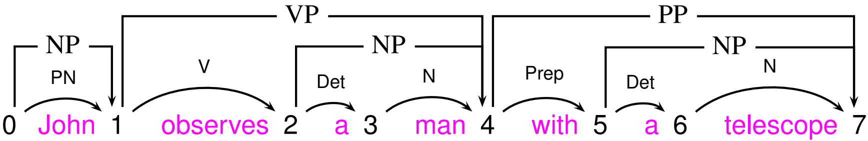
CKY table entries visually represented by edges and stored as **items** $\langle i, j, Cat \rangle$.



Time complexity in $O(n^{v+1})$

CKY as a passive chart algorithm

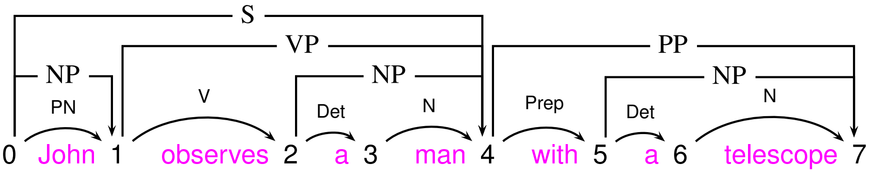
CKY table entries visually represented by edges and stored as **items** $\langle i, j, Cat \rangle$.



Time complexity in $O(n^{v+1})$

CKY as a passive chart algorithm

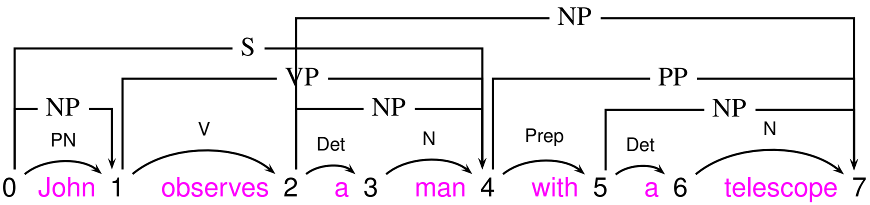
CKY table entries visually represented by edges and stored as **items** $\langle i, j, Cat \rangle$.



Time complexity in $O(n^{v+1})$

CKY as a passive chart algorithm

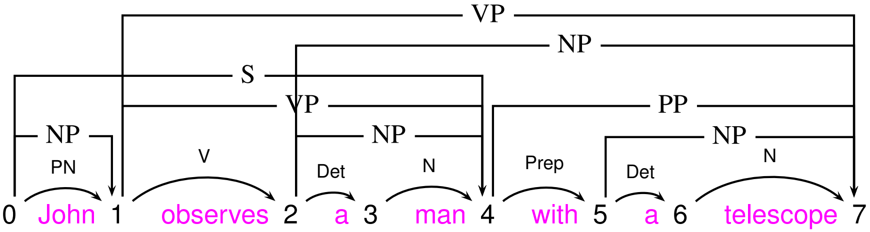
CKY table entries visually represented by edges and stored as **items** $\langle i, j, Cat \rangle$.



Time complexity in $O(n^{v+1})$

CKY as a passive chart algorithm

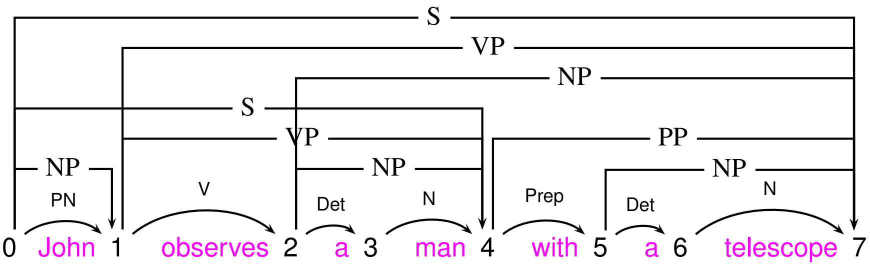
CKY table entries visually represented by edges and stored as **items** $\langle i, j, Cat \rangle$.



Time complexity in $O(n^{v+1})$

CKY as a passive chart algorithm

CKY table entries visually represented by edges and stored as **items** $\langle i, j, Cat \rangle$.



Time complexity in $O(n^{v+1})$

Active chart and dotted rules

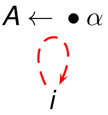
An active chart not only store recognized constituents but also partial ones.

Use of

- **dotted rules** $A_0 \leftarrow A_1 \ldots A_i \bullet A_{i+1} \ldots A_n$
- edges labeled by dotted rules (items $\equiv \langle i, j, A \leftarrow \alpha \bullet \beta \rangle$)
- a deductive system specifying how to derive items

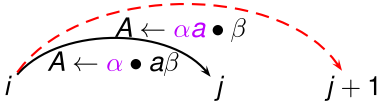
CKY as a deductive system

$$\frac{}{\langle i, i, A \leftarrow \bullet \alpha \rangle} \quad \exists A \leftarrow \alpha$$



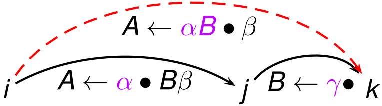
(Seed)

$$\frac{\langle i, j, A \leftarrow \alpha \bullet a\beta \rangle}{\langle i, j+1, A \leftarrow \alpha a \bullet \beta \rangle} \quad a = a_{j+1}$$



(Scan)

$$\frac{\langle i, j, A \leftarrow \alpha \bullet B\beta \rangle \quad \langle j, k, B \leftarrow \gamma \bullet \rangle}{\langle i, k, A \leftarrow \alpha B \bullet \beta \rangle}$$

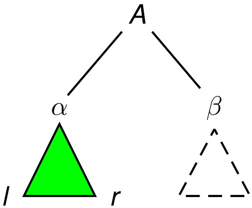


(Complete)

Notion of Parsing as Deduction F. Pereira & D.H.D. Warren

Invariant and Complexity

Each item $\langle l, r, A \leftarrow \alpha \bullet \beta \rangle$ satisfies the invariant: $\alpha \rightarrow^\star a_{l+1} \dots a_r$

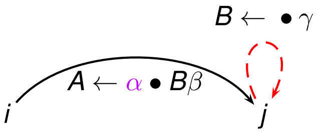


Using dotted rules provides implicit binarization
$\Longrightarrow O(n^3)$ time complexity

Earley algorithm: prediction and active chart

Possibility to use a (top-down) predictive rule $\Longrightarrow$ **Earley algorithm** [1970]

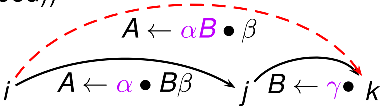
$$\frac{\langle i, j, A \leftarrow \alpha \bullet B\beta \rangle}{\langle j, j, B \leftarrow \bullet \gamma \rangle} \qquad \exists B \leftarrow \gamma$$



(Pred)

$+$ rules (Scan) and (Complete) (but not (Seed))

$$\frac{\langle i, j, A \leftarrow \alpha \bullet B\beta \rangle \quad \langle j, k, B \leftarrow \gamma \bullet \rangle}{\langle i, k, A \leftarrow \alpha B \bullet \beta \rangle}$$

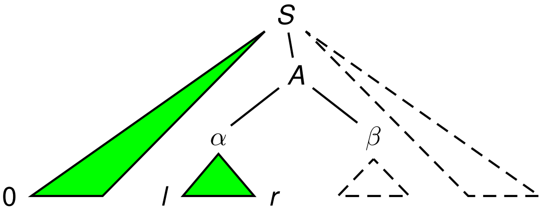


(Complete)

Invariant and complexity

Each item $\langle l, r, A \leftarrow \alpha \bullet \beta \rangle$ satisfies two invariants:

1. Recognition of α between l and r (as for CKY)
2. **prefix validity**: $\exists \gamma \in (\Sigma \cup \mathcal{N})^\star,\ S \rightarrow^\star a_1 \dots a_l A \gamma$



Worst-case time complexity remains $O(n^3)$
But in practice, prediction cuts search space and reduces complexity.

Chart: setup

A chart algorithm relies on:

- a **table** (i.e. chart) where are stored the items, without duplicates.
- an **agenda** where are stored items to be treated

Chart: setup

A chart algorithm relies on:

- a **table** (i.e. chart) where are stored the items, without duplicates.
- an **agenda** where are stored items to be treated

One algorithm step implies:

1. Select an item I in the agenda
2. Unless I already tabulated, store it ; otherwise move to step 1
3. Build new items by combining I with tabulated items
4. Insert the new items in the agenda

Chart: setup

A chart algorithm relies on:

- a **table** (i.e. chart) where are stored the items, without duplicates.
- an **agenda** where are stored items to be treated

One algorithm step implies:

1. Select an item I in the agenda
2. Unless I already tabulated, store it ; otherwise move to step 1
3. Build new items by combining I with tabulated items
4. Insert the new items in the agenda

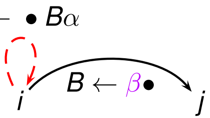
Variant: The items are **first** tabulated then inserted in the agenda
Earley selection order: $\langle i, j, A\rangle$ selected before $\langle k, l, B\rangle$ if $j < l$:
$\Longrightarrow$ left-to-right synchronized scanning

For CFGs, the selection order is not so important (finite universe):
the algorithm terminates and is complete.

Variants

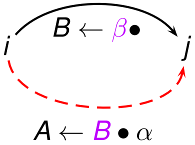
Possibility of better filtering out useless steps (CKY examples)

$$\frac{\langle i, j, B \leftarrow \beta \bullet \rangle}{\langle i, i, A \leftarrow \ \bullet B\alpha \rangle} \qquad \exists A \leftarrow B\alpha \qquad \text{(FilteredSeed)}$$



Possibility to merge several steps (CKY example)

$$\frac{\langle i, j, B \leftarrow \beta \bullet \rangle}{\langle i, j, A \leftarrow B \bullet \alpha \rangle} \qquad \exists A \leftarrow B\alpha \qquad \text{(GreedySeed)}$$



Parsing Schemata

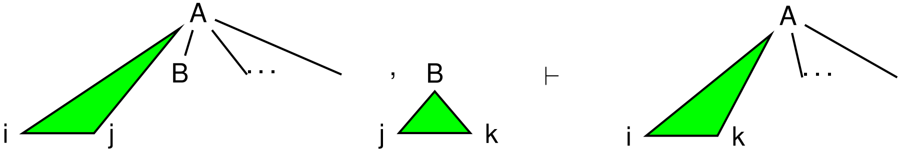
Description of parsing strategies in terms (of classes) of partial parse trees

> [Sikkel] *"These intermediate results are not necessarily partial trees, but they must be objects that denote relevant properties of those partial parses."*

A schema indicates

- the domain of items (and their form)
- the item invariants

Very close from chart algorithms

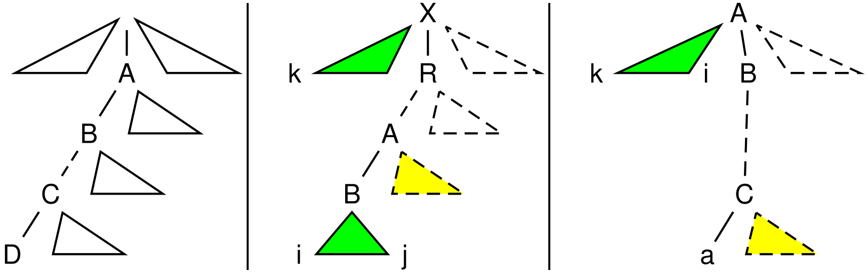


Left-Corner Strategies

$$\frac{\langle i,j,B \leftarrow \beta\bullet\rangle \quad \langle k,i,X \leftarrow \nu \bullet R\mu\rangle}{\langle i,j,A \leftarrow B \bullet \alpha\rangle} \qquad \exists A \leftarrow B\alpha \text{ avec } A \angle R \qquad \text{(LCSeed)}$$

$$\frac{\langle k,i,A \leftarrow \alpha \bullet B\beta\rangle}{\langle i,i+1,C \leftarrow a \bullet \gamma\rangle} \qquad \exists C \leftarrow a\gamma \text{ avec } a_{i+1} = a \angle B \qquad \text{(LCPred)}$$

B, C, D left corners of *A*, denoted by $D \angle A$



Bidirectional scanning: Head driven parsing

Chart parsers not restricted to left-to-right scanning

bidirectional scanning is possible,
for instance with strategies driven by syntactic heads.

- similar to left-corner parsing strategies, except that the head is not necessarily the following word
- mixed top-down & bottom-up parsing strategies
- 2- or 4-positions items
  $\langle l, r, A \leftarrow \alpha \bullet \beta H \gamma \bullet \delta \rangle \qquad \langle l, r, A \leftarrow \alpha \bullet \beta H \gamma \bullet \delta, al, ar \rangle$

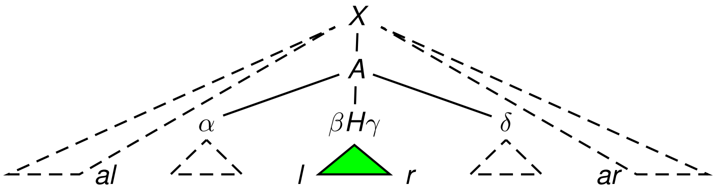


Chart parser limitations

Large variety of items and deductive systems
$\Longrightarrow$ allow coupling tabulation with many parsing strategies

but still difficult with some strategies

Need of a rule like (Complete) using the recognition of a constituent to advance

Characterize bottom-up strategies, with or without some top-down prediction
$\Longrightarrow$ a strictly top-down parsing strategy can't be expressed with a chart parser (or parsing schemata)

Outline

LR parsing strategies

Originally described by Knuth (1965) and mostly used by programming language compilers (YACC, bison) to process deterministically CFG sub-classes [Aho, Ulman, and Hopcroft 1972].

Adapted for non-deterministic CFGs as found for natural languages [GLR – Tomita 1985].

Principle:

- L : Left-to-right scanning
  Scan rightward while the current prefix is a valid one
- R : Right-to-left reduction
  Reduce when a production has been fully recognized

closure and **goto** relations

LR strategy combines left corner and prefix sharing.
Based on the computation of the **closure** and **goto** relations.

closure of $A \leftarrow \alpha \bullet B\beta$ includes all dotted rules $C \leftarrow \bullet \gamma$ with C left corner of B.

goto of $A \leftarrow \alpha \bullet B\beta$ is $A \leftarrow \alpha B \bullet \beta$

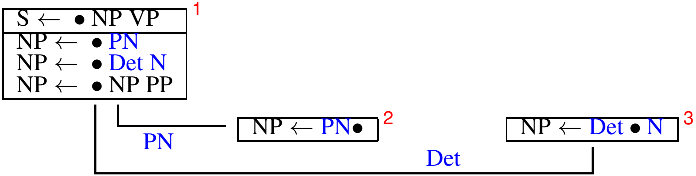
The "**the finite state grammar automaton**" defined by:

- states are closures
- A "goto B" transition exists from S_1 to S_2 if there exists $A \leftarrow \alpha \bullet B\beta \in S_1$ and $A \leftarrow \alpha B \bullet \beta \in S_2$

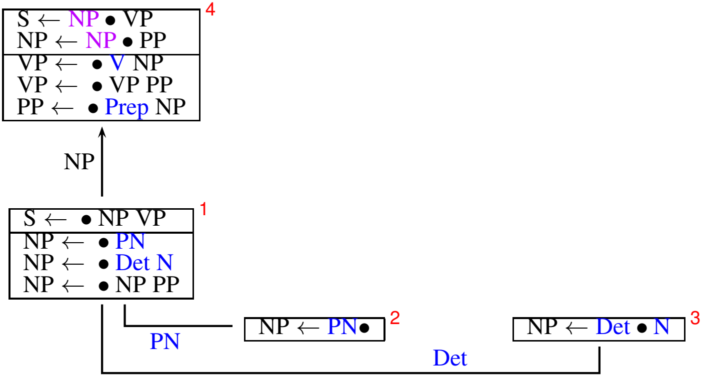
LR automaton

S ← • NP VP	1
NP ← • PN	
NP ← • Det N	
NP ← • NP PP	

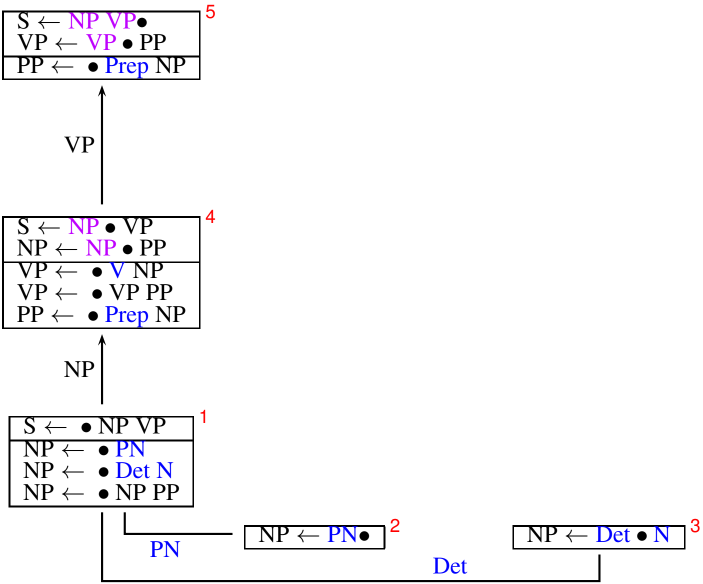
LR automaton



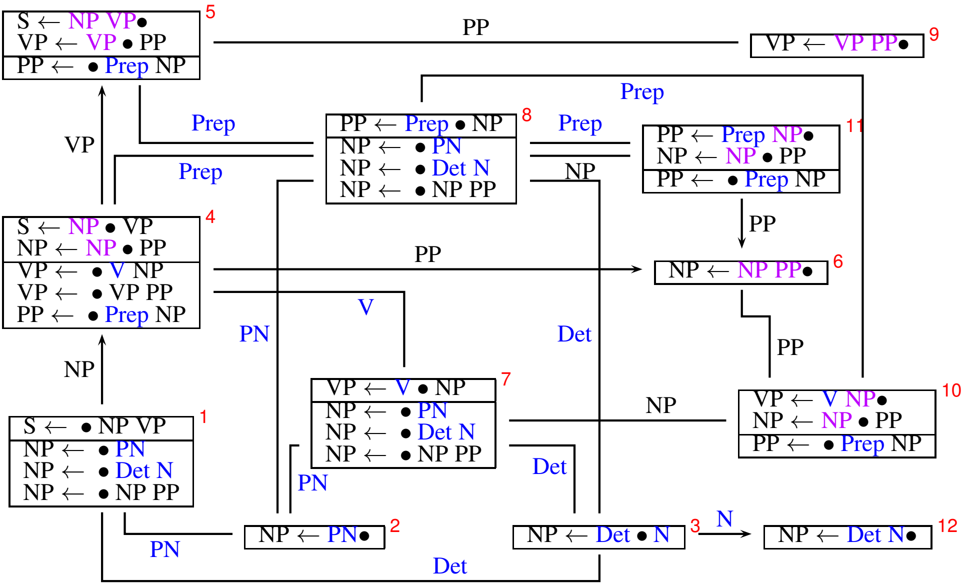
LR automaton



LR automaton



LR automaton



LR tables

Automaton exploited through 2 tables:

- action table: shift (s`<state>`), reduction (r`<prod>`), . . .
- goto table: g`<state>`

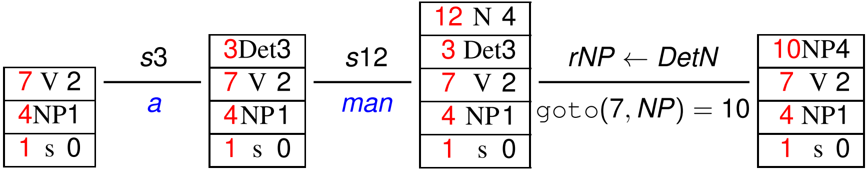
state	Action						Goto			
	PN	DET	N	V	PREP	end	NP	PP	VP	S
1	s2	s3					g4			g0
2	rNP1	rNP1	rNP1	rNP1	rNP1					
3			s12							
4				s7	s8			g6	g5	
5	rS2	rS2	rS2	rS2	s8/rS2			g9		
6	rNP2	rNP2	rNP2	rNP2	rNP2					
7	s2	s3					g10			
8	s2	s3					g11			
9	rVP2	rVP2	rVP2	rVP2	rVP2					
10	rVP2	rVP2	rVP2	rVP2	s8/rVP2			g6		
11	rPP2	rPP2	rPP2	rPP2	s8/rPP2			g6		
12	rNP2	rNP2	rNP2	rNP2	rNP2					

Potential existence of conflicts shift/reduce or reduce/reduce.

Using the LR tables

The LR tables guide the actions of a Push-Down Automata:

- Stacks formed of triples (state, terminal or non-terminal, position).
- A shift action pushes a new state
- A reduce action for a production $P_u : A \leftarrow A_1 \dots A_n$ pops n states and, pushes the state given by "$goto(\sigma, A)$" where σ topmost state



The conflicts could be handled by *backtracking*,
but exponential time complexity and potential loops

GLR Algorithm (Tomita 1985)

Tabular algorithm:

- All alternatives are explored (in case of conflicts)
- Maximum sharing of sub-stacks
  $\Longrightarrow$ **graph-structured stacks** or cactus stacks.

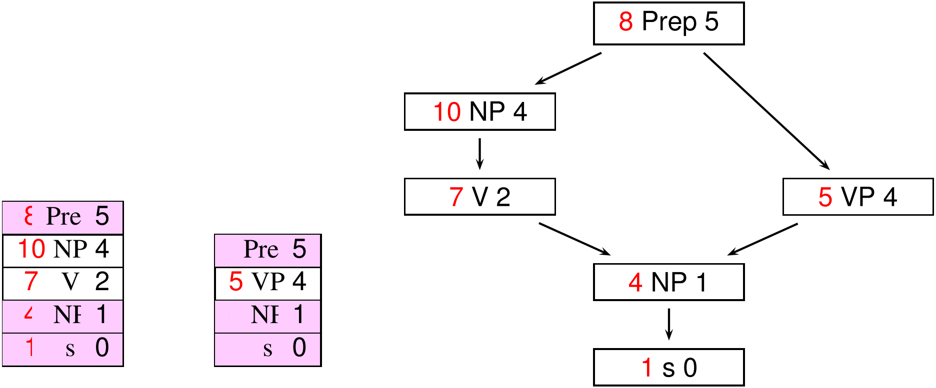
8	Pre	5
10	NP	4
7	V	2
4	NP	1
1	s	0

	Pre	5
5	VP	4
	NP	1
	s	0

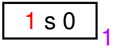
GLR Algorithm (Tomita 1985)

Tabular algorithm:

- All alternatives are explored (in case of conflicts)
- Maximum sharing of sub-stacks
  $\Longrightarrow$ **graph-structured stacks** or cactus stacks.

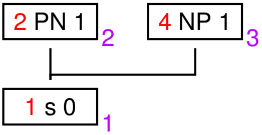


Graph structured stacks



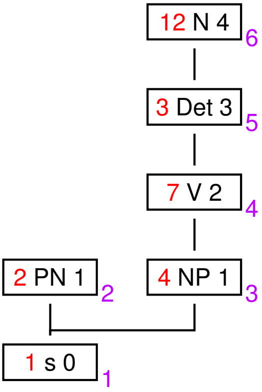
Trace for “John observes a man with a telescope”

Graph structured stacks



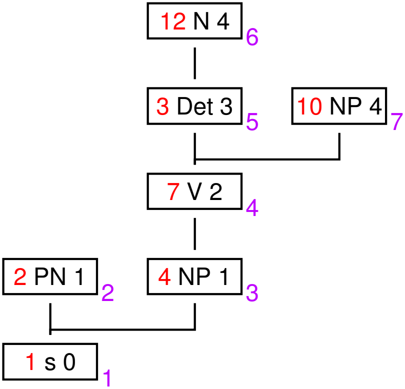
Trace for “John observes a man with a telescope”

Graph structured stacks



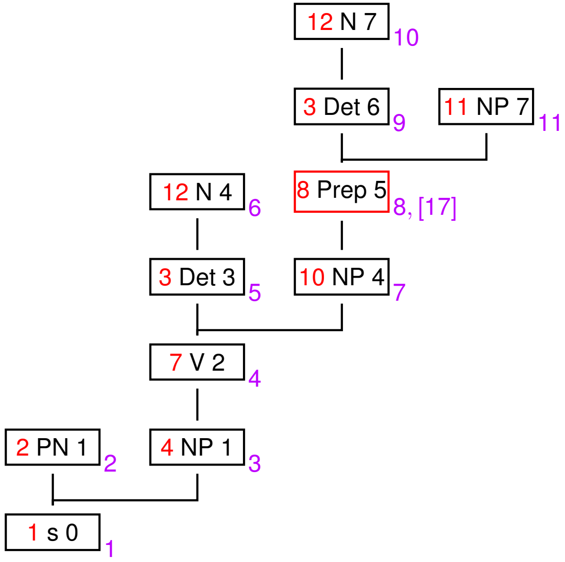
Trace for "John observes a man with a telescope"

Graph structured stacks



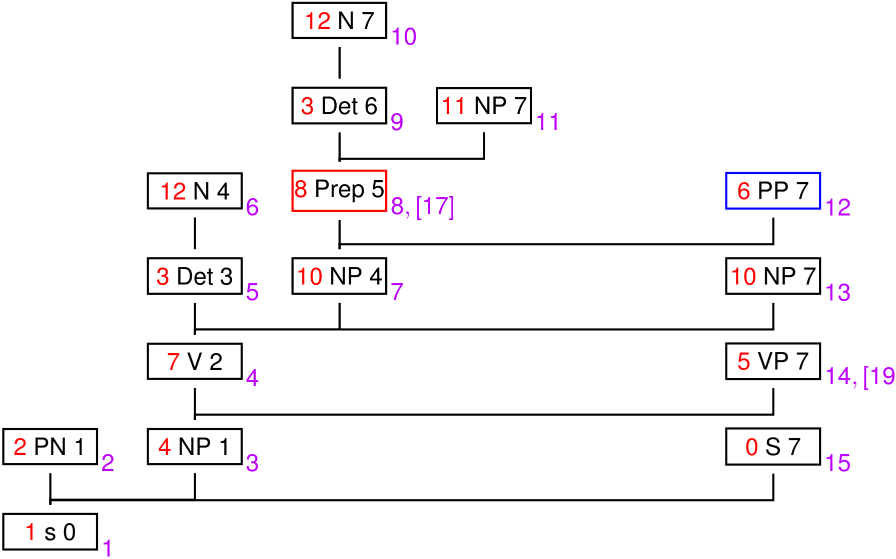
Trace for “John observes a man with a telescope”

Graph structured stacks



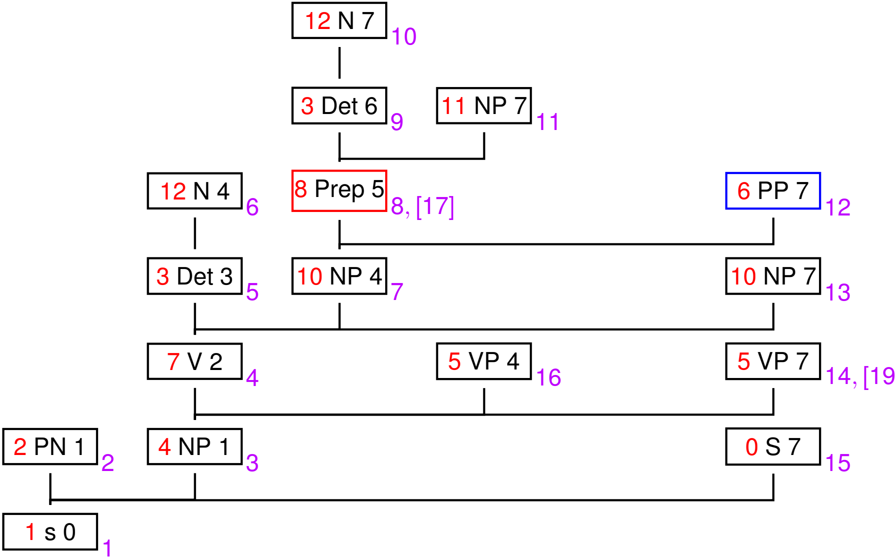
Trace for “John observes a man with a telescope”

Graph structured stacks



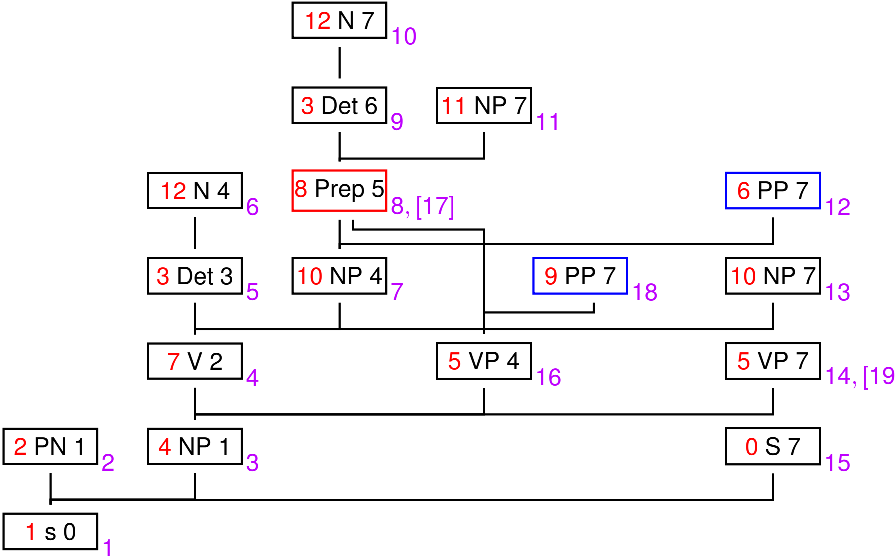
Trace for “John observes a man with a telescope”

Graph structured stacks



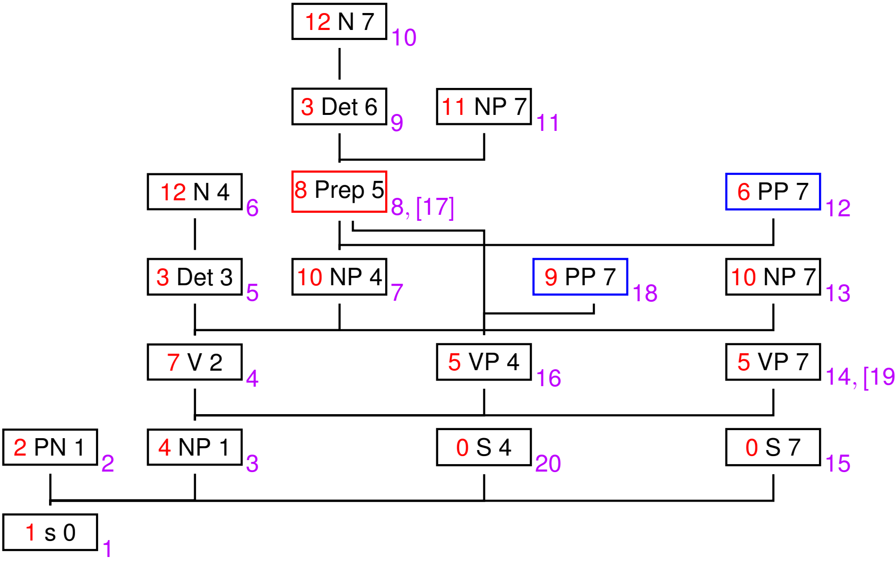
Trace for “John observes a man with a telescope”

Graph structured stacks



Trace for "John observes a man with a telescope"

Graph structured stacks

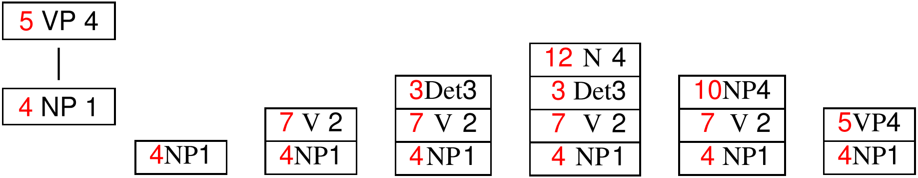


Trace for "John observes a man with a telescope"

Complexity and invariant

- Ensures a time complexity in $O(n^{v+1})$, v length of longest production
- Modifiable to ensure $O(n^3)$ time complexity
- $O(n^2)$ space complexity (for recognizer)
- Variants to handle cyclic grammars

Invariant no-longer expressed in terms of partial parse trees
but of **derivations** of the PDA



Outline

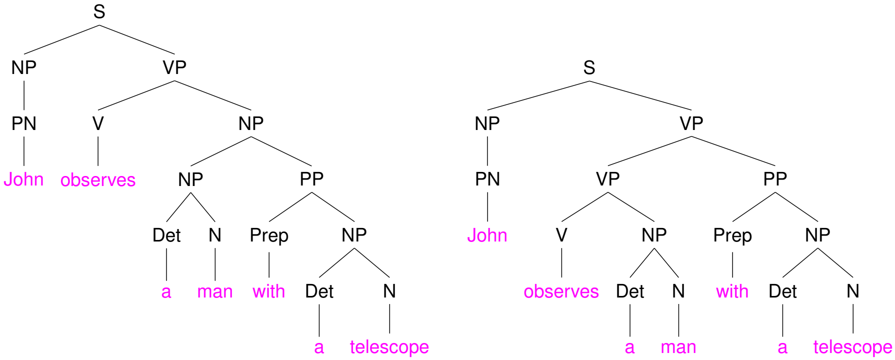
Shared forests

Language ambiguity $\Longrightarrow$
Several possible parses per sentence !

Forest $\equiv$ set of parse trees

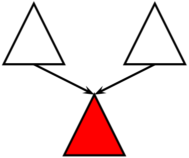
Shared (or packed) forest $\equiv$ Compact forest representation sharing identical or similar sub-trees.

Syntactic ambiguities and sharing

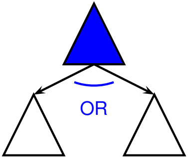


We can observe common subparts

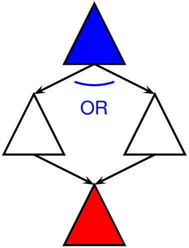
AND-OR graphs



Sharing sub-trees



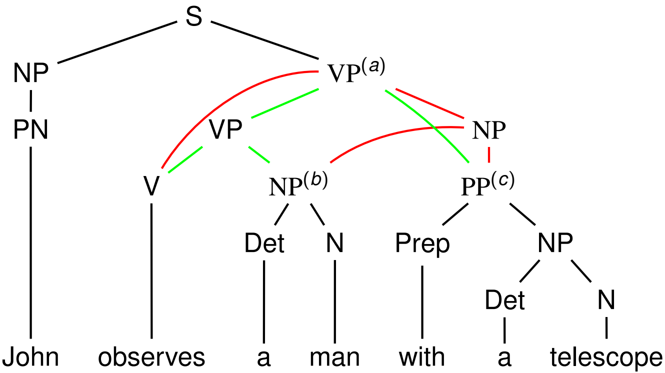
Sharing context



Full sharing

AND-OR graphs may also been formalized as hyper-graphs

Shared forests and trees



Shared forests and grammars

A forest is a grammar G' instance of G [Lang].

0 John 1 observes 2 a 3 man 4 with 5 a 6 telescope 7

```
s  --> np vp
np --> pn
np --> det n
np --> np pp
vp --> v np
vp --> vp pp
pp --> prep np
```

```
s07  --> np01 vp17
np01 --> pn01
vp17 --> v12 np27
vp17 --> vp14 pp47
np27 --> np24 pp47
n37  --> n34 pp47
np24 --> det23 n34
pp47 --> prep45 np57
np57 --> det56 n67
vp14 --> v12 np24
```

```
  pn01 --> John
   v12 --> observes
 det23 --> a
   n34 --> man
prep45 --> with
 det56 --> a
   n67 --> telescope
```

Some non-terminals (vp17) multiply defined (ambiguities).
Some non-terminals (v12,np24,pp47) are used several times (sharing).

Actually, a shared forest is the intersection of a grammar with a regular language (generated by a Finite State Automaton [FSA]).

$$L(G') = L(G) \cap \{\text{“John observes a man with a telescope”}\}$$

Bar Hillel 1964

The intersection of a context free language with a regular language is again a context free language

Intersecting with a FSA

Given a CFG $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ and a FSA $A = (\mathcal{Q}, \Sigma, \delta, I, F)$,
we construct $G_\cap = (\mathcal{N} \times \mathcal{Q} \times \mathcal{Q}, \Sigma \times \mathcal{Q} \times \mathcal{Q}, \langle S, I, F \rangle, \mathcal{P}')$
For each $A_0 \leftarrow A_1 \dots A_n \in \mathcal{P}$, add to $\mathcal{P}'$

$$\langle A_0, q_o, q_n \rangle \leftarrow \langle A_1, q_0, q_1 \rangle \dots \langle A_n, q_{n-1}, q_n \rangle$$

with

$$\forall i \in \{1, \dots, n-1\}, \begin{cases} A_{i+1} \in \mathcal{N} \implies (q_i, q_{i+1}) \in \mathcal{Q} \\ A_{i+1} \in \Sigma \implies (q_i, A_{i+1}, q_{i+1}) \in \delta \\ A_{i+1} = \epsilon \implies q_i = q_{i+1} \end{cases}$$

We show that

$$L(G) \cap L(A) = L(G_\cap)$$

Construction in time $O(|G|.|\mathcal{Q}|^{n+1})$, where n is length of longest clause.

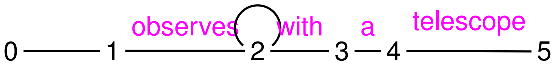
Many useless productions in $\mathcal{P}'$
$\implies$ need grammar reduction: removal of non reachable clauses from axiom
$\sim$ parsing !

Input strings vs Finite State Automata

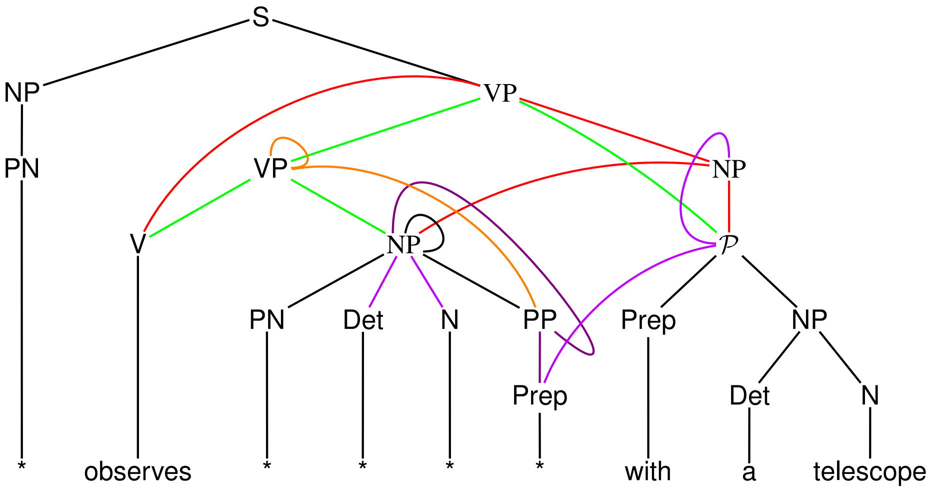
For parsing, input string may be replaced by input FSAs

$$L(G') = L(G) \cap L(FSA)$$

“[unreadable word] observes [unreadable words] with a telescope”



Parse forest for an incomplete sentence



Grammar for an incomplete sentence

```
 s05 --> np01   vp15           pn01 --> *
np01 --> pn01                   v12 --> observes
vp15 --> v12    np25           pn22 --> *
vp15 --> vp12   pp25          det22 --> *
np25 --> np22   pp25            n22 --> *
vp12 --> vp12   pp22         prep22 --> *
vp12 --> v12    np22         prep23 --> with
pp25 --> prep22 np25          det34 --> a
pp25 --> prep23 np35            n45 --> telescope
np22 --> np22   pp22
np22 --> det22  n22
np22 --> pn22
pp22 --> prep22 np22
np35 --> det34  n45
```

FSAs

Tabular parsers easily modifiable to take as input an FSA (or a word lattice)

Parsing an FSA done with time complexity in $O(n^3)$ for CFGs where n is the number of states of the FSA.

FSAs (or word lattice) useful for

- noisy or incomplete sentences (speech data)
- lexical ambiguities
- segmentation ambiguities

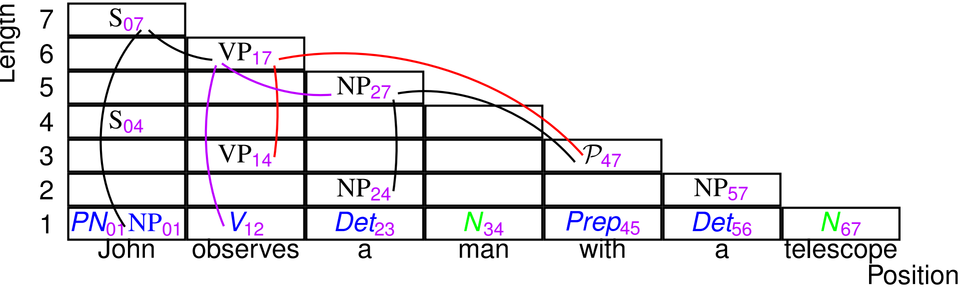
FSAs Possibly with probabilities or weights (weighted FSAs)

Same results extend for most grammatical formalisms (Unification grammars,TAGs, LIGs,. . .)

Shared forests may be built or extracted post parsing.

Extraction uses backpointers from tabulated object to their parents

> *Starting from answer (`S07`), backpointers are followed to retrieve instantiated productions and identify non-terminals*



Note: Space complexity increases from $O(n^2)$ to $O(n^3)$ for binarized grammars

Part II

Towards Unification-based grammars

CFG in practice

CFG are not really adequate for fine-grained descriptions !

```
s   --> np vp
np --> pn
np --> det n
np --> np pp
vp --> v np
vp --> vp pp
pp --> prep np
s --> vp  %% imperative
```

How to rule out ?

- ⋆**il manges les pomme**
- ⋆**mangeront la pomme**

Duplicate CFG rules !

```
s  --> np_p1_sing vp_p1_sing
s  --> np_p1_pl vp_p1_pl
s  --> np_p2_sing vp_p2_sing
s  --> np_p2_pl vp_p2_pl
s  --> np_p3_sing vp_p3_sing
s  --> np_p3_pl vp_p3_pl
```

but also

```
np_p3_sing  --> det_masc_sing n_masc_sing
np_p3_sing  --> det_fem_sing n_fem_sing
np_p3_pl  --> det_masc_pl n_masc_pl
np_p3_pl  --> det_fem_pl n_fem_pl
```

and

```
s  --> vp_imperative
vp_imperative --> v_imperative np
```

actually, need to combine all these bits of informations

$\Longrightarrow$ greatly increase the number of relatively similar productions

Underspecified rules

Using *underspecified rules* with variables ranging over (finite) set of values

```
s  --> np(P,G,N) vp((P,N,M).
s --> vp(P,imperative).
np(3,G,N) --> det(G,N) n(G,N).
```

Alternate notations

```
s  -->
    np{ person => P, number => N },
    vp{ person => P, number => N }.
s --> vp{ mood => imperative }.
np{ person => 3, gender => G, number => N } -->
      det{ gender => G, number => N},
      n{ gender => G, number => N}.
```

The abstracted rules and possible instantiations may be used to generate CFG rules, but large number of CFG rules

Also, wish of richer instantiations, with no finite expansion
$\Longrightarrow$ Better to move to Unification Grammars

Unification Grammars

Decorated non-terminals
but no fundamentally different rule applications

CFG	Datalog	Horn Clauses DCG LFG HPSG	
NP	V(sing)	S(gap(np))	λ-Prolog

literal complexity

CFG productions & Horn clauses are very similar

$$\mathrm{S} \leftarrow \mathrm{NP}\ \mathrm{VP} \leadsto \mathrm{S}(X_0, X_2) \mathrel{:-} \mathrm{NP}(X_0, X_1)\ \mathrm{VP}(X_1, X_2).$$

- Allow information propagation from one point to another
  logical variables, reentrancy
- Allow underspecification (partial information)

Outline

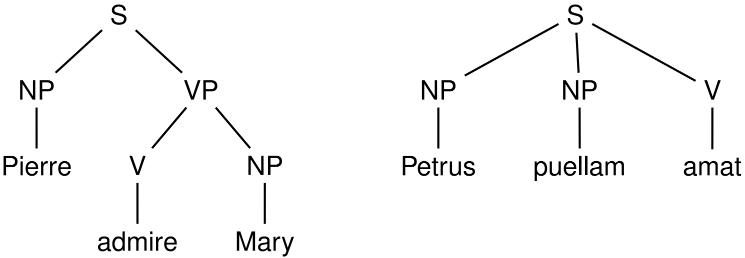
5 LFG and Feature Structures

6 Charts revisited for Unification Grammars

7 Push-Down Automata

Lexical Functional Grammars [LFG]

Bresnam et Kaplan (1982) *The mental representation of grammatical representation*
Théorie: Associate constituent structures (*c-structures*) & functional structures (*f-structure*):

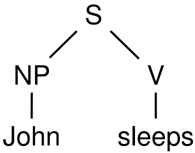


$$\begin{bmatrix} \text{Subj} & \begin{bmatrix} \text{Pred} & \text{pierre} \end{bmatrix} \\ \text{Pred} & \text{admirer} \\ \text{Obj} & \begin{bmatrix} \text{Pred} & \text{mary} \end{bmatrix} \end{bmatrix} \qquad \begin{bmatrix} \text{Subj} & \begin{bmatrix} \text{Pred} & \text{pierre} \end{bmatrix} \\ \text{Pred} & \text{admirer} \\ \text{Obj} & \begin{bmatrix} \text{Pred} & \text{mary} \end{bmatrix} \end{bmatrix}$$

Productions & functional equations

A grammar is given as CFG productions whose non-terminals are decorated by functional equations.

S $\longrightarrow$	NP ($\uparrow$Subj)=$\downarrow$	V $\uparrow$=$\downarrow$	NP $\longrightarrow$	John ($\uparrow$Num)= sing ($\uparrow$Gender)= masc ($\uparrow$Pred)= 'John'	V $\longrightarrow$	sleeps ($\uparrow$Subj Num)= sing ($\uparrow$Subj Pers)= 3 ($\uparrow$Mood)= indicative ($\uparrow$Pred)= 'sleep<Subj>'



Subj	Pred 'John'	
	Gender masc	
	Num sing	
Pred 'sleeps<Subj>'		
Mood indicative		

Formalism: Feature structure

FS may been seen as property-value records,

- possibly with FS as values (recursion)
- possibly with reentrency (shared FS)

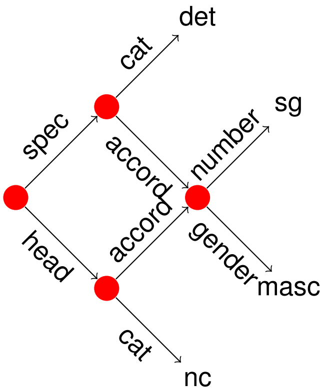
Generally represented as Attribute Value Matrix

$$\begin{bmatrix} \text{Det} & \begin{bmatrix} \text{Agr} & \boxed{1} \begin{bmatrix} \text{Num sing} \\ \text{Gender masc} \end{bmatrix} \\ \text{Cat Det} & \end{bmatrix} \\ \text{Nom} & \begin{bmatrix} \text{Agr} & \boxed{1} \\ \text{Cat N} & \end{bmatrix} \end{bmatrix}$$

Graph notation

Feature structures may be formalized as acyclic directed graphs (maybe extended with cycles)

$$\begin{bmatrix} \text{spec} & \begin{bmatrix} \text{accord} & \boxed{1} \begin{bmatrix} \text{number} & \text{sg} \end{bmatrix} \\ \text{cat} & \text{det} \end{bmatrix} \\ \text{head} & \begin{bmatrix} \text{accord} & \boxed{1} \begin{bmatrix} \text{gender} & \text{masc} \end{bmatrix} \\ \text{cat} & \text{nc} \end{bmatrix} \end{bmatrix}$$



Note: leads to a notion of path for a sequence of features, in graph and AVM
ex: chemin spec.accord.gender

FS: formalization

We suppose given a signature $S = (V, F)$ where F is a finite set of properties/features and V a set of atomic values

Formally, a FS A over S is denoted by

$$(Q_A, r_A, \delta_A, \theta_A)$$

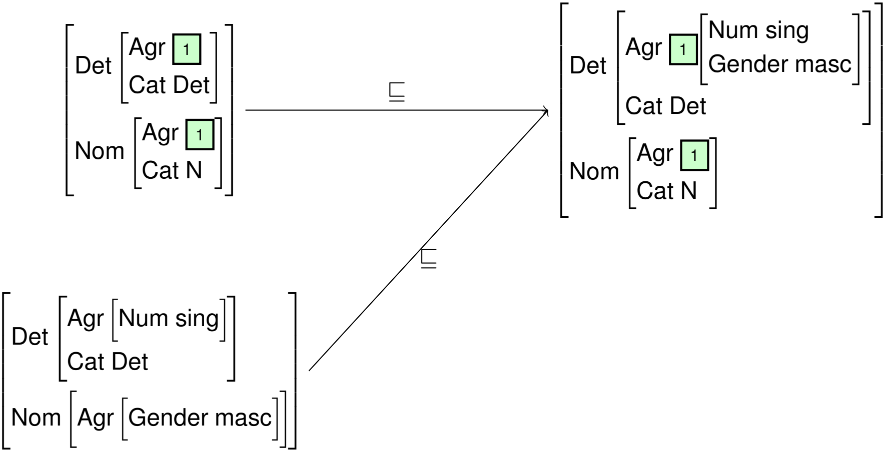
where:

- Q_A is a set of state
- $r_A \in Q_A$ is the root state
- $\delta_A : Q_A \times F \leftarrow Q_A$ is a partial function for following features such that each state in Q_A is reachable from r_A by reflexive-transitive closure of δ_A
  i.e. $\forall q \in Q_A,\ q = r_A \vee \exists(q', f),\ \delta(q', f) = q$
- $\theta_A : Q_A \leftarrow V$, a partial labeling function
  only defined on terminal states, ie $q \in Q_A, \forall f \in F, \delta(q, f) \uparrow$

Path $\pi(A)$ defined as $\{p \in F^\star | \delta(r_A, p) \downarrow\}$
$p_1 \neq p_2$ are 2 reentrant paths iff $\delta(r_A, p_1) = \delta(r_A, p_2) \downarrow$

FS Subsumption

FS may be seen as *specifying information* (properties of entities)
$\leadsto$ subsumption order $A \sqsubseteq B$ if A more general than B
or alternatively A less constraint than B
$\Longrightarrow \sqsubseteq$ is a partial pre-order on feature structures



FS subsumption: algorithm

Sketch of an algorithm:

$A \sqsubseteq B$ iff for each path p in A, there exists a path p.q in B
but beware of reentrency !

FS unification

Unification accumulates partial information:

$$\begin{bmatrix} \text{Det} & \begin{bmatrix}\text{Agr} & \begin{bmatrix}\text{Num} & \text{sing}\end{bmatrix}\end{bmatrix} \\ \text{Nom} & \begin{bmatrix}\text{Agr} & \begin{bmatrix}\text{Num} & \text{sing}\end{bmatrix} \\ \text{Cat} & \text{N}\end{bmatrix}\end{bmatrix} \sqcup \begin{bmatrix}\text{Det} & \begin{bmatrix}\text{Agr} & \begin{bmatrix}\text{Gender} & \text{masc}\end{bmatrix}\end{bmatrix}\end{bmatrix} = \begin{bmatrix} \text{Det} & \begin{bmatrix}\text{Agr} & \begin{bmatrix}\text{Num} & \text{sing} \\ \text{Gender} & \text{masc}\end{bmatrix}\end{bmatrix} \\ \text{Nom} & \begin{bmatrix}\text{Agr} & \begin{bmatrix}\text{Num} & \text{sing}\end{bmatrix} \\ \text{Cat} & \text{N}\end{bmatrix}\end{bmatrix}$$

$$\begin{bmatrix} \text{Det} & \begin{bmatrix}\text{Agr} & \boxed{1}\begin{bmatrix}\text{Num} & \text{sing}\end{bmatrix}\end{bmatrix} \\ \text{Nom} & \begin{bmatrix}\text{Agr} & \boxed{1} \\ \text{Cat} & \text{N}\end{bmatrix}\end{bmatrix} \sqcup \begin{bmatrix}\text{Det} & \begin{bmatrix}\text{Agr} & \begin{bmatrix}\text{Gender} & \text{masc}\end{bmatrix}\end{bmatrix}\end{bmatrix} = \begin{bmatrix} \text{Det} & \begin{bmatrix}\text{Agr} & \boxed{1}\begin{bmatrix}\text{Num} & \text{sing} \\ \text{Gender} & \text{masc}\end{bmatrix}\end{bmatrix} \\ \text{Nom} & \begin{bmatrix}\text{Agr} & \boxed{1} \\ \text{Cat} & \text{N}\end{bmatrix}\end{bmatrix}$$

Formally, most general instance of A and B

$$A \sqcup B = C, \text{ such that } \forall D,\ A \sqsubseteq D \land B \sqsubseteq D \implies C \sqsubseteq D$$

Typed Feature Structures

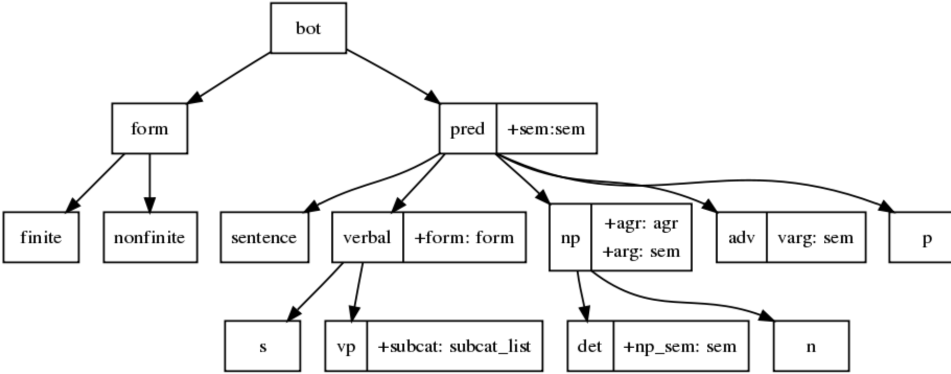
Formalized by B. Carpenter and used in HPSG (*Head-driven Phrase Structure Grammars*.

FS are typed, with types τ in some finite multiple-inheritance hierarchy

- τ may have several parents
- τ may introduce authorized features f,
  with most general appropriate type $\rho_{\tau,f}$ for values
- τ may further instanciate a feature introduced by an ancestor

Type hierarchy

Fragment of a type hierarchy
(*Semantic-Head-Driven Generation*, Shieber et al, in ALE)



LFG: richer equations

Constraints on existence

NP $\longrightarrow$	(Det)	N	
	$\uparrow=\downarrow$	$\uparrow=\downarrow$	

N $\longrightarrow$	Jean
	$\sim(\uparrow$Det)

N $\longrightarrow$	chien
	($\uparrow$Det)

Det $\longrightarrow$	le
	($\uparrow$Det)=le

Constraint equations

S' $\longrightarrow$	NP	S
	$(\downarrow \text{Wh}) =_c +$	($\uparrow$Wh)=+
		$\uparrow=\downarrow$

Set equations

VP $\longrightarrow$	V	(NP)	(PP)*
	$\uparrow=\downarrow$	$\uparrow$Obj=$\downarrow$ ou $\uparrow$Adjunct$\ni\downarrow$	$\uparrow$Adjunct$\ni\downarrow$

(Jean dort le matin. Jean mange le gateau Jean mange ce gateau avec Anne)

Grammatical functions

Possible functions: Suject, Object, Comp(letive), XComp (infinitives and participiales), Prep-Obj (prepositional complements)

Vcomp	Jean veut **partir à Rio**.
Acomp	Jean devient **fou**.
Ncomp	Ils ont élu Jean **président**
Vajout	**Partant en voyage**, Marie se prépare
Aajout	Paul est parti **content**
Prep-Obj	Paul ressemble **à Jean**

LFG: sub-categorization

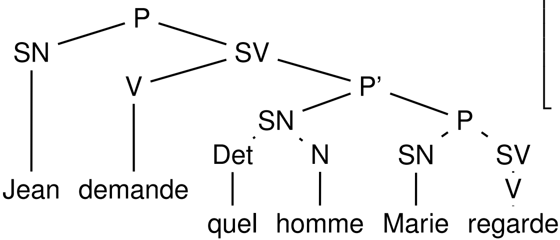
The `Pred` feature states the expected functions for a word

- **manger** (↑Pred)='manger<Suj,Obj>'
- **donner** (↑Pred)='donner<Suj,Obj,A-Obj>'
- **falloir** (↑Pred)='falloir<Obj>Suj' et (↑Suj Form) $=_c$ il
- **vouloir** (↑Pred)='vouloir<Suj,Vcomp>' et (↑Suj)=(↑Vcomp Suj) Jean veut venir
- **proposer** (↑Pred)='proposer<Suj,A-Obj,Vcomp>' et (↑Vcomp Suj)=(↑Suj)/(↑A-Obj)
  Jean propose à Jean de venir
- **destruction** (↑Pred)='destruction<De-Obj,Par-Obj>' Destruction de la maison par les promoteurs

P' $\longrightarrow$	SN	P
	$(\downarrow \text{Qu}) =_c +$	$\uparrow = \downarrow$
	$(\uparrow \text{Focus}) = \uparrow$	$(\downarrow \text{Qu}) = +$
	$(\uparrow \text{Focus}) = (\uparrow \text{Obj})$	

demande,V: $(\uparrow \text{Pred})$='demander<Suj,Comp>'
$(\uparrow \text{Comp Qu}) =_c +$

quel,Det: $(\uparrow \text{Det})$='quel'
$(\uparrow \text{Qu}) = +$



[Suj [Pred 'Jean']
Pred 'demander<Suj,Comp>'
Comp [Focus 1 [Pred 'homme'
Det 'quel']
Suj [Pred 'Marie']
Qu +
Pred 'regarder<Suj,Obj>'
Obj @1]]

Arbitrary number of embeddings between an extract constituent and its associated predicates:

Jean demande [quel homme Paul pense [que Marie regarde ϵ]]

S' $\longrightarrow$ NP S

NP:
(↓Wh) $=_c$ +
(↑Focus) = ↑
(↑Focus)=↑$(\mathrm{Comp})^\star\mathrm{Obj}$

S:
↑=↓
(↓Wh)=+

$$
\left[\begin{array}{ll}
\text{Suj} & \left[\text{Pred 'Jean'}\right] \\
\text{Pred 'demander<Suj,Comp>'} & \\
\text{Comp} & \left[\begin{array}{ll}
\text{Focus} & \boxed{1}\left[\begin{array}{l}\text{Pred 'homme'} \\ \text{Det 'quel'}\end{array}\right] \\
\text{Wh +} & \\
\text{Pred 'penser<Suj,Comp>'} & \\
\text{Suj} & \left[\text{Pred 'Paul'}\right] \\
\text{Comp} & \left[\begin{array}{ll}
\text{Pred 'regarder<Suj,Obj>'} & \\
\text{Suj} & \left[\text{Pred 'Marie'}\right] \\
\text{Obj} & \boxed{1}
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Power of Unification Grammars

Very easy for Unification Grammars to have the power of a Turing machine !

Essentially, because of recursive feature structures

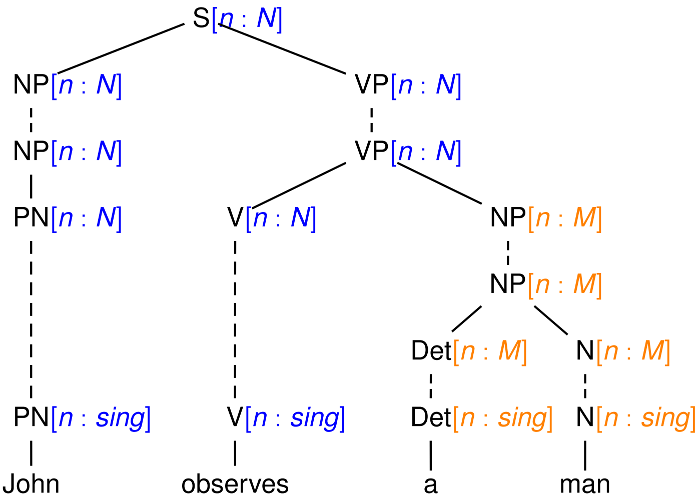
Nevertheless, interesting to explore parsing algorithms for UG

Note: actually, decorations and unification may be added to most base formalism

Outline

Evaluation strategy & tree traversal

Unification (*N&sing*) is used to glue partial parse trees
Existence of information flow propagated thanks to substitutions (*N/sing*)



Earley revisited

In inference rules, **unification** used to combine items.

$$\frac{\langle i, j, A \leftarrow \alpha \bullet B\beta\rangle \quad \langle j, k, C \leftarrow \gamma\bullet\rangle}{\langle i, k, (A \leftarrow \alpha B \bullet \beta)\sigma\rangle} \qquad \sigma = \mathrm{mgu}(B, C) \qquad \text{(Complete)}$$

$$\frac{\langle i, j, A \leftarrow \alpha \bullet B\beta\rangle}{\langle j, j, (C \leftarrow \ \bullet\gamma)\sigma\rangle} \qquad \exists C \leftarrow \gamma \text{ and } \sigma = \mathrm{mgu}(B, C) \qquad \text{(Pred)}$$

$$\frac{\langle i, j, A \leftarrow \alpha \bullet a\beta\rangle}{\langle j, j+1, A \leftarrow \alpha a \bullet \beta\rangle} \qquad a = a_{j+1} \qquad \text{(Scan)}$$

Variable renaming

Renaming of item variables before rule application

- Traditionally, productions are renamed before use (Prolog)

$$\frac{q(X) \leftarrow \bullet q(f(X))}{q(f(X)) \leftarrow \bullet q(f(f(X)))} \quad \exists q(X') \leftarrow q(f(X')) \text{ et } \sigma = \{X'/f(X)\} \qquad \text{(Pred)}$$

  Failure if no renaming of X/X' in production $q(X) \leftarrow q(f(X))$

- But require also item renaming, for instance for (Complete)

Redundancy test: variance & subsumption

Item redundancy checking by simple identity not longer enough because of renaming ($q(X) \neq q(X')$)

Need more powerful redundancy checking

Variance Items identical modulo variable renaming
$q(X)$ variant of $q(X')$.

Subsumption Logical terms are ordered by $\preceq$

$$A \preceq B \iff \exists \sigma,\ B = A\sigma \qquad \begin{cases} A \text{ generalizes } B \\ A \text{ subsumes } B \\ B \text{ is an instance of } A \end{cases}$$

Examples: $g(X, Y) \preceq g(Z, Z) \preceq g(f(a), f(a))$

An item is not tabulated if it is an instance of an already tabulated item

Justification: Each item J' derivable from I' instance of I is instance of some item J derivable from I.

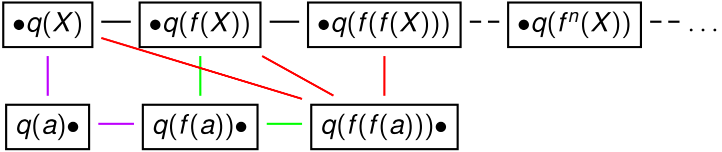
Loops: variance vs subsumption

For the program

```
q(X) :- q(f(X)).
q(f(f(a))).
```

and goal $?-q(X)$, the expected answers are: $X = f(f(a))$, $X = f(a)$, $X = a$

Loops with variance test (+ computation duplication)



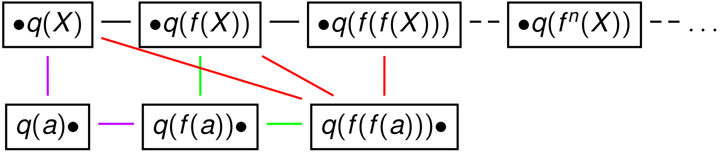
Loops: variance vs subsumption

For the program

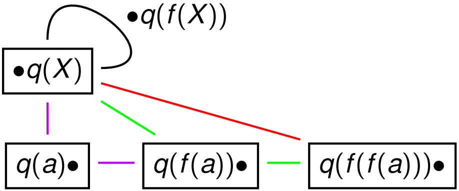
```
q(X) :- q(f(X)).
q(f(f(a))).
```

and goal $?-q(X)$, the expected answers are: $X = f(f(a)),\ X = f(a),\ X = a$

Loops with variance test (+ computation duplication)



Terminates when using subsumption



Trade-of between simple variance and more precise and costly subsumption

Termination & spirals

Termination not always ensured, even using subsumption.

The item family with growing sub-terms $f(a), f(f(a)), \ldots, f^n(a)$ not cut by subsumption (spiral case)

First remedy: only consider Datalog grammars
(i.e. without function symbol f)

But not satisfactory!

Two origins to loops

1. Loops due to answers
2. Loops due to predictions

Loops due to answers

A program or grammar produces an infinite set of answers
due to a loop during answer propagation.

In generation mode, `append` produces infinitely many answers

```
append([],Y,Y).
append([A|X],Y,[A|Z]) :- append(X,Y,Z).
```

$\leadsto$

append([],Y,Y)
append([A],Y,[A|Y])
append([A,B],Y,[A,B|Y])
...

Rare in Parsing, possible in Generation.

Solution: No real solution, except using finitely ambiguous grammars.

Off-line parsable grammars

Off-line parsable grammars are finitely ambiguous:

> *[Shieber] There exists a projection ρ towards a finite domain generalizing parse trees. i.e. $\rho\tau \preceq \tau$, in such a way that no projected tree $\rho\tau$ is its own sub-tree for a given input string.*

In particular, if satisfied when projecting to the CF backbone, then the grammar is off-line parsable.

Off-line parsable grammars

Off-line parsable grammars are finitely ambiguous:

> *[Shieber] There exists a projection ρ towards a finite domain generalizing parse trees. i.e. $\rho\tau \preceq \tau$, in such a way that no projected tree $\rho\tau$ is its own sub-tree for a given input string.*

In particular, if satisfied when projecting to the CF backbone, then the grammar is off-line parsable.
But existence of off-line parsable grammars whose CF backbone is cyclic.

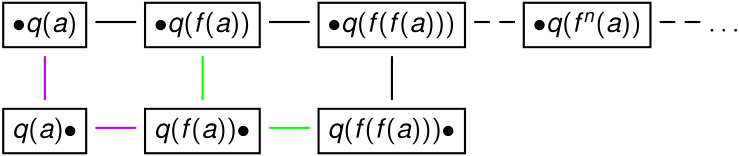
$$\begin{array}{ccccc} q(f(f(a)),u) & \rightsquigarrow & q(\_,u) & \rightsquigarrow & q(\_,\_) \\ | & & | & & | \\ q(f(a),v) & & q(\_,v) & & q(\_,\_) \\ | & & | & & | \\ q(a,w) & & q(\_,w) & & q(\_,\_) \end{array}$$

In Logic Programming $\equiv$ data driven stratification

Loops due to prediction

Non termination may arise from more and more precise predictions.

```
q(f(f(a))).                    ?-q(a).
q(X)  :- q(f(X)).
```



Prediction items may be generalized without altering neither correction or answer completeness

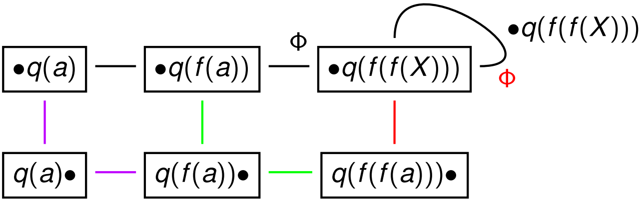
Use of **prediction restrictions** [Shieber]

$$\frac{\langle i, j, A \leftarrow \alpha \bullet B\beta \rangle}{\langle j, j, (C \leftarrow \bullet \gamma)\sigma \rangle} \qquad \exists C \leftarrow \gamma \text{ et } \sigma = mgu(\Phi B, C) \qquad \text{(PredR)}$$

with ΦB generalization of B ($\Phi B \preceq B$)

Idea: Transform spirals into loops that may be cut by subsumption.

$\Phi q(f^{n\geq 1}(t)) = q(f(\_)) \Longrightarrow$



Restrictions (cont'd)

In parsing, used to cut prediction spirals:

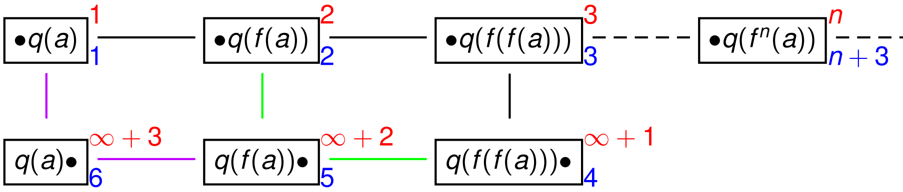
- on constituent lists
- on trace lists (*gap*)

They can improve computation sharing by removing pieces of information not needed to guide computations (ex: semantic forms)

But they may also induce useless computations (over-generalization)

Note: in Logic Programming: Term depth abstraction

- No tabular techniques can ensure a systematic termination



- However, tabulation allows suspension and resuming of computations $\Longrightarrow$ ensures computation completeness by scheduling in a fair way computation steps.

fairness No computation step can be forever ignored

Complexity

Complexities may be exponentional both in time and space

- Number of items (exponential in n) $\Longrightarrow$ table look-up
- Term size (exponential in n)
- Access to variable values (constant to linear wrt derivation lengths)
- Occurrence checking (exponential wrt term size)

Furthermore, 2 costly operations: unification & subsumption.

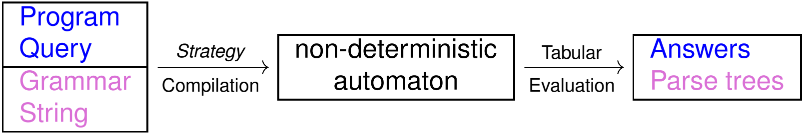
Note: Polynomial complexity for Datalog programs and grammars

Outline

Push-Down Automata / Dynamic Programming

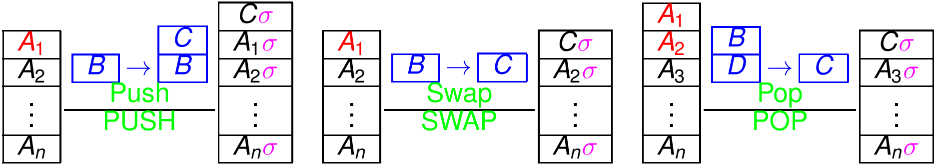
Approach [Lang, De la Clergerie] relying on:

1. automata to describe the steps of a parsing strategy
   ⟹ use of Push-Down Automata [PDA] working on "information-rich" stacks.
   Note: PDAs well-known for CFGs (equivalence)
2. Dynamic Programming principles to design tabular evaluations for these automata



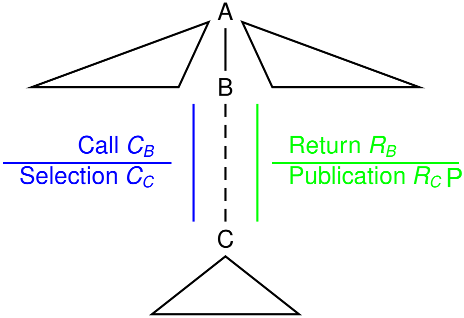
PDA extension::

- Stacks of 1st order logical terms
- 3 transition kinds (PUSH, SWAP & POP).
- Unification used to apply transitions

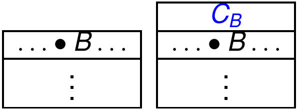


$\sigma = \mathrm{mgu}(A_1, B)$ $\sigma = \mathrm{mgu}(A_1, B)$ $\sigma = \mathrm{mgu}(A_1 A_2, BD)$

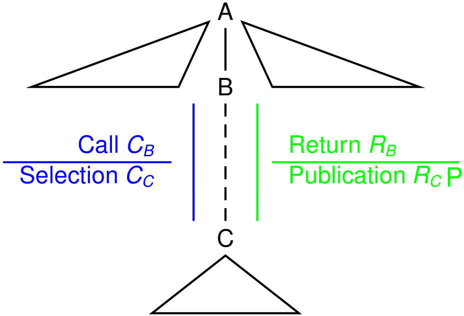
Parsing steps



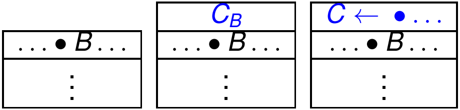
Call	of a non-terminal to recognize
Selection	of a production
Publication	of a recognized non-terminal
Return	to the calling production



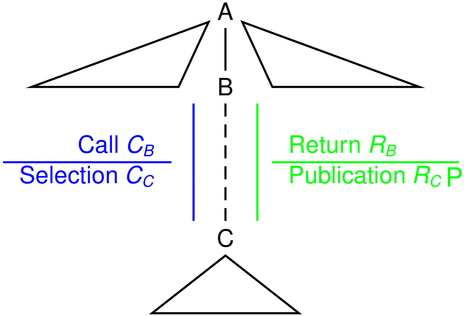
Parsing steps



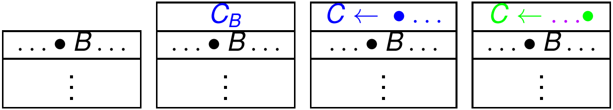
Call	of a non-terminal to recognize
Selection	of a production
Publication	of a recognized non-terminal
Return	to the calling production



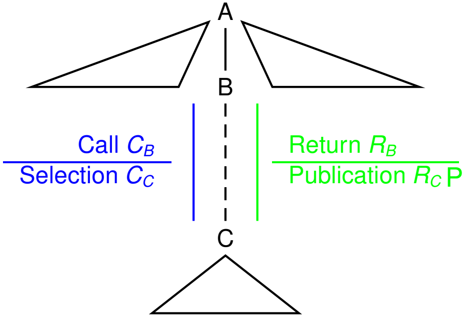
Parsing steps



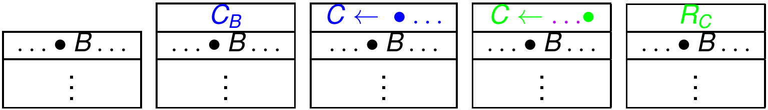
Call	of a non-terminal to recognize
Selection	of a production
Publication	of a recognized non-terminal
Return	to the calling production



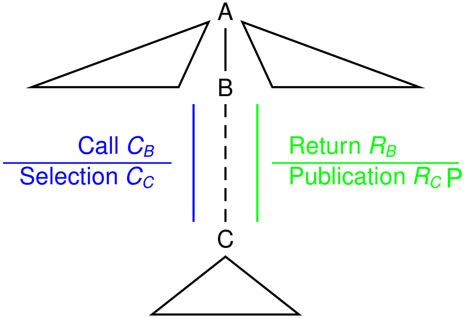
Parsing steps



- **Call** of a non-terminal to recognize
- **Selection** of a production
- **Publication** of a recognized non-terminal
- **Return** to the calling production



Parsing steps

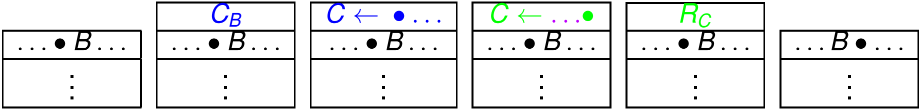


Call of a non-terminal to recognize

Selection of a production

Publication of a recognized non-terminal

Return to the calling production



Modulated Call/Return strategies

Approximation of each non-terminal A by $\begin{cases} C_A \text{ for Call \& Selection steps} \\ R_A \text{ for Return \& Publication steps} \end{cases}$

Strategy	C_A	R_A
Top-Down	A	$\bot$
Bottom-Up	$\bot$	A
Earley	A	A'

[Select] $\boxed{C_A} \longrightarrow \boxed{A \leftarrow \bullet \ldots}$

[Publish] $\boxed{A \leftarrow \ldots \bullet} \longrightarrow \boxed{R_A}$

[Call] $\boxed{A \leftarrow \ldots \bullet B \ldots} \longrightarrow \boxed{C_B} \; \boxed{A \leftarrow \ldots \bullet B \ldots}$

[Return] $\boxed{R_B} \; \boxed{A \leftarrow \ldots \bullet B \ldots} \longrightarrow \boxed{A \leftarrow \ldots B \bullet \ldots}$

Modulation validity:

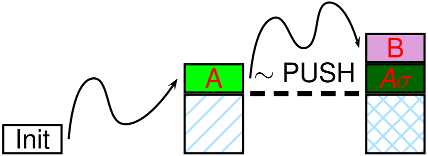
$$\text{“Information”}(A) = \text{“Information”}(C_A) + \text{“Information”}(R_A)$$

Dynamic Programming

Dynamic Programming Recursive decomposition of a problem into simpler sub-problems that may be re-used (ex. knapsack problem).

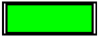
For (L)PDAs, we try to

1. Identify elementary sub-derivations
2. Identify pertinent information in these derivations to build traces (items) as compact as possible.
   (motivation: save space and improve computation sharing)
3. Combine these items to get a tabular evaluation sound and complete w.r.t. the standard derivations.



 not consulted nor modified (but instantiated) ⟹ Sharing

A ~ PUSH derivation representable:

[forgetting about instantiation] by pair ( , )

[taking into account instantiation] same pair + instantiation measure

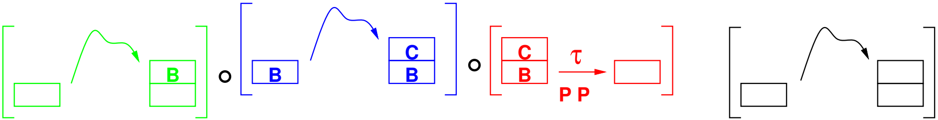
transition properties ⟹ ( , ) or even 

Conclusion: An item is a PUSH derivation representable by a stack fragment.

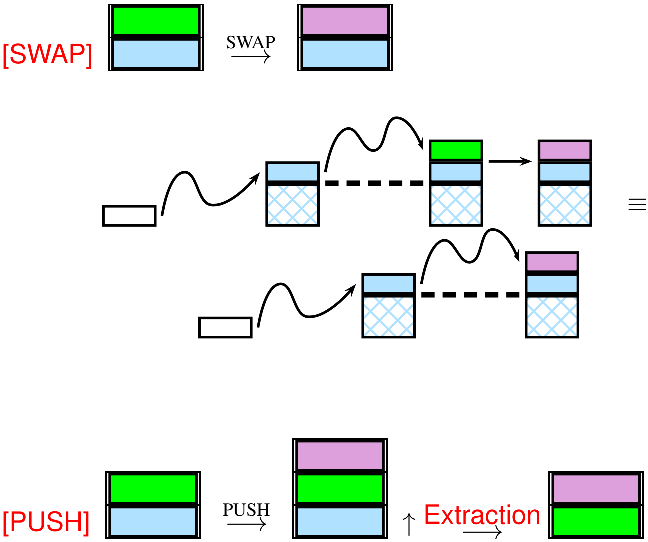
Item composition for LPDAs

PDA derivations may be retrieved by composition of items and transitions.

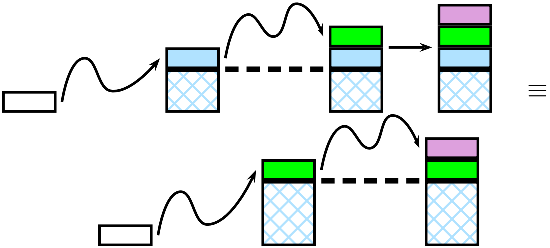
Composition of a POP transition with two items: $(A, B) \circ (B, C) \circ \tau = (A, D)$



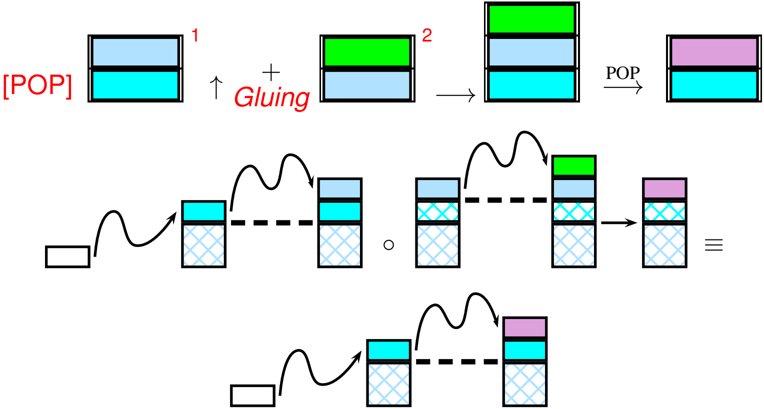
Item composition (without instantiation) I



Item composition (without instantiation) II

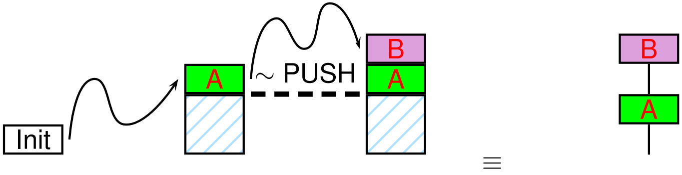


Item composition (without instantiation) III

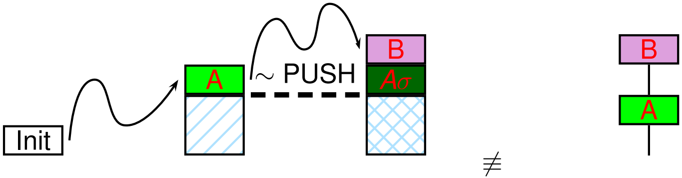


Relationships with Graph-structured stacks

No instantiation (CFG case) 2-items & Graph-structured stacks are similar



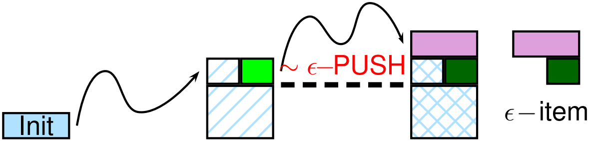
With instantiation Not equivalent because of σ



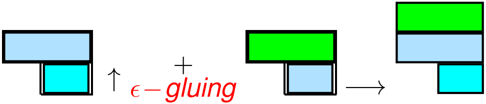
Graph-structured stacks also factorize on B
(interesting when no instantiation)

Items / 2nd form [$\mathcal{S}1+\epsilon$]

Instead of PUSH transition, we consider ϵ–PUSH that only examine a fraction ϵ of information on stack tops.



Combination: Similar to $\mathcal{S}2$ but more complex combining



$\mathcal{S}1+\epsilon$ is sound and complete for PDDs using ϵ–PUSH transitions.

Approximation of each non-terminal A by $\begin{cases} C_A \text{ for each Call \& Select} \\ R_A \text{ for each Return \& Publish} \end{cases}$

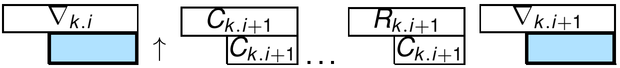
[S]elect $C_{l.0} \longrightarrow \nabla_{l.0}$

[P]ublish $\nabla_{l.n_l} \longrightarrow R_{l.0}$

[C]all $\nabla_{k.i} \longrightarrow \begin{array}{c} C_{k.i+1} \\ \nabla_{k.i} \end{array}$

[R]eturn $\begin{array}{c} R_{k.i+1} \\ \nabla_{k.i} \end{array} \longrightarrow \nabla_{k.i+1}$

PUSH Call transitions equivalent to $\epsilon-$PUSH with $\epsilon(\nabla_{k.i}) = C_{k.i+1}$.



1-component item [$\mathcal{S}1$]

For bottom-up strategies (with or without prediction), i.e. $R_A \equiv A$, the stack topmost element holds a lot of information.

$$\begin{array}{|c|}\hline A \\ \hline \end{array}\ \boxed{C_A} \quad \text{induces } \text{“info”}(C_A) \subset \text{“info”}(A)$$

$\Longrightarrow$ Possible to take the topmost stack elements as items

$\mathcal{S}1$ interpretation similar to deductive systems

$$\boxed{\text{A}} + \boxed{\text{B}} + \text{POP}\,\{(A,B) \to C\} \qquad \text{equivalent} \qquad \frac{A\ B}{C}$$

$1 + \epsilon$-items as efficient as 1-items for a wider spectrum of strategies.