Logical and Computational Structures for Linguistic Modeling
Part 3 – Mildly Context-Sensitive Formalisms

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

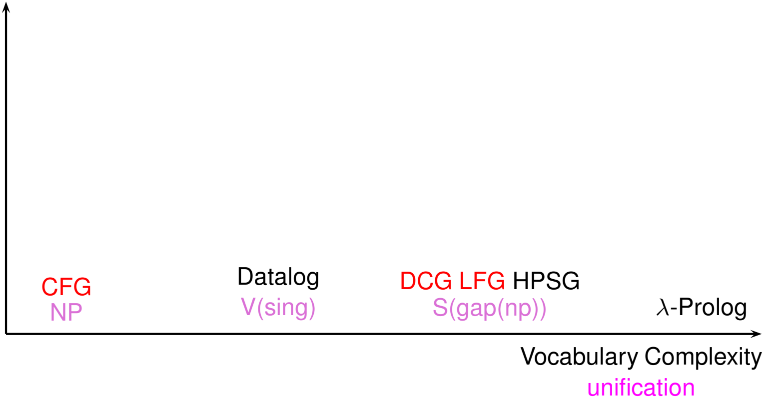
30 Septembre 2014

Part I

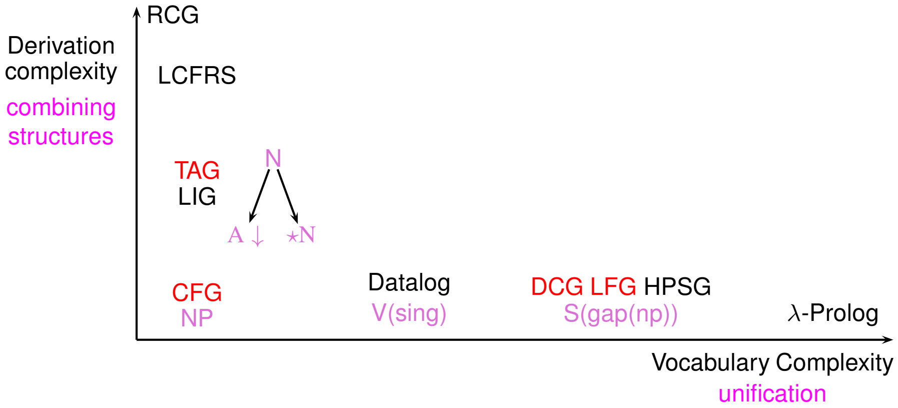
Tree Adjoining Grammars

Extending CFG with structures

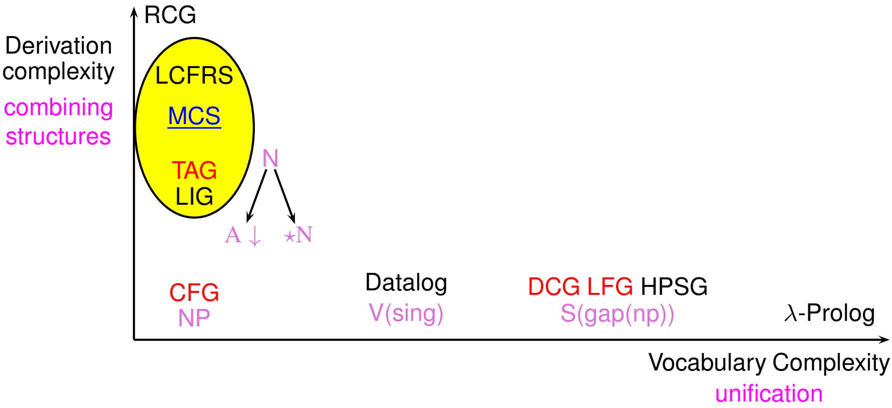
Extending CFG with structures



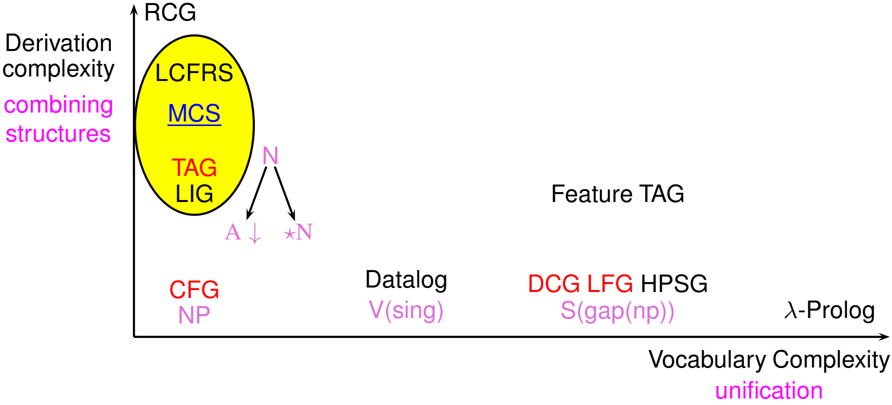
Extending CFG with structures



Extending CFG with structures



Extending CFG with structures



Outline

1. Some background about TAGs
2. Deductive chart-based TAG parsing
3. Automata-based tabular TAG parsing

From CFG to Tree Substitution Grammars

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

CFG productions:

- are too local
  $\Longrightarrow$ need decorations for info propagation
- are generally not lexicalized
  but info often propagated from words
  also more efficient parsing algo for lexicalized grammars

From CFG to Tree Substitution Grammars

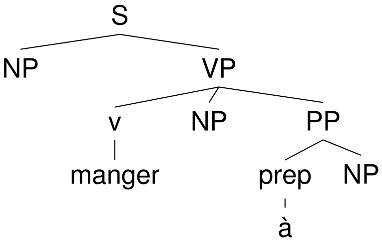
```
s   --> np vp
np  --> pn
np  --> det n
np  --> np pp
vp  --> v np
vp  --> vp pp
pp  --> prep
  np
```

CFG productions:

- are too local
  $\Longrightarrow$ need decorations for info propagation
- are generally not lexicalized
  but info often propagated from words
  also more efficient parsing algo for lexicalized grammars

CFG productions can be grouped into trees
$\Longrightarrow$ we get Tree Substitution Grammars (TSG)

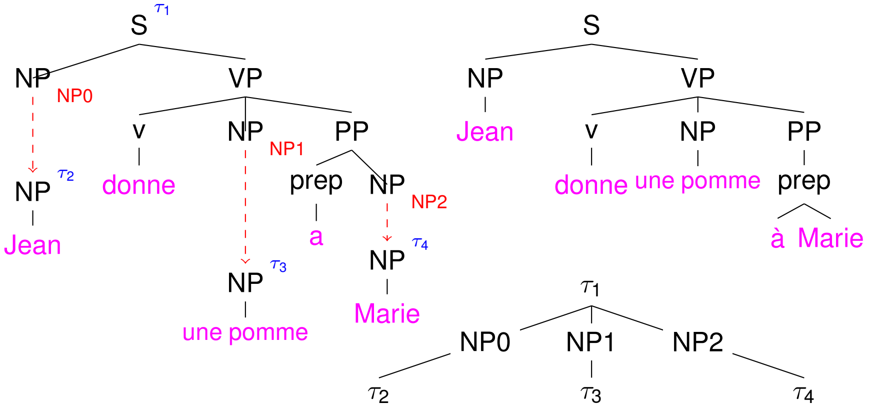
For instance, dealing with ditransitive verb **donner**



Derivation Tree vs Parse Tree

TSG are strongly equivalent to CFG

However, for TSG, parse trees and derivation trees are not equivalent



Furthermore, several derivations may lead to a same parse tree

One step farther: adjoining

How to deal with:

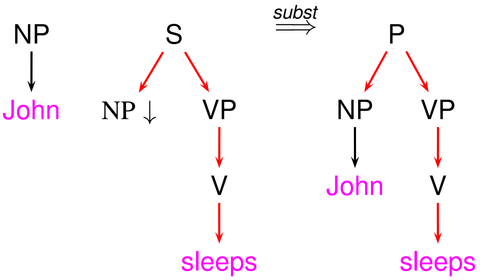
Jean donne souvent une pomme à Marie

Need a way to insert the adverb somewhere in the verbal tree
$\Longrightarrow$ adjoining operation

$\leadsto$ Tree Adjoining Grammars

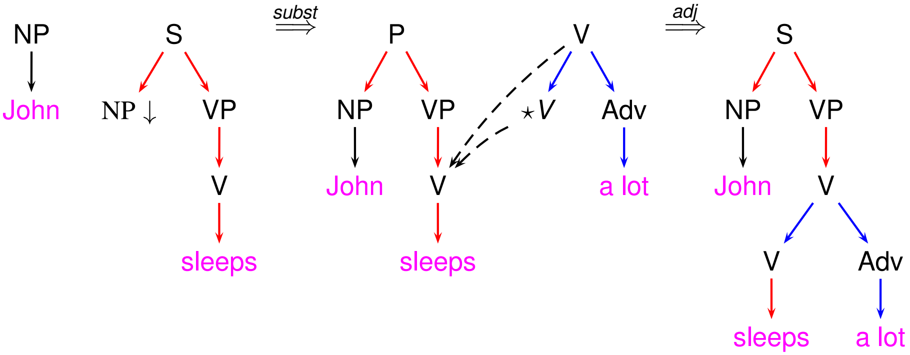
TAG: a small example

Tree Adjoining Grammars [TAGs] [Joshi] build parse trees from initial and auxiliary trees by using 2 tree operations: substitution and adjoining

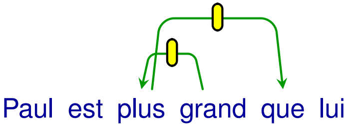


TAG: a small example

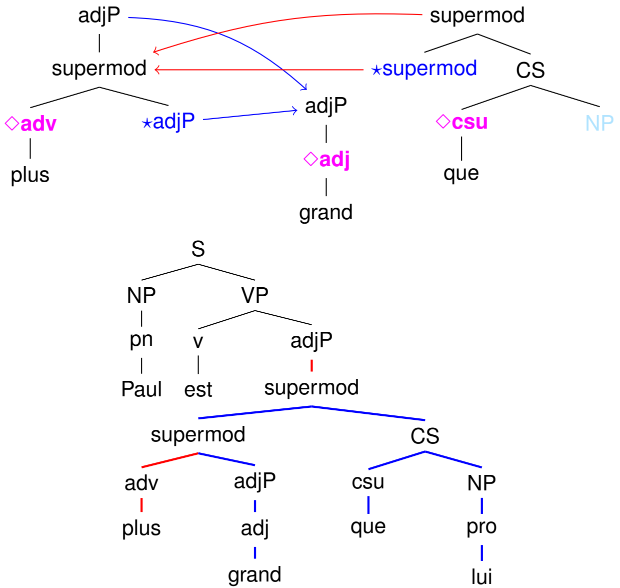
Tree Adjoining Grammars [TAGs] [Joshi] build parse trees from initial and auxiliary trees by using 2 tree operations: substitution and adjoining



A more complex example: French comparative



A more complex example: French comparative



TAGs: (First) Formal definition

A TAG G is a tuple $(\mathcal{N}, \Sigma, S, \mathcal{I}, \mathcal{A})$ where

- Σ a finite set of terminal symbols
- $\mathcal{N}$ a finite set of non-terminal symbols
- $S \in \mathcal{N}$ the axiom
- $\mathcal{I}$ and $\mathcal{A}$ are two finite sets of elementary trees over $\mathcal{N} \cup \Sigma \cup \{\epsilon\}$
  only leaf nodes ν may have a label $l(\nu) \in \Sigma \cup \{\epsilon\}$
  - the trees α in $\mathcal{I}$ are initial trees
  - the trees β in $\mathcal{A}$ are auxiliary trees and have a unique leaf node marked $(\star)$ as a foot f_β with same label than the root node r_β, i.e. $l(f_\beta) = l(r_\beta)$

Two operations may be used to combine the elementary trees

- substitution of a leaf node ν of γ by some initial tree $\alpha \in \mathcal{I}$, $(l(\nu) = l(t_\alpha))$
- adjoining of an (internal) node ν of γ by some auxiliary tree $\beta \in \mathcal{A}$

G generates a tree language and a string language

$$T(G) = \{\gamma | \alpha \Longrightarrow^\star \gamma \wedge \mathrm{yield}(\gamma) \in \Sigma^\star \wedge \alpha \in \mathcal{I} \wedge r_\alpha = S\}$$

$$L(G) = \{\mathrm{yield}(\gamma) | \gamma \in T(G)\}$$

Adjoining

Assuming $\gamma = (V, E)$ with $\nu \in V$ and $\beta = (V_\beta, E_\beta)$ with $r_\beta, f_\beta \in V_\beta$, such that $l(\nu) = l(r_\beta) \in \mathcal{N}$

$$\gamma[adj(\nu, \beta)] = (V', E')$$

with

$$\begin{cases} V' = V \cup V_\beta \backslash \{\nu\} \\ E' = \bigcup \begin{cases} \{(x,y) \in E | x \neq \nu \wedge y \neq \nu\} \\ E_\beta \\ \{(x, r_\beta) | (x,\nu) \in E\} \\ \{(f_\beta, y) | (\nu, y) \in E\} \end{cases} \end{cases}$$

Note: The node sets are assumed to be renamed to avoid clashes, i.e. $E \cap E' = \emptyset$

Adjoining contraints

A full definition of TAGs should include constraints on adjoining nodes:

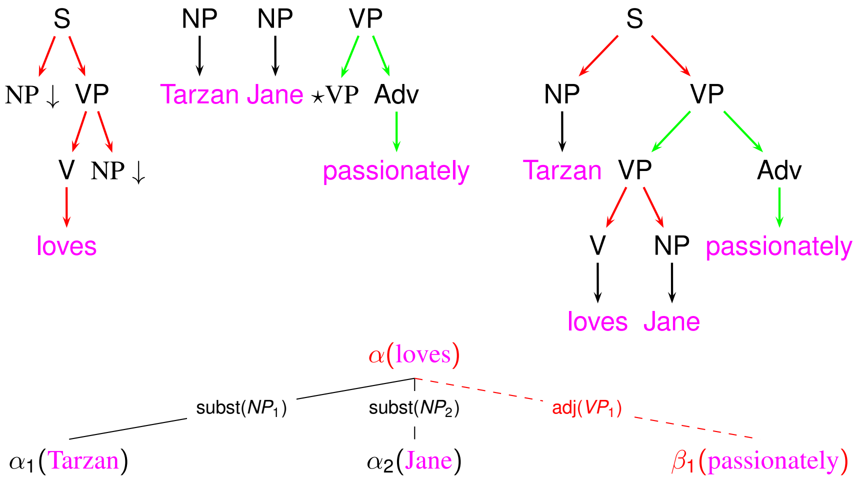
A TAG G is a tuple $(\mathcal{N}, \Sigma, S, \mathcal{I}, \mathcal{A}, f_{OA}, f_{SA})$ where,
assuming V set of nodes in $\mathcal{I} \cup \mathcal{A}$,

- $f_{OA} : V \mapsto \{0, 1\}$ specify if adjoining on ν is obligatory (1) or not (0)
- $f_{SA} : V \mapsto 2^{\mathcal{A}}$ specify which auxiliary trees may be adjoined on ν
  note: ν becomes non-adjoinable with $f_{SA}(\nu) = \emptyset$

Adjoining constraints necessary for getting the full expressive power of TAGs but they are often implicit:

- no adjoining on leaf nodes (including foot nodes)
- explicit mandatory adjoining (MA, $+$) marks on some nodes
- explicit non adjoining (NA, $-$) marks on some nodes

Derivation tree



For TAGs, derivation tree not isomorphic to parse tree
but close from semantic level.

Regular Tree Languages

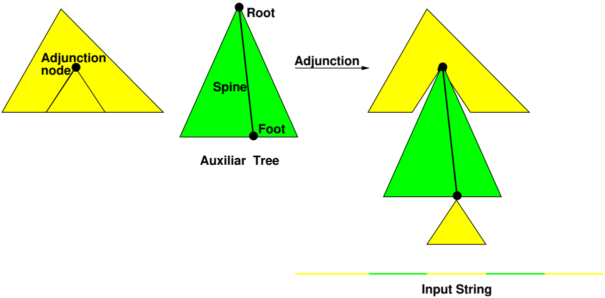
For a TAG G, its set of derivation trees $D(G)$ forms a regular tree language

i.e., $D(G)$ may be generated by a finite tree automaton
(top-down) term rewrite rules of the form

$$q_0 \leftarrow a(q_1, \ldots, q_n), \; q_i \in \mathcal{Q}, a \in \mathcal{F}$$

may also be seen as the parse trees for some CFG G'

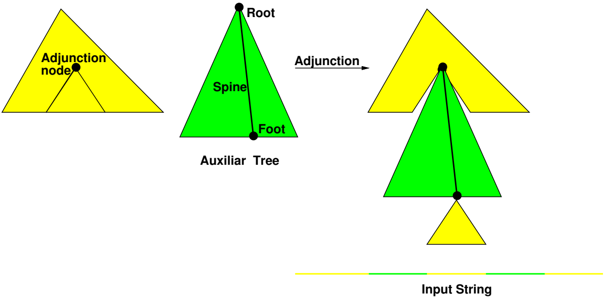
TAG complexity: Adjoining



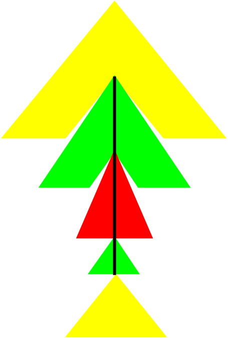
adjoining

- discontinuity (hole in aux tree)
- crossing (both sides of the hole)

TAG complexity: Adjoining



adjoining



nested adjoining

- discontinuity (hole in aux tree)
- crossing (both sides of the hole)
- unbounded synchronization (both sides of spine)

Expressive power of TAGs

The adjoining operation extends the expressive power of TAGs w.r.t. CFGs.

- long distance dependencies (wh-pronoun extraction for instance)
- crossed dependencies as given by copy language "ww" or by language "$a^n b^n c^n$"

(1) omdat ik Cecillia de nijlpaarden zag voeren
because I Cecilia the hippopotamuses saw feed
because I saw Cecilia feed the hippopotamuses

Expressive power (limits)

TAGs can't handle the following languages:

- $a^n b^m c^n d^m e^n f^m$
- multiple copy languages w^n with $n > 2$.

Pumping lemma

Tree Adjoining Languages satisfy a pumping lemma

If L is a TAL, then there exists N, such for all $w \in L$ and $|w| > N$, there exist $x, y, z, v_1, v_2, w_1, w_2, w_3, w_4 \in \Sigma^\star$, such that

$$\begin{cases} |v_1 v_2 w_1 w_2 w_3 w_4| \leq N \\ |w_1 w_2 w_3 w_4| \geq 1 \end{cases}$$

and one of the following case holds

1. $w = xw_1 v_1 w_2 y w_3 v_2 w_4 z$ and $\forall k \geq 0$, $xw_1^k v_1 w_2^k y w_3^k v_2 w_4^k z \in L$
2. $w = xw_1 v_1 w_2 v_2 w_3 y w_4 z$ and $\forall k \geq 0$, $xw_1^{k+1} v_1 w_2 v_2 w_3 (w_2 w_4 w_3)^k y w_4 z \in L$
3. $w = xw_1 y w_2 v_1 w_3 v_2 w_4 z$ and $forallk \geq 0$, $xw_1 y (w_2 w_1 w_3)^k w_2 v_1 w_3 v_2 w_4^{k+1} \in L$

Closure properties of TALs

As CFLs, TALs form an Abstract Family of Languages (AFL):

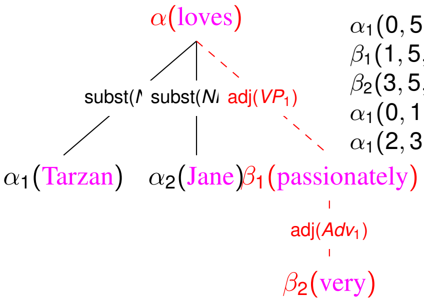
1. closed by intersection with regular languages
2. closed by union, concatenation, and Kleene-iteration
3. closed by homomorphism and inverse homomorphism

In particular, (1) $\Longrightarrow$ notion of Shared Derivation Forest

Shared Derivation Forests

Formal definition in Vijay-Shanker & Weir 1993

$_0$ *Tarzan* $_1$ *loves* $_2$ *Jane* $_3$ *very* $_4$ *passionately* $_5$



$\alpha_1(0,5) \rightarrow \alpha_1(0,1)$ loves$(1,2)$ $\alpha_2(2,3)$ $\beta_1(1,5,1,3)$
$\beta_1(1,5,1,3) \rightarrow \beta_2(3,5,4,5)$ passionately$(4,5)$
$\beta_2(3,5,4,5) \leftarrow$ very$(3,4)$
$\alpha_1(0,1) \leftarrow$ Tarzan$(0,1)$
$\alpha_1(2,3) \leftarrow$ Janes$(0,1)$

More formally, use tree nodes rather than trees
Space complexity in $O(n^6)$ by binarization (adj on spine node ν)

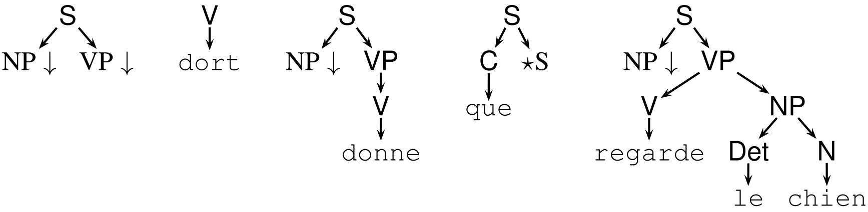
$$\nu^\top(i,j,r,s) \rightarrow r_\beta^\top(i,j,p,q)\ \nu^\perp(p,q,r,s)$$

Well formed trees

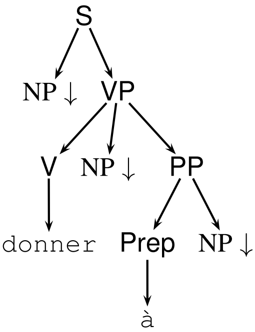
Many possible ways to define elementary trees
In practive, elementary trees follow some linguistic principles:

- lexical anchoring: at least, one non-empty lexical (frontier) node the head (or anchor)
- sub-categorization: a frontier node for each argument sub-categorized by the head
  domain of locality
- semantic consistency: a tree correspond to the scope of a semantic predicate with its arguments
- non-composition: a tree stands for a single semantic unit

A few bad trees:

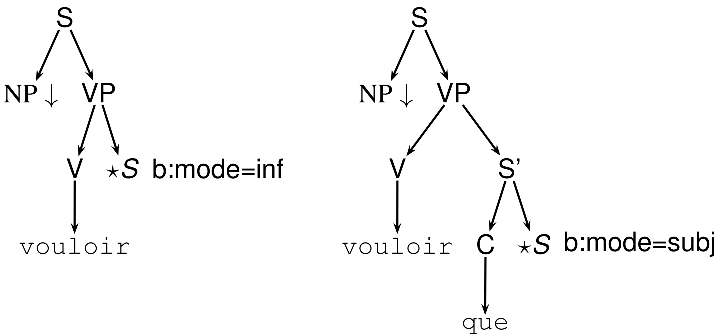


Subcategorization



Feature TAGs

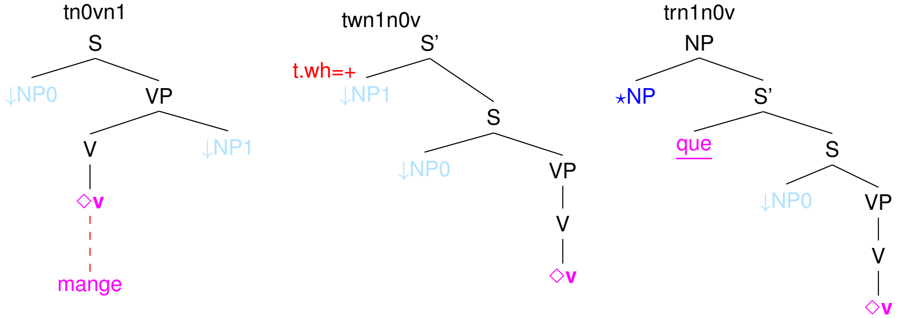
The nodes may be decorated with a pair (top, bot) of decorations



When adj on ν, unification of $\nu.top$ with $r_\beta.top$ and $\nu.bot$ with $f_\beta.bot$
alternate way to express adjoining constraints

Note: for flat decorations, same expressive power and complexity

Trees derived from a canonical ones grouped into families
e.g. family of transitive verbs



+ all other extractions (on NP_0) + passive + extractions on passive
+ ordering + multiple realizations + . . .

⇝ XTAG architecture

- a set of trees (with anchor nodes) grouped into families
- a lexicon $\mathcal{L}$ specifying for each word *w* the set of families it may anchor + additional constraints

Meta-grammars

Large coverage TAG $\Longrightarrow$ many trees to write and maintain !

Alternative: generate the trees from a higher description level: meta-grammars Abeillé, Candito

- hierarchy of classes, containing constraints
  A precedes B, A dominates B, ...
- a class deals with a linguistic facet
  e.g. verb argument, refined into subject or object
- a class may require or provide functionalities
- the classes may be combined to form neutral classes
- the constraints of the neutral classes used to generate elementary trees

$\Longrightarrow$ used for FRMG, a large-coverage French TAG
`http://alpage.inria.fr/frmgwiki`
(plus mechanisms for factorizing elementary trees)

Long-distance dependencies

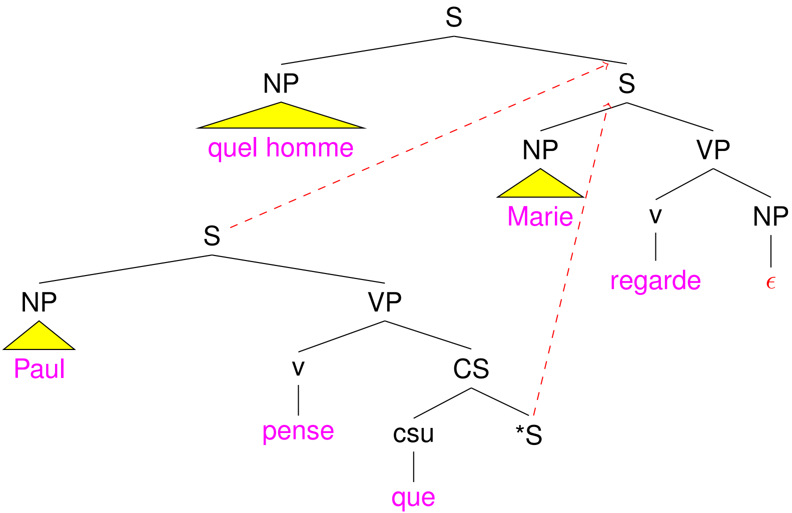
(Recursive) Adjoining may replace LFG's functional uncertainty for long-distance dependencies

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

$$
\begin{array}{lll}
\text{S'} \longrightarrow & \text{NP} & \text{S} \\
 & (\downarrow\text{Wh}) =_c + & \uparrow=\downarrow \\
 & (\uparrow\text{Focus}) = \uparrow & (\downarrow\text{Wh})=+ \\
 & (\uparrow\text{Focus})=\uparrow(\mathrm{Comp})^{\star}\mathrm{Obj} &
\end{array}
$$

Long-distance dependencies (TAGs)

Handled through repeated adjoining



Outline

Deductive parsing

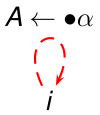
Formalization of chart parsing

Use of

- universe of tabulable **items**, representing (set of) partial parses
- items often build upon **dotted rules** $A_0 \leftarrow A_1 \dots A_i \bullet A_{i+1} \dots A_n$
- chart edges labeled by dotted rules (items $\equiv \langle i, j, A \leftarrow \alpha \bullet \beta \rangle$)
- a deductive system specifying how to derive items

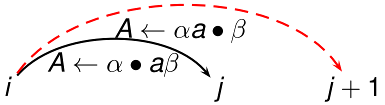
CKY as a deductive system (for CFGs)

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



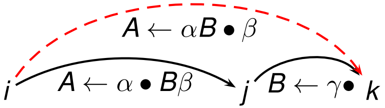
(Seed)

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



(Scan)

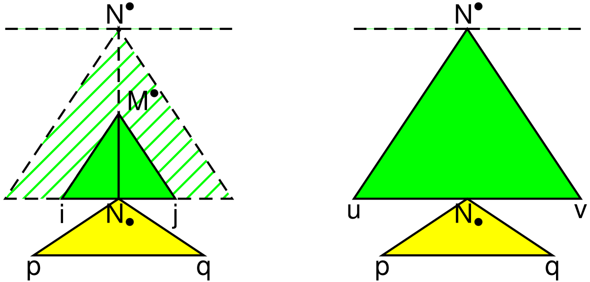
$$\frac{\langle i, j, A \leftarrow \alpha \bullet B\beta \rangle \quad \langle j, k, B \leftarrow \gamma\bullet \rangle}{\langle i, k, A \leftarrow \alpha B \bullet \beta \rangle}$$



(Complete)

CKY algorithm for TAGs [Vijay-Shanker & Joshi 85]
Presentation:

- Dotted trees $N^\bullet$ and $N_\bullet$ where N is a node of an elementary tree
- Items $\langle N^\bullet, i, p, q, j\rangle$ and $\langle N_\bullet, i, p, q, j\rangle$ with p, q possibly covering a foot node.



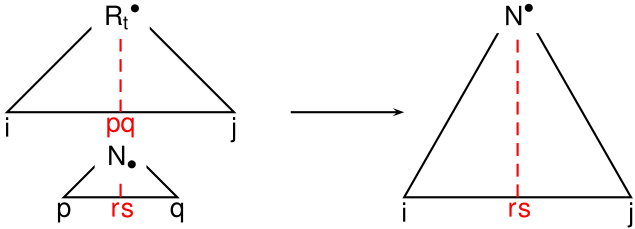
$\langle M_\bullet, i, p, q, j\rangle$

Without adjoining: $\langle N_\bullet, p, -, -, q\rangle$
With adjoining: $\langle N^\bullet, u, -, -, v\rangle$

Rule (Adjoin)

Gluing a sub-tree at a foot node.

$$\frac{\langle \mathrm{N}_{\bullet}, p, r, s, q\rangle \;\; \langle \mathrm{R_t}^{\bullet}, i, p, q, j\rangle}{\langle \mathrm{N}^{\bullet}, i, r, s, j\rangle} \qquad \mathrm{label}(N) = \mathrm{label}(R_t) \qquad \text{(Adjoin)}$$



Rule (NoAdjoin)

When no adjoining on a node

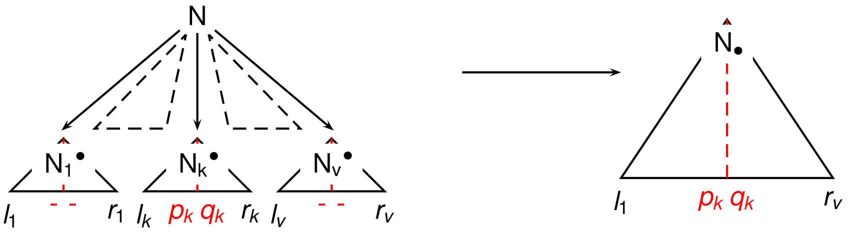
$$\frac{\langle \mathrm{N}_{\bullet}, p, r, s, q \rangle}{\langle \mathrm{N}^{\bullet}, p, r, s, q \rangle} \qquad \text{(NoAdjoin)}$$

Rule (Complete)

Gluing all node's children

$$\frac{\langle N_i^{\bullet}, l_i, p_i, q_i, r_i \rangle_{i=1,\ldots,v}}{\langle N_{\bullet}, l_1, \cup p_i, \cup q_i, r_v \rangle} \quad \begin{array}{c} N \\ \swarrow \quad \downarrow \quad \searrow \\ N_1 \quad \ldots \quad N_v \end{array} \quad \text{and } \forall i,\ l_{i+1} = r_i \qquad \text{(Complete)}$$

Note: At most one child (k) covers a foot node with $(\cup p_i, \cup q_i) = (p_k, q_k)$



Complexity

Other deductive rules needed to handle

1. substitution
2. terminal scanning

+ axioms

Time complexity $O(n^{\max(6,1+v+2)})$ with

- v : maximal number of children per node
- 2 : number of indexes to cover a possible unique foot node

Normalization using binary-branching trees ($v = 2$) $\Longrightarrow$ complexity $O(n^6)$

4 indexes per item $\Longrightarrow$ Space complexity in $O(n^4)$ for a recognizer
$O(n^6)$ for a parser, keeping backpointers to parents

Optimal worst-case complexities
but practically, even less efficient than CKY for CFGs

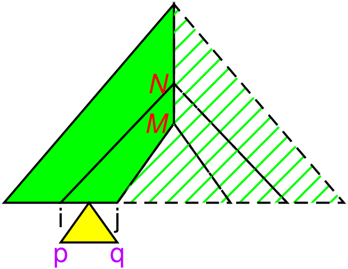
Prediction, dotted trees and dotted producctions

To mark prediction, new dotted trees [Shabes]: $^{\bullet}$N and $_{\bullet}$N

Alternative: equivalence with dotted productions

$$\begin{tikzcd} & N \arrow[dl] \arrow[d] \arrow[dr] & \\ N_1 & \ldots & N_v \end{tikzcd} \equiv N \leftarrow N_1 \ldots N_v$$

dotted tree	dotted production
$N_k^{\bullet}$, $^{\bullet}N_{k+1}$	$N \leftarrow N_1 \ldots N_k \bullet N_{k+1} \ldots N_v$
$^{\bullet}$R (root)	$\top \leftarrow \bullet R$
R$^{\bullet}$ (root)	$\top \leftarrow R \bullet$
$_{\bullet}$N	$N \leftarrow \bullet N_1 \ldots N_v$
N$_{\bullet}$	$N \leftarrow N_1 \ldots N_n \bullet$



$$\langle N \leftarrow \alpha \bullet M\beta, i, p, q, j \rangle$$

Non prefix valid Earley algorithm

- Glue a sub-tree at foot node F_t (maybe useless !)

$$\frac{\langle M \leftarrow \gamma\bullet, p, r, s, q\rangle \;\; \langle \top \leftarrow R_t\bullet, i, p, q, j\rangle}{\langle M \leftarrow \gamma\bullet, i, r, s, j\rangle} \quad \mathrm{label}(M) = \mathrm{label}(R_t) \qquad \text{(Adjoin)}$$

- Advance in recognition of N's children

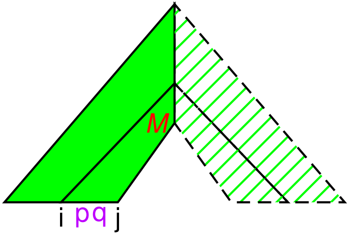
$$\frac{\langle N \leftarrow \alpha \bullet M\beta, i, u, v, j\rangle \;\; \langle M \leftarrow \gamma\bullet, j, r, s, k\rangle}{\langle N \leftarrow \alpha M \bullet \beta, i, u \cup r, v \cup s, k\rangle} \qquad \text{(Complete)}$$

(Adjoin) and (Complete) similar to CKY (binary form)

Adjoining Prediction

Predict adjoining at M

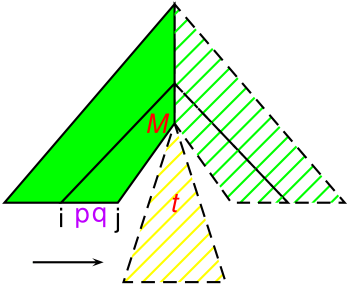
$$\frac{\langle N \leftarrow \alpha \bullet M\beta, i, p, q, j\rangle}{} \qquad \text{label}(M) = \text{label}(R_t) \qquad \text{(CallAdj)}$$



Adjoining Prediction

Predict adjoining at M

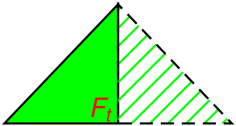
$$\frac{\langle N \leftarrow \alpha \bullet M\beta, i, p, q, j\rangle}{\langle \top \leftarrow \bullet R_t, j, -, -, j\rangle} \qquad \text{label}(M) = \text{label}(R_t) \qquad \text{(CallAdj)}$$



Foot Prediction

Predict a sub-tree root at M to recognize below foot node F_t

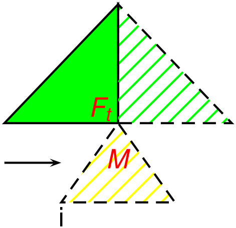
$$\frac{\langle F_t \leftarrow \bullet \perp, i, -, -, i\rangle}{} \qquad \mathrm{label}(F_t) = \mathrm{label}(M) \qquad \text{(CallFoot)}$$



Foot Prediction

Predict a sub-tree root at M to recognize below foot node F_t

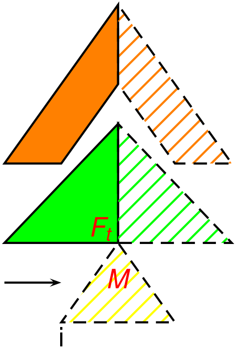
$$\frac{\langle F_t \leftarrow \bullet \perp, i, -, -, i\rangle}{\langle M \leftarrow \bullet\gamma, i, -, -, i\rangle} \qquad \text{label}(F_t) = \text{label}(M) \qquad \text{(CallFoot)}$$



Foot Prediction

Predict a sub-tree root at M to recognize below foot node F_t

$$\frac{\langle F_t \leftarrow \bullet \perp, i, -, -, i\rangle}{\langle M \leftarrow \bullet\gamma, i, -, -, i\rangle} \quad \text{label}(F_t) = \text{label}(M) \qquad \text{(CallFoot)}$$



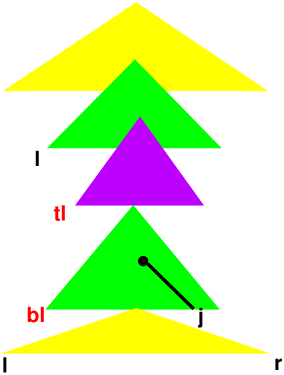
The prediction of M not related to the node M' having triggered the adjoining of t
$\Longrightarrow$ Non prefix valid parsing strategy

Complexity

- Space complexity remains $O(n^4)$
- Dotted productions $\Longrightarrow$ implicit binarization $\Longrightarrow$ time in $O(n^6)$
- Non prefix valid: impact difficult to evaluate in practice
- **Note**: Dotted productions also applicable to improve CKY

Prefix valid Early [Shabes]

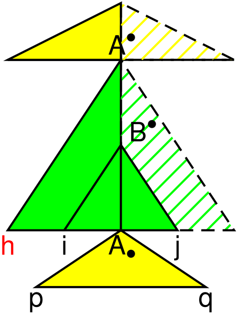
Complexities time in $O(n^9)$ and space in $O(n^6)$ due to 6-index items



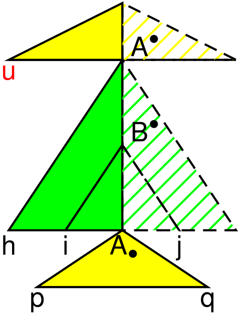
Actually, *tl* and *bl* may be avoided using dotted productions

Prefix valid Earley [Nederhof]

Item with only an extra index h: $\langle h, N \leftarrow \alpha \bullet \beta, i, p, q, j\rangle$
h states starting (leftmost) position of current tree



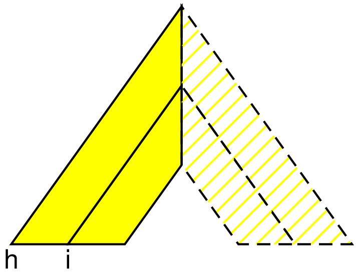
$\langle h, N \leftarrow \alpha B \bullet \beta, i, p, q, j\rangle$



$\langle u, A \leftarrow \gamma \bullet, p, -, -, q\rangle$

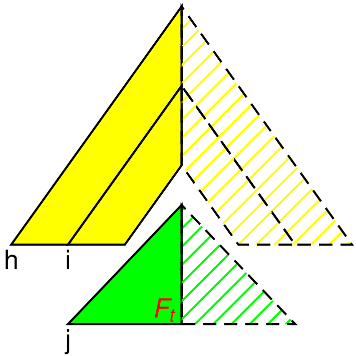
Foot prediction

$$\frac{\langle h, N \leftarrow \alpha \bullet M\beta, i, p, q, j \rangle}{} \quad \text{label}(F_t) = \text{label}(M) \qquad \text{(CallFootPf)}$$



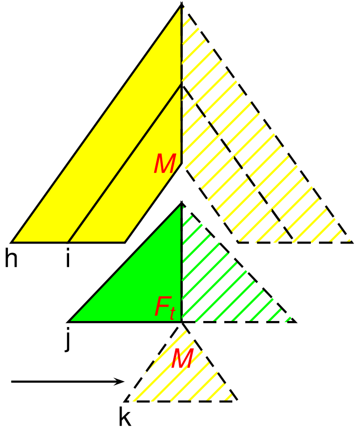
Foot prediction

$$\frac{\begin{array}{c}\langle h, N \leftarrow \alpha \bullet M\beta, i, {\color{purple}p}, {\color{purple}q}, j\rangle \\ \langle j, F_t \leftarrow \bullet \perp, k, {\color{purple}-}, {\color{purple}-}, k\rangle\end{array}}{} \quad \mathrm{label}(F_t) = \mathrm{label}(M) \qquad \text{(CallFootPf)}$$



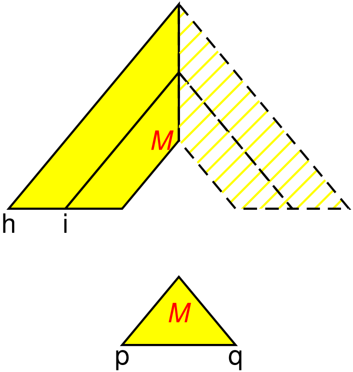
Foot prediction

$$\frac{\begin{array}{c}\langle h, N \leftarrow \alpha \bullet M\beta, i, p, q, j\rangle \\ \langle j, F_t \leftarrow \bullet \perp, k, -, -, k\rangle\end{array}}{\langle h, M \leftarrow \bullet\gamma, k, -, -, k\rangle} \qquad \text{label}(F_t) = \text{label}(M) \qquad \text{(CallFootPf)}$$



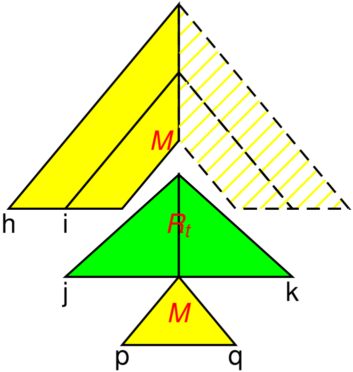
Adjoining return

$$\frac{\begin{array}{c}\langle h, N \leftarrow \alpha \bullet M\beta, i, u, v, j\rangle \\ \langle h, M \leftarrow \gamma\bullet, p, r, s, q\rangle\end{array}}{} \qquad \text{label}(M) = \text{label}(R_t) \qquad \text{(AdjoinPf)}$$



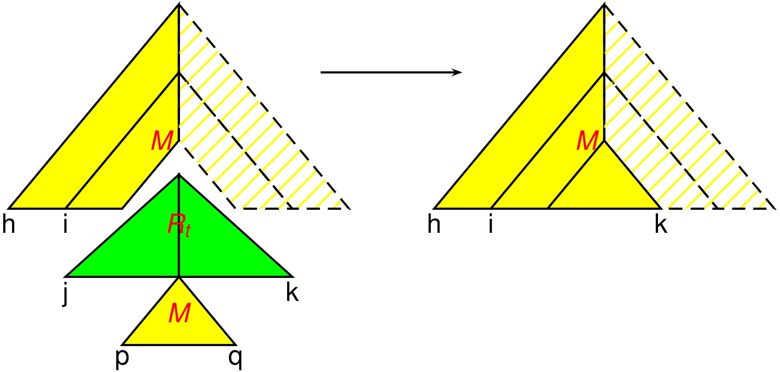
Adjoining return

$$\frac{\begin{array}{c}\langle h, N \leftarrow \alpha \bullet M\beta, i, u, v, j\rangle \\ \langle j, \top \leftarrow R_t \bullet, j, p, q, k\rangle \\ \langle h, M \leftarrow \gamma \bullet, p, r, s, q\rangle\end{array}}{} \quad \mathrm{label}(M) = \mathrm{label}(R_t) \qquad \text{(AdjoinPf)}$$



Adjoining return

$$\frac{\begin{array}{c}\langle h, N \leftarrow \alpha \bullet M\beta, i, u, v, j\rangle \\ \langle j, \top \leftarrow R_t\bullet, j, p, q, k\rangle \\ \langle h, M \leftarrow \gamma\bullet, p, r, s, q\rangle\end{array}}{\langle h, N \leftarrow \alpha M \bullet \beta, i, u \cup r, v \cup s, k\rangle} \quad \mathrm{label}(M) = \mathrm{label}(R_t) \qquad \text{(AdjoinPf)}$$



Raw complexity

Maximal time complexity provided by (AdjoinPf) : $O(n^{10})$ because of 10 indexes

$$\frac{\begin{array}{c}\langle h, N \leftarrow \alpha \bullet M\beta, i, u, v, j\rangle \\ \langle j, \top \leftarrow R_t\bullet, j, p, q, k\rangle \\ \langle h, M \leftarrow \gamma\bullet, p, r, s, q\rangle\end{array}}{\langle h, N \leftarrow \alpha M \bullet \beta, i, u \cup r, v \cup s, k\rangle} \quad \mathrm{label}(M) = \mathrm{label}(R_t) \qquad \text{(AdjoinPf)}$$

But (u, v) or (r, s) equals $(-, -)$
$\Longrightarrow$ (Case analysis) splitting rule into 2 sub-rules $\Longrightarrow O(n^8) \Longrightarrow$ not sufficient !

Splitting and intermediary structures

Split (AdjoinPf) into 2 successive steps with an intermediary structure

$$[M \leftarrow \gamma \bullet, j, r, s, k]$$

This intermediary structure combines the aux. tree with the subtree rooted at M

$$\frac{\begin{array}{c}\langle j, \top \leftarrow R_t \bullet, j, p, q, k\rangle \\ \langle h, M \leftarrow \gamma \bullet, p, r, s, q\rangle\end{array}}{[M \leftarrow \gamma \bullet, j, r, s, k]} \qquad \text{(AdjoinPf-1)}$$

$$\frac{\begin{array}{c}\langle h, N \leftarrow \alpha \bullet M\beta, i, u, v, j\rangle \\ [M \leftarrow \gamma \bullet, j, r, s, k] \\ \langle h, M \leftarrow \gamma \bullet, p, r, s, q\rangle\end{array}}{\langle h, N \leftarrow \alpha M \bullet \beta, i, u \cup r, v \cup s, k\rangle} \qquad \text{(AdjoinPf-2)}$$

Projection

$$\frac{\begin{array}{c}\langle j, \top \leftarrow R_t \bullet, j, p, q, k\rangle \\ \langle h, M \leftarrow \gamma \bullet, p, r, s, q\rangle\end{array}}{[M \leftarrow \gamma \bullet, j, r, s, k]} \quad \text{(AdjoinPf-1)}$$

Involves 7 indexes $\{j, p, q, k, h, r, s\}$ but h not consulted

$$\frac{\langle h, M \leftarrow \gamma \bullet, p, r, s, q\rangle}{\langle \star, M \leftarrow \gamma \bullet, p, r, s, q\rangle} \quad \text{(Proj)}$$

$$\frac{\begin{array}{c}\langle j, \top \leftarrow R_t \bullet, j, p, q, k\rangle \\ \langle \star, M \leftarrow \gamma \bullet, p, r, s, q\rangle\end{array}}{[M \leftarrow \gamma \bullet, j, r, s, k]} \quad \text{(AdjoinPf-1)}$$

Finally, $O(n^6)$ time complexity

Case of (AdjoinPf-2)

$$\frac{\begin{array}{c}\langle h, N \leftarrow \alpha \bullet M\beta, i, u, v, j\rangle \\ [M \leftarrow \gamma\bullet, j, r, s, k] \\ \langle h, M \leftarrow \gamma\bullet, p, r, s, q\rangle\end{array}}{\langle h, N \leftarrow \alpha M \bullet \beta, i, u \cup r, v \cup s, k\rangle} \quad \text{(AdjoinPf-2)}$$

10 indexes $\Longrightarrow$ Raw complexity in $O(n^{10})$

At least one pair in (u, v) or (r, s) equals $(-, -)$;
Case splitting $\Longrightarrow O(n^8)$

Pair (p,q) not consulted; projection $\Longrightarrow O(n^6)$

Preliminary conclusion

Rule splitting, intermediary structures, and projections decrease complexities but increase the number of steps
To be practically validated !

Designing a tabular algorithm for TAGs is complex!

- Designing items
- Understanding the invariants
- Formulating the deductive rules (simultaneously handling tabulation and strategy)
- Optimizing rules (splitting and projections)

How to adapt for close formalisms such as Linear Indexed Grammars [LIG]?

$$A_0([\circ \circ x]) \leftarrow A_1([]) \dots A_k([\circ \circ y]) \dots A_n([])$$

LIG

Indexed Grammars: Context-Free grammars with non terminals decorated with stacks
Linear Indexed Grammars: a single stack propagated per production

A LIG $G = (\mathcal{N}, \Sigma, \mathcal{I}, S, \mathcal{P})$ where

- $\mathcal{I}$ is a finite set of indices
- $\mathcal{P}$ is a finite set of productions of the form

  $$A[\circ\circ\alpha] \rightarrow A_1[] \ldots A_i[\circ\circ\beta] \ldots A_n[]$$

  or

  $$A[] \rightarrow \gamma$$

  with $\gamma \in \Sigma^\star$ and $\alpha, \beta \in \mathcal{I}^\star$

Relationship with (linear monadic) Context-Free Tree Languages

LIGs and TAGs

LIGs and TAGs are weakly equivalent, and almost strongly equivalent

TAGs may be easily encoded by LIGs, using tree nodes as non-terminals

- adjoining node ν in γ using aux. tree β

$$\nu[\circ\circ] \rightarrow r_\beta[\circ\circ\,\nu]$$

- discharging a node ν with children $\nu_1, \ldots, \nu_n$ at a foot node f_β

$$f_\beta[\circ\circ\,\nu] \leftarrow \nu_1[\alpha_1] \ldots \nu_n[\alpha_n]$$

  where $\alpha_i = [\circ\circ]$ if ν_i on spine, and $\alpha_i = []$ otherwise
- traversing a node ν without adjoining

$$\nu[\circ\circ] \leftarrow \nu_1[\alpha_1] \ldots \nu_n[\alpha_n]$$

  with same conditions on α_i than above

Reverse way more difficult: no locality constraint between push and pop points (same aux. tree β for TAGs)

Suggest using LPDAs to parse LIGs and TAGs
but non efficient and non termination

Outline

From formalisms to automata

Methodology:

- Automata are operational devices
  used to describe the steps of Parsing Strategies
- Dynamic Programming interpretations of automata used to identify context-free subderivations that may be tabulated.

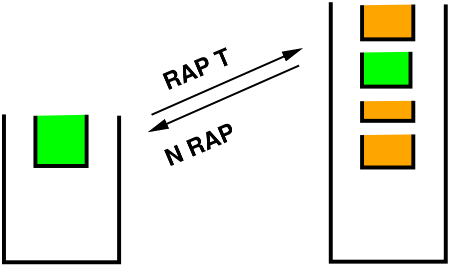
Formalisms	Automata	Tabulation	Notes
RegExp	FSA	-	
CFG	PDA	$O(n^3)$	Lang
TAG / LIG	2-Stack Automata	$O(n^6)$	Becker, Clergerie & Pardo
	Embedded PDA	$O(n^6)$	Nederhof

Problem: 2-stack automata (or EPDA) have the power of Turing Machine (intuition) moving left- or rightward $\equiv$ pushing on first or second stack & popping the other one
$\Longrightarrow$ need restrictions

EPDA

Embedded Push-Down Automata Becker are natural candidates for LIGs (and TAGs) by handling stack of stacks.

Two flavors: Top-Down and Bottom-Up EPDAs



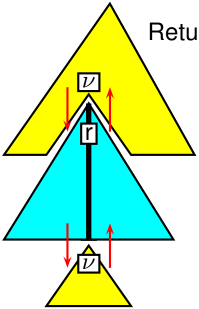
Solution: stack asymmetry

Master Stack: to keep trace of uncompleted tree traversals

Auxiliary Stack: only to keep trace of uncompleted adjunctions
Adjunction info: (top-down) $\overline{\nu}^n = \nu$ and (bottom-up) $\underline{\nu}_n = \perp$
$^\bullet$T, T$^\bullet$, $^\bullet$B, B$^\bullet$: prediction and propagation info about top and bottom node decorations (Feature TAGs)

Calls (top-down prediction)

Returns (bottom-up propagation)



2-stack automata for TAGs

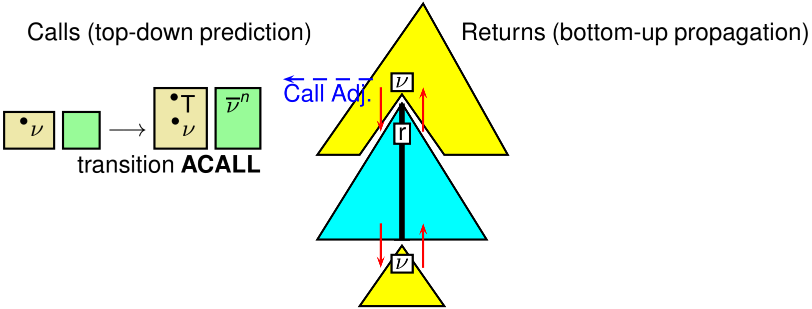
Solution: stack asymmetry

Master Stack: to keep trace of uncompleted tree traversals

Auxiliary Stack: only to keep trace of uncompleted adjunctions

Adjunction info: (top-down) $\overline{\nu}^n = \nu$ and (bottom-up) $\underline{\nu}_n = \perp$

$^\bullet$T, T$^\bullet$, $^\bullet$B, B$^\bullet$: prediction and propagation info about top and bottom node decorations (Feature TAGs)



2-stack automata for TAGs

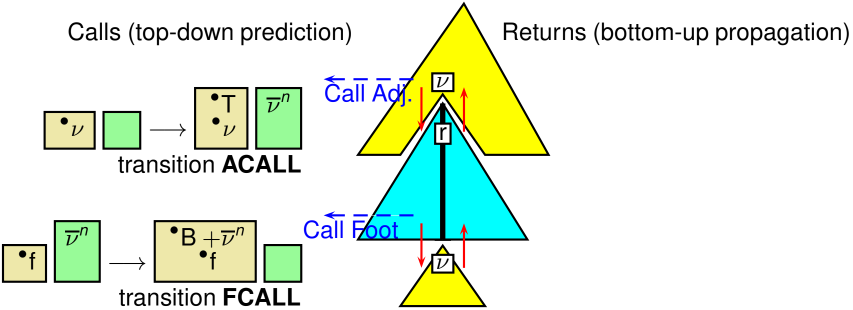
Solution: stack asymmetry

Master Stack: to keep trace of uncompleted tree traversals

Auxiliary Stack: only to keep trace of uncompleted adjunctions

Adjunction info: (top-down) $\overline{\nu}^n = \nu$ and (bottom-up) $\underline{\nu}_n = \perp$

${}^\bullet$T, T${}^\bullet$, ${}^\bullet$B, B${}^\bullet$: prediction and propagation info about top and bottom node decorations (Feature TAGs)



2-stack automata for TAGs

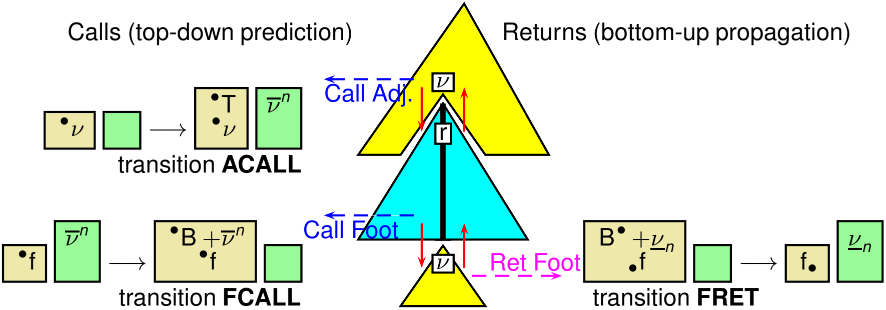
Solution: stack asymmetry

Master Stack: to keep trace of uncompleted tree traversals

Auxiliary Stack: only to keep trace of uncompleted adjunctions

Adjunction info: (top-down) $\overline{\nu}^n = \nu$ and (bottom-up) $\underline{\nu}_n = \perp$

$^\bullet$T, T$^\bullet$, $^\bullet$B, B$^\bullet$: prediction and propagation info about top and bottom node decorations (Feature TAGs)



2-stack automata for TAGs

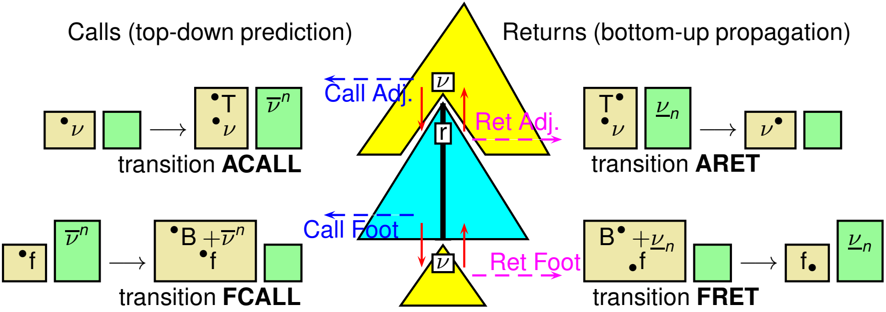
Solution: stack asymmetry

Master Stack: to keep trace of uncompleted tree traversals

Auxiliary Stack: only to keep trace of uncompleted adjunctions

Adjunction info: (top-down) $\overline{\nu}^n = \nu$ and (bottom-up) $\underline{\nu}_n = \perp$

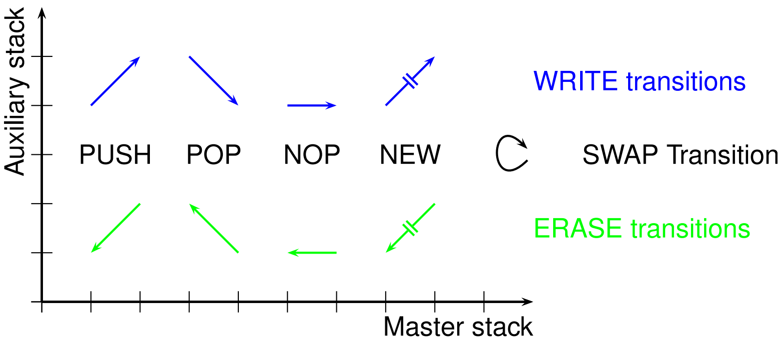
${}^\bullet$T, T${}^\bullet$, ${}^\bullet$B, B${}^\bullet$: prediction and propagation info about top and bottom node decorations (Feature TAGs)



Transitions

Retracing in erase mode concerns only the size of **AS** (not its content).
Retracing possible because :

> *WRITE transitions leave marks (PUSH, POP, NOP, NEW) in the Master Stack that can only be removed by a dual ERASE transition.*



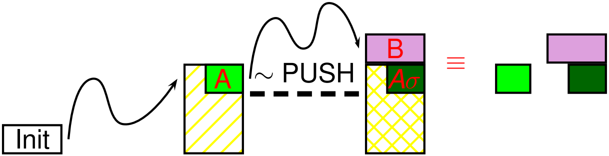
Context-Free derivation for PDAs

Dynamic Programming : Recursive decomposition of problems into elementary subproblems that may be combined, tabulated, and reused
eg the knapsack problem

Context-Free derivation for PDAs

Dynamic Programming : Recursive decomposition of problems into elementary subproblems that may be combined, tabulated, and reused
eg the knapsack problem

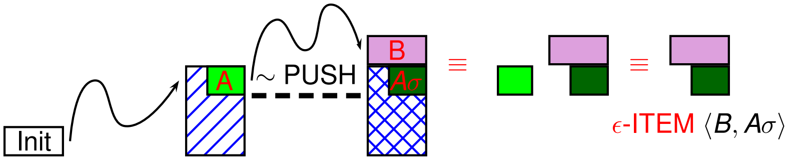
For PDAs, derivations broken into elementary Context-Free sub-derivations:



Context-Free derivation for PDAs

Dynamic Programming : Recursive decomposition of problems into elementary subproblems that may be combined, tabulated, and reused
eg the knapsack problem

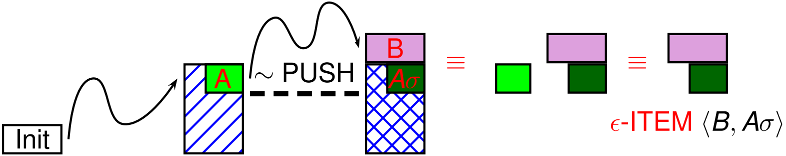
For PDAs, derivations broken into elementary Context-Free sub-derivations:



Context-Free derivation for PDAs

Dynamic Programming : Recursive decomposition of problems into elementary subproblems that may be combined, tabulated, and reused
eg the knapsack problem

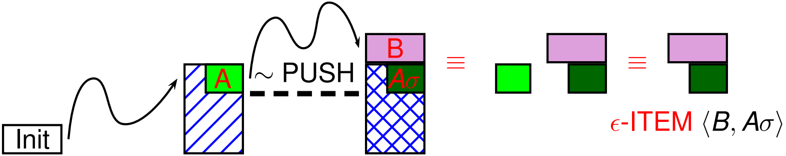
For PDAs, derivations broken into elementary Context-Free sub-derivations:



Context-Free derivation for PDAs

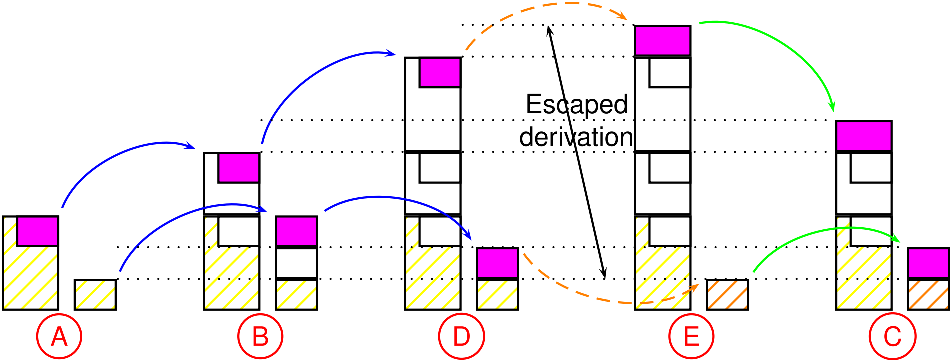
Dynamic Programming : Recursive decomposition of problems into elementary subproblems that may be combined, tabulated, and reused
eg the knapsack problem

For PDAs, derivations broken into elementary Context-Free sub-derivations:

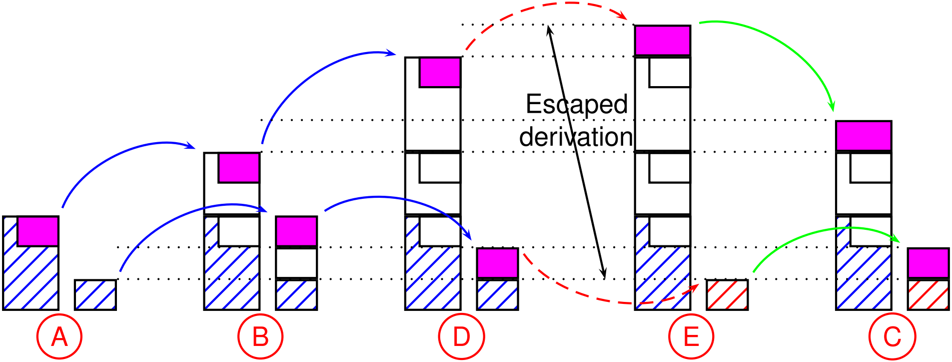


A is the fraction ϵ of information consulted to trigger the subderivation and not propagated to B.

(Escaped) CF derivations for 2SA



(Escaped) CF derivations for 2SA

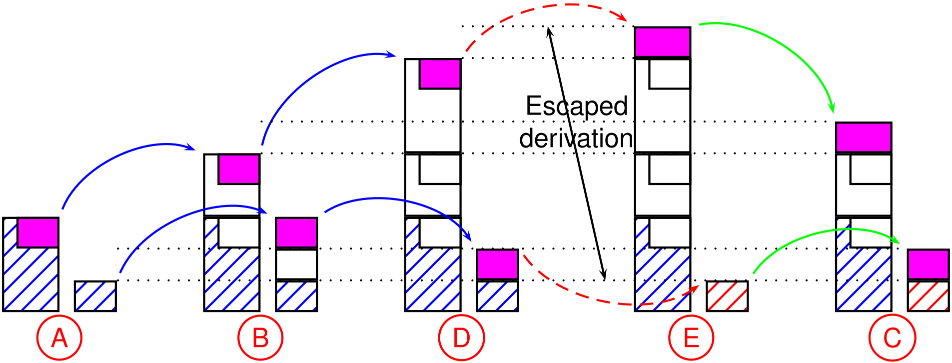


$\Longrightarrow$ 5-point xCF items $AB[DE]C = \langle\epsilon A\rangle\langle\epsilon B, b\rangle[\langle\epsilon D, d\rangle\langle E\rangle]\langle C, c\rangle$
[TAG] $\rightsquigarrow \langle\epsilon A\rangle\langle\epsilon B\rangle[\langle\epsilon D\rangle\langle E\rangle]\langle C\rangle$

When no escaped part $\Longrightarrow$ 3-point CF items $ABC = \langle\epsilon A\rangle\langle\epsilon B, b\rangle\langle C\rangle$

(new generalization) escaped part $[DE]$ may take place between A and B

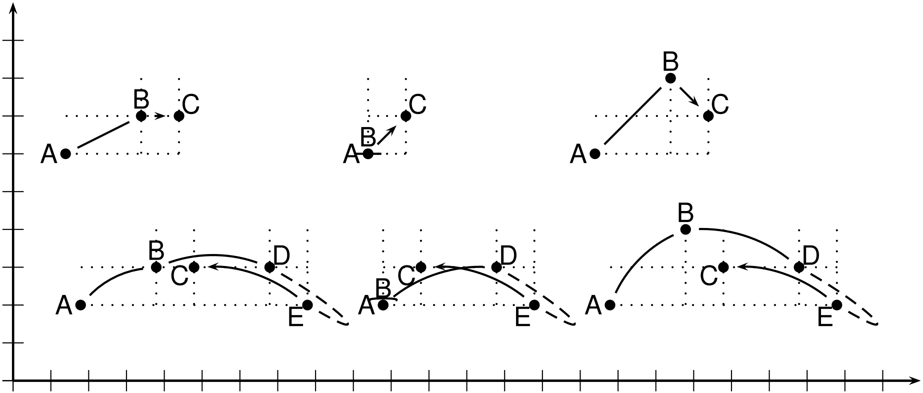
xCFs and TAGs



- *A* root of elementary tree
- *B* start of adjoining
- *C* current position in the tree
- *D* and *E* left and right borders of the foot

Item shapes

At most 5 indexes per items $\Longrightarrow$ Space complexity in $O(n^5)$
SD-2SA restrictions & transition kinds $\Longrightarrow$ 6 possible item shapes



Combining items and transitions

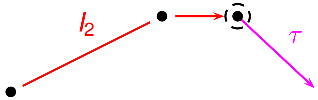
By graphically playing with items and transitions, we find 10 composition rules with $O(n^8)$ time complexity
may be split into 11 rules with $O(n^6)$ time complexity

(Easy:) Write a POP mark: $I_1 + I_2 + \tau = I_3$

Combining items and transitions

By graphically playing with items and transitions, we find 10 composition rules with $O(n^8)$ time complexity
may be split into 11 rules with $O(n^6)$ time complexity

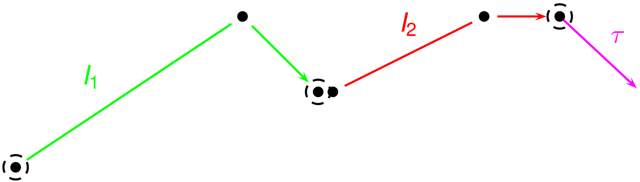
(Easy:) Write a POP mark: $I_1 + I_2 + \tau = I_3$



Combining items and transitions

By graphically playing with items and transitions, we find 10 composition rules with $O(n^8)$ time complexity
may be split into 11 rules with $O(n^6)$ time complexity

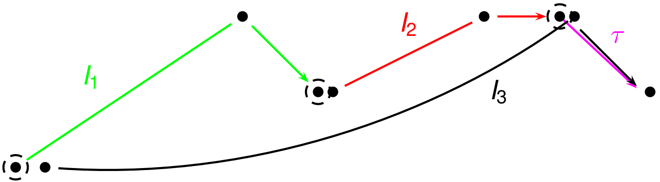
(Easy:) Write a POP mark: $I_1 + I_2 + \tau = I_3$



Combining items and transitions

By graphically playing with items and transitions, we find 10 composition rules with $O(n^8)$ time complexity
may be split into 11 rules with $O(n^6)$ time complexity

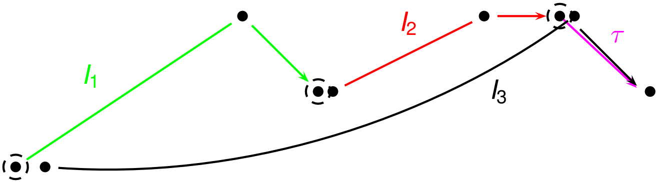
(Easy:) Write a POP mark: $I_1 + I_2 + \tau = I_3$



Combining items and transitions

By graphically playing with items and transitions, we find 10 composition rules with $O(n^8)$ time complexity
may be split into 11 rules with $O(n^6)$ time complexity

(Easy:) Write a POP mark: $I_1 + I_2 + \tau = I_3$



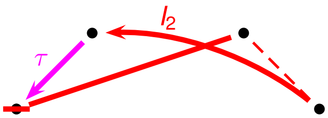
Consultation of 3 indexes ⊙ $\Longrightarrow$ Complexity $O(n^3)$

Combining items and transitions (2)

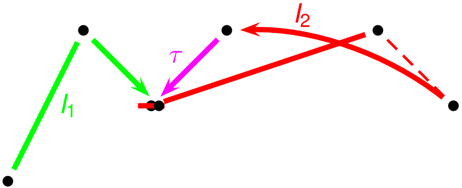
(complex:) Erasing a PUSH mark: $I_1 + I_2 + I_3 + \tau = I_4$
e.g. when returning from auxiliary tree (ending adjoining)

Combining items and transitions (2)

(complex:) Erasing a PUSH mark: $I_1 + I_2 + I_3 + \tau = I_4$
e.g. when returning from auxiliary tree (ending adjoining)

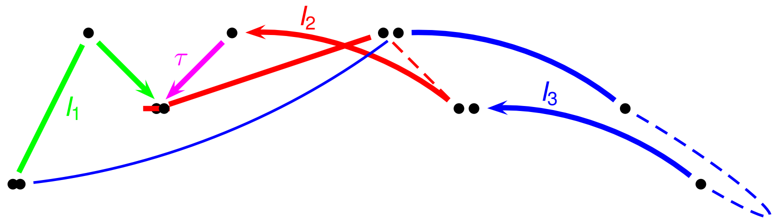


(complex:) Erasing a PUSH mark: $I_1 + I_2 + I_3 + \tau = I_4$
e.g. when returning from auxiliary tree (ending adjoining)



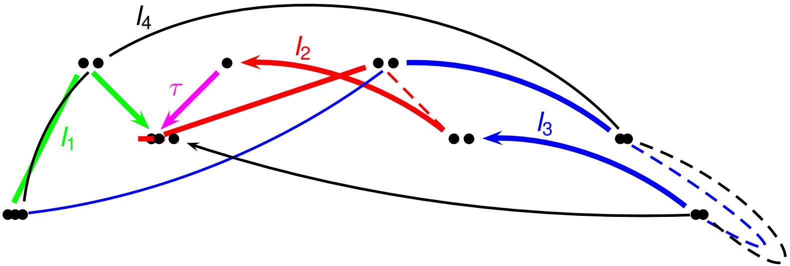
Combining items and transitions (2)

(complex:) Erasing a PUSH mark: $I_1 + I_2 + I_3 + \tau = I_4$
e.g. when returning from auxiliary tree (ending adjoining)



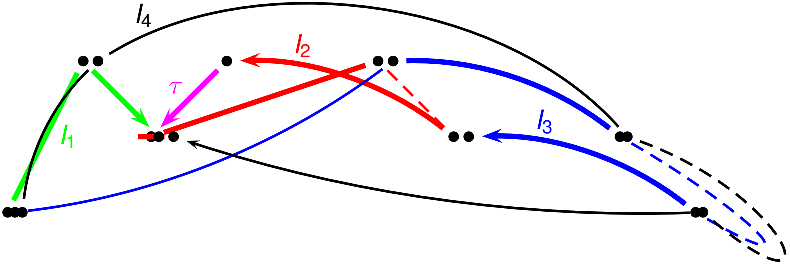
Combining items and transitions (2)

(complex:) Erasing a PUSH mark: $I_1 + I_2 + I_3 + \tau = I_4$
e.g. when returning from auxiliary tree (ending adjoining)



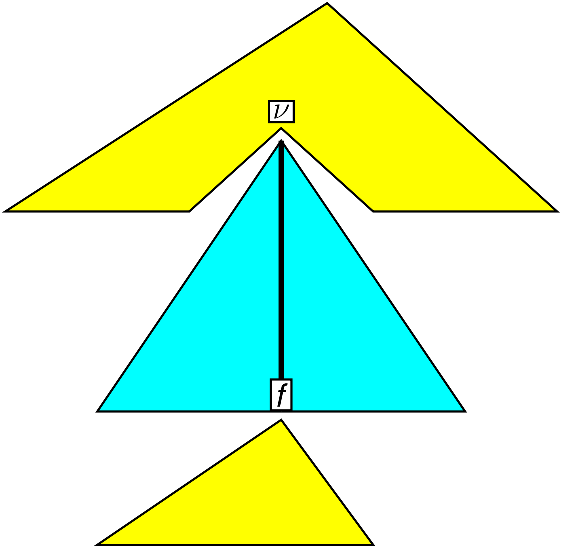
Combining items and transitions (2)

(complex:) Erasing a PUSH mark: $I_1 + I_2 + I_3 + \tau = I_4$
e.g. when returning from auxiliary tree (ending adjoining)

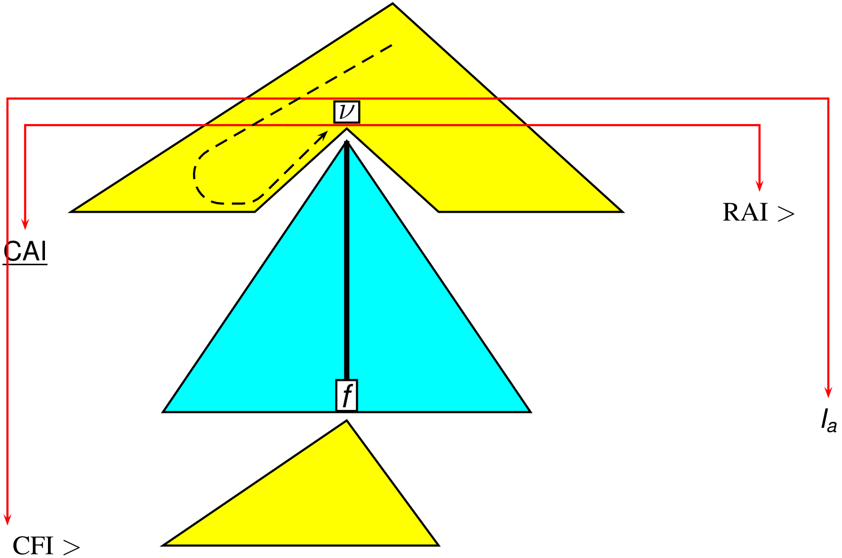


- Consultation of 8 indexes $\Longrightarrow$ Complexity $O(n^8)$
- need to decompose, project and use intermediary steps (as seen before)

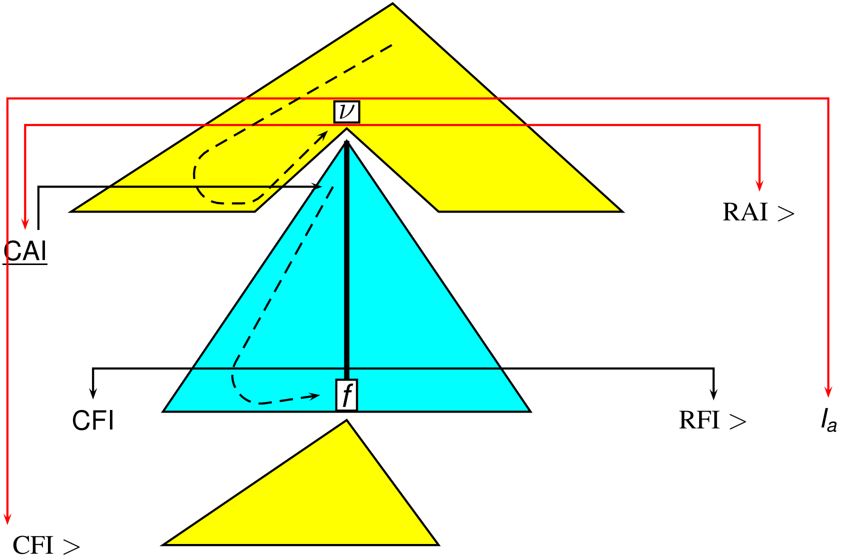
Cascade of partial evaluations



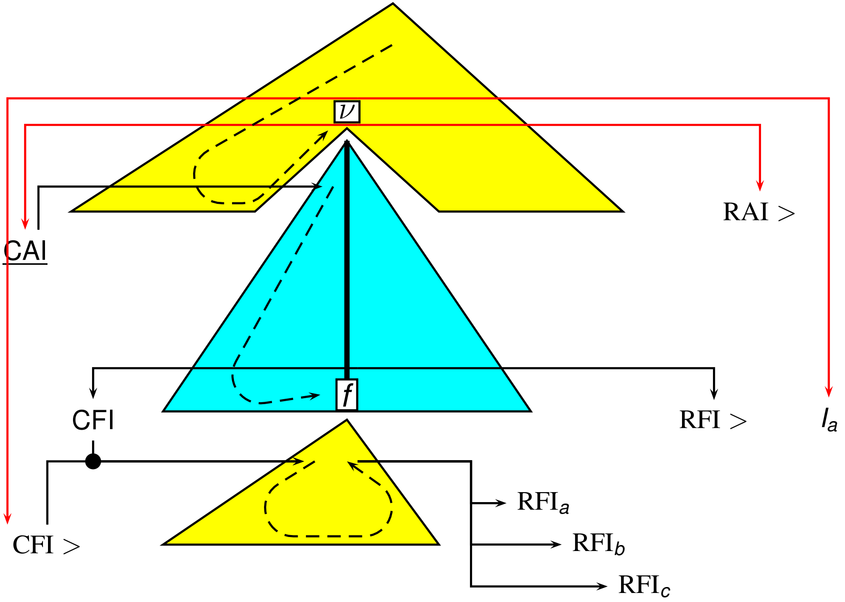
Cascade of partial evaluations



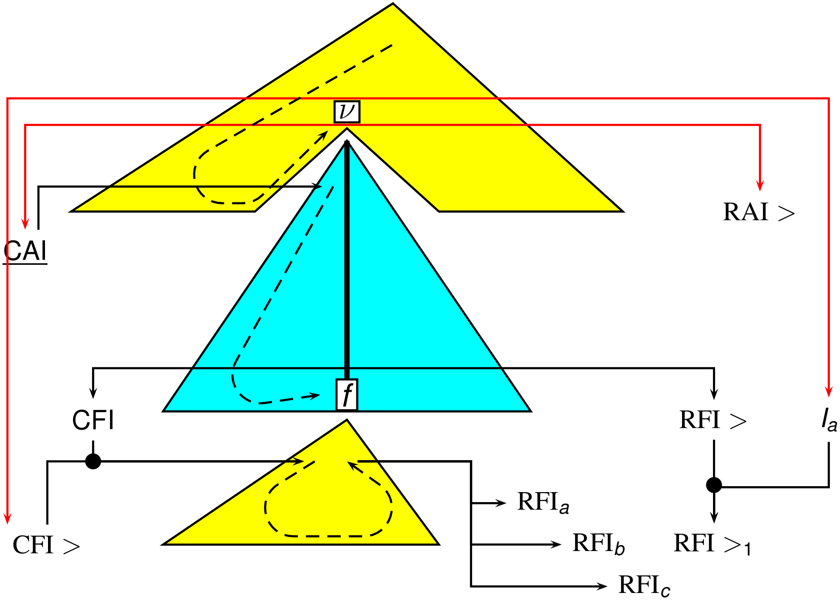
Cascade of partial evaluations



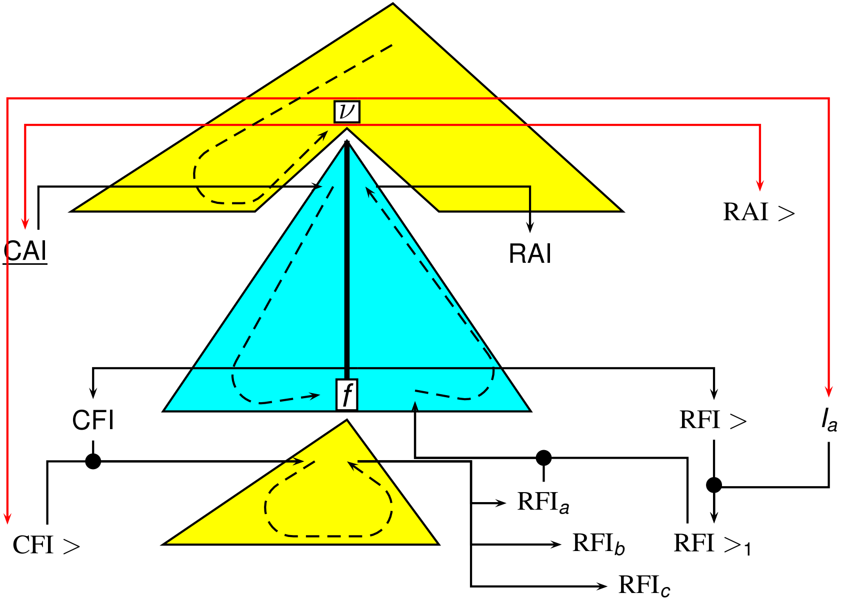
Cascade of partial evaluations



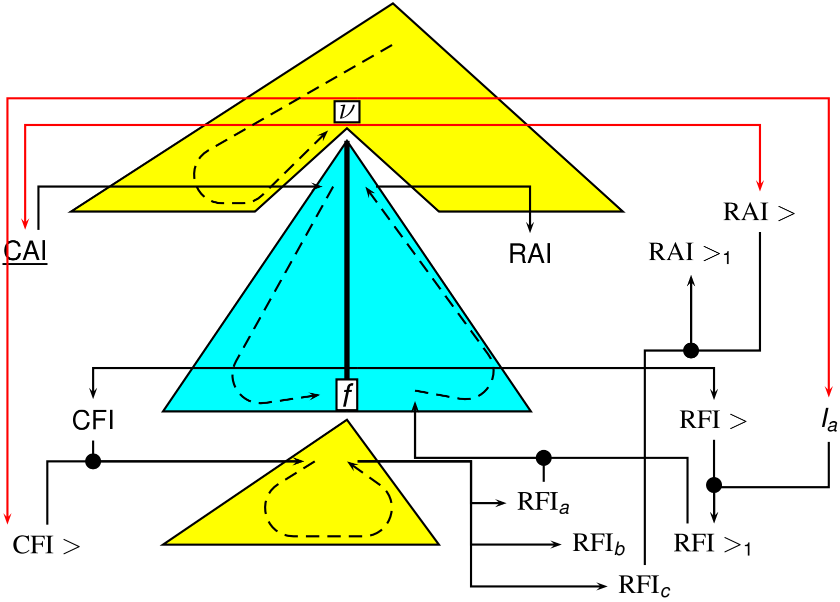
Cascade of partial evaluations



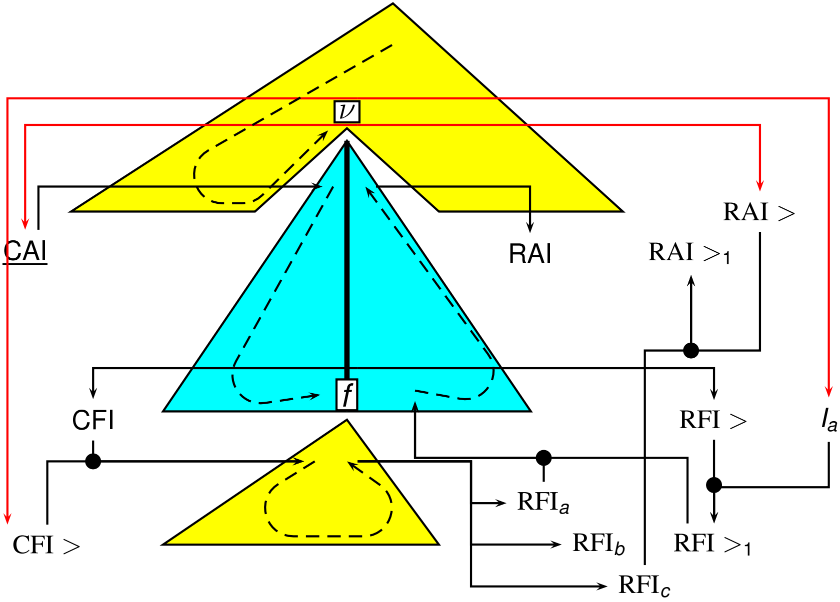
Cascade of partial evaluations



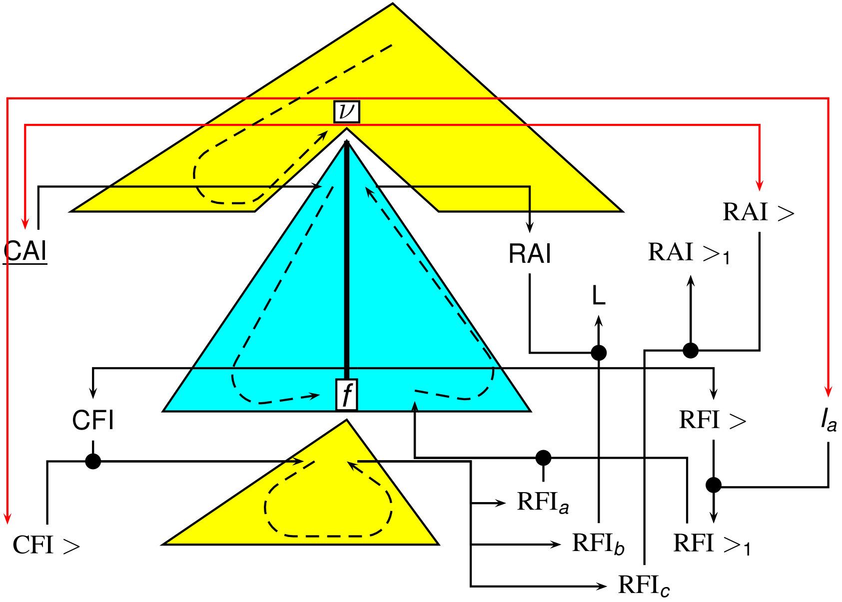
Cascade of partial evaluations



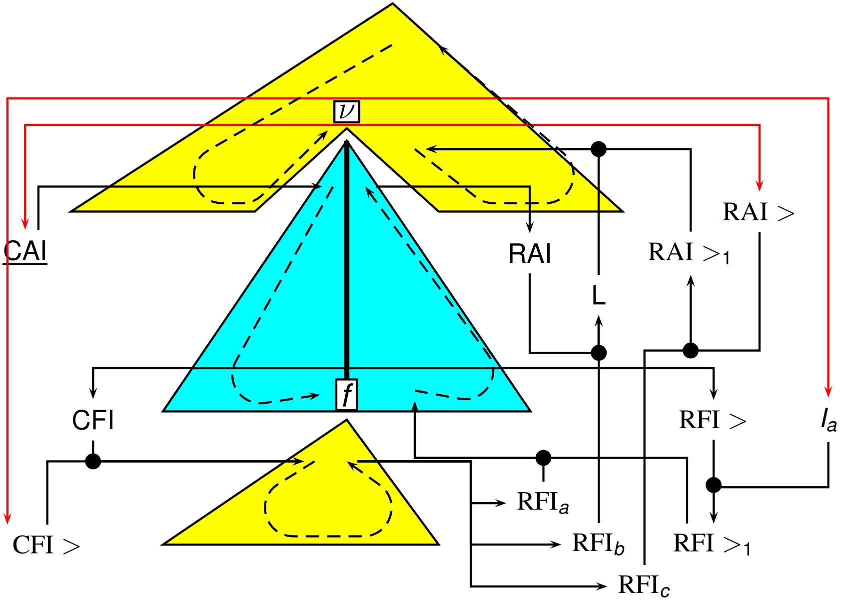
Cascade of partial evaluations



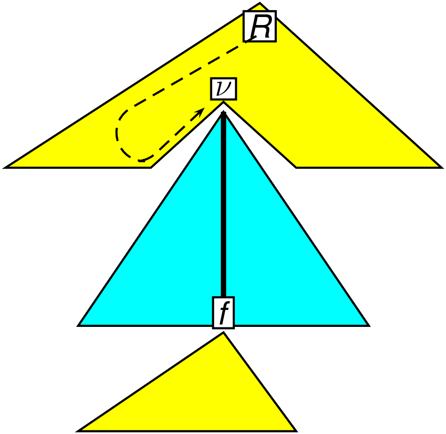
Cascade of partial evaluations



Cascade of partial evaluations

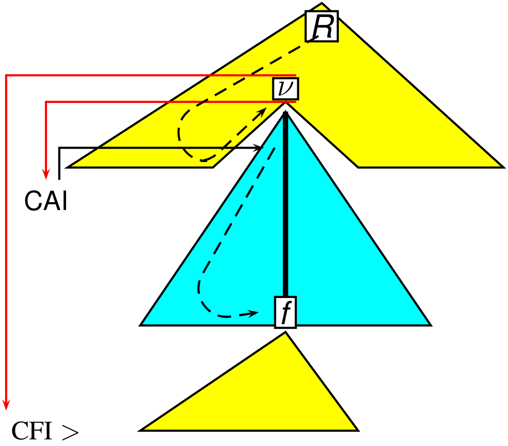


Simplified Cascade of partial evaluations



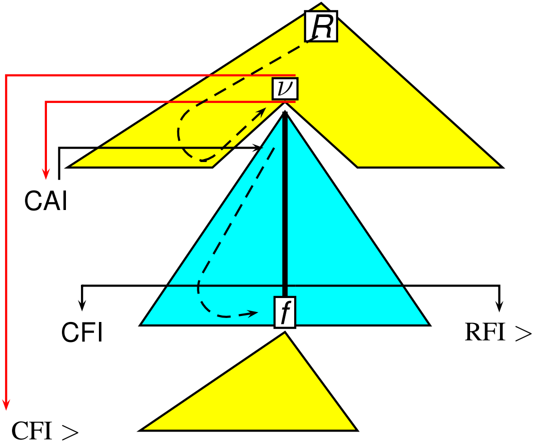
- Not the optimal worst case complexity (because yellow subtree traversed in the context of larger yellow subtree, keeping trace of unfinished adjoinings)
- But more efficient in practice !
- And suggesting extensions, based on the idea of continuation

Simplified Cascade of partial evaluations



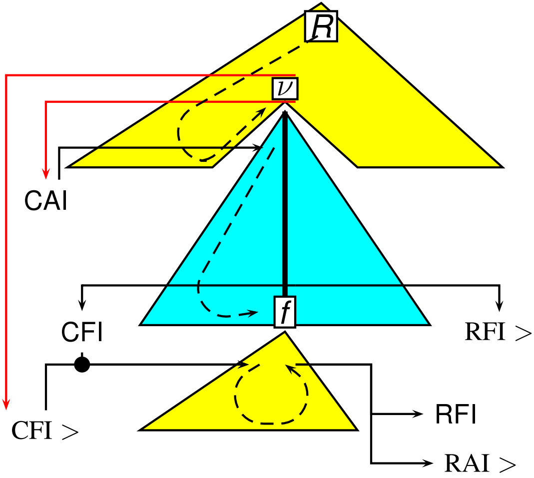
- Not the optimal worst case complexity (because yellow subtree traversed in the context of larger yellow subtree, keeping trace of unfinished adjoinings)
- But more efficient in practice !
- And suggesting extensions, based on the idea of continuation

Simplified Cascade of partial evaluations



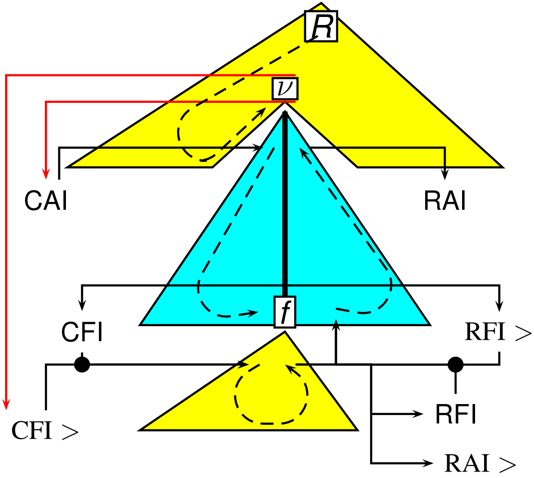
- Not the optimal worst case complexity (because yellow subtree traversed in the context of larger yellow subtree, keeping trace of unfinished adjoinings)
- But more efficient in practice !
- And suggesting extensions, based on the idea of continuation

Simplified Cascade of partial evaluations



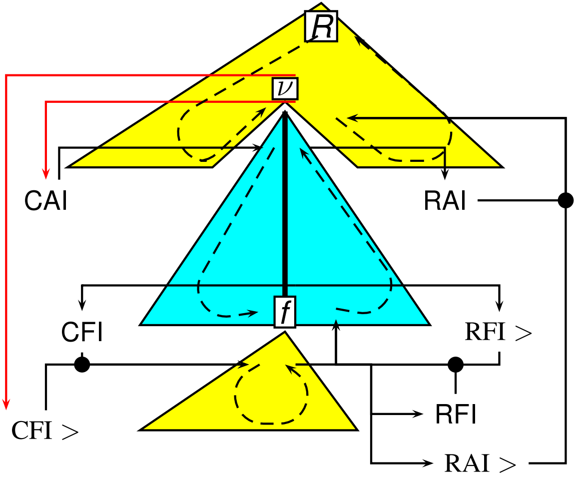
- Not the optimal worst case complexity (because yellow subtree traversed in the context of larger yellow subtree, keeping trace of unfinished adjoinings)
- But more efficient in practice !
- And suggesting extensions, based on the idea of continuation

Simplified Cascade of partial evaluations



- Not the optimal worst case complexity (because yellow subtree traversed in the context of larger yellow subtree, keeping trace of unfinished adjoinings)
- But more efficient in practice !
- And suggesting extensions, based on the idea of continuation

Simplified Cascade of partial evaluations



- Not the optimal worst case complexity (because yellow subtree traversed in the context of larger yellow subtree, keeping trace of unfinished adjoinings)
- But more efficient in practice !
- And suggesting extensions, based on the idea of continuation

Part II

MCS in general

Outline

4 Thread Automata and MCS formalisms

5 A Dynamic Programming interpretation for TAs

Mildly Context Sensitivity

An informal notion covering formalisms such that:

- they are powerful enough to model crossing, such as $a^n b^n c^n$
- they are parsable with polynomial complexity
  i.e. Given L, there exists k, membership $w \in$ checked in $O(|w|^k)$
- they generate string languages satisfying the constant growth property

  $$\exists G, G \text{ finite }, \exists n_0, \forall w \in \mathcal{L}, |w| > n_0 \Longrightarrow \exists g \in G, \exists w' \in \mathcal{L}, |w| = |w'| + g$$

  (intuition) the languages are generated by finite sets of generators

Mildly Context Sensitivity

An informal notion covering formalisms such that:

- they are powerful enough to model crossing, such as $a^n b^n c^n$
- they are parsable with polynomial complexity
  i.e. Given L, there exists k, membership $w \in$ checked in $O(|w|^k)$
- they generate string languages satisfying the constant growth property

  $$\exists G, G \text{ finite }, \exists n_0, \; \forall w \in \mathcal{L}, |w| > n_0 \Longrightarrow \; \exists g \in G, \exists w' \in \mathcal{L}, \; |w| = |w'| + g$$

  (intuition) the languages are generated by finite sets of generators

Some MCS languages:

- TAGs and LIGs
- Local Multi Component TAGs (MC-TAGs Weir)
- Linear Context-Free Rewriting Systems (LCFRS Weir)
- Simple Range Concatenation Grammars (sRCG Boullier)

Semi-linearity

The Constant Growth property subsumed by stronger semi-linearity under Parikh image

The Parikh image of $w \in \{a_1, \ldots, a_n\}^\star$ defined as $p(w) = (|w|_{a_1}, \ldots, |w|_{a_n})$

The Parikh image of L defined as $p(L) = \{p(w) | w \in L\}$

A set V of vectors over $\mathbb{N}^\Sigma$ is linear is generated by a base $v_0, v_1, \ldots, v_n \in \mathbb{N}^\Sigma$ by

$$V = \{v_0 + \Sigma_{i=1}^n k_i v_i | k_i \in \mathbb{N}\}$$

V is semilinear if $V = \bigcup_{i=1}^k V_i$ is a finite union of linear sets V_i

A language L is semilinear if $p(L)$ is semilinear

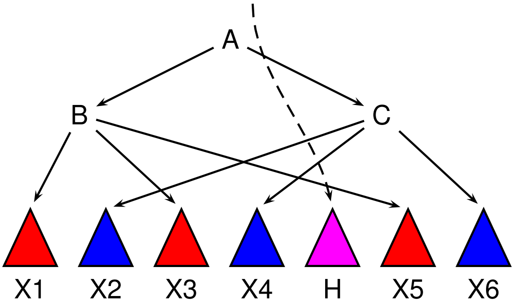
(intuition) A MCS language is generated, modulo some permutations, by a finite set of generators

MCS: discontinuity and interleaving

Discontinuous interleaved constituents present in linguistic phenomena
Nesting, Crossing, Topicalization, Deep extraction, Complex Word-Order ...

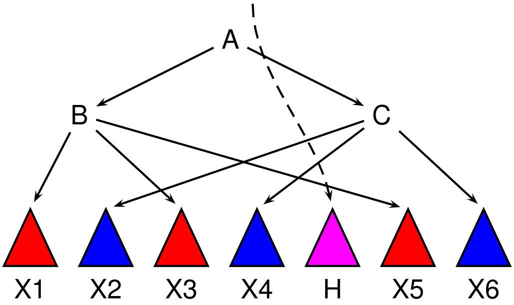
MCS: discontinuity and interleaving

Discontinuous interleaved constituents present in linguistic phenomena
Nesting, Crossing, Topicalization, Deep extraction, Complex Word-Order ...



MCS: discontinuity and interleaving

Discontinuous interleaved constituents present in linguistic phenomena
Nesting, Crossing, Topicalization, Deep extraction, Complex Word-Order ...



- **LFCRS**: $A \leftarrow f(B, C)$, f linear non erasing function on string tuples.

$$f(\langle x_1, x_3, x_5\rangle, \langle x_2, x_4, x_6\rangle) = \langle x_1 x_2 x_3 x_4, x_5 x_6\rangle$$

- **sRCG** $A(x_1.x_2.x_3.x_4, x_5.x_6) \leftarrow B(x_1, x_3, x_5), C(x_2, x_4, x_6)$
range variables x_i; concatenation "."; holes ","

Linear Context-Free Rewriting Systems (LCFRS) , a restricted form of *generalized CFGs*

A LCFRS is a tuple $G = (\mathcal{N}, \Sigma, S, \mathcal{P}, \mathcal{F})$ where

- $\mathcal{P}$ is a finite set of productions as follows, with $f \in \mathcal{F}$

$$A \leftarrow f(A_1, \ldots, A_n)$$

- $\mathcal{F}$ is a set of linear regular operations over tuples of strings in $\Sigma^\star$

$$f(\langle x_{1,1}, \ldots, x_{1,k_1} \rangle, \ldots, \langle x_{n,1}, \ldots, x_{1,k_n} \rangle) = \langle t_1, \ldots t_k \rangle$$

  where $V = \{x_{i,j}\}$ are variables (over $\Sigma^\star$) and $t_i \in (\Sigma \cup V)^\star$ and
  - (regular or non-erasing) $\forall x_{i,j}, \exists t_u, x_{i,j} \in t_u$
  - (linear) $\forall x_{i,j}, x_{i,j} \in t_u \wedge x_{i,j} \in t_v \Longrightarrow u = v$

Assuming $arity(S) = 1$,

$$L(G) = \{w | S \Longrightarrow \langle w \rangle\}$$

where

$$\begin{array}{ll} A \Longrightarrow f() & \text{if } A \rightarrow f() \in \mathcal{P} \\ A \Longrightarrow f(t_1, \ldots, t_n) & \text{if } A \rightarrow f(A_1, \ldots, A_n) \in \mathcal{P} \ \wedge \forall i,\ A_i \Longrightarrow t_i \end{array}$$

RCG

Range Concatenation Grammars (RCG) [Boullier] :
Constraints on intervals on the input string.
For language $a^n b^n c^n$

```
S(X @ Y @ Z) --> A(X,Y,Z).
A("a" @ X, "b" @ Y, "c" @ Z) -->
                 A(X,Y,Z).
A("","","") --> .
```

aabbcc	$S(]0,6]) \longrightarrow$
aabbcc	$A(]0,2],]2,4],]4,6]) \longrightarrow$
aabbcc	$A(]1,2],]3,4],]5,6]) \longrightarrow$
aabbcc	$A(]2,2],]4,4],]6,6]) \longrightarrow$

RCG is an operational formalism for encoding linguistic formalisms where discontinuous constituents are used.

RCG allow modular grammar writing

concatenation G(X @ Y) --> G1(X),G2(Y).

union G(X) --> G1(X) | G2(X).

intersection G(X) --> G1(X),G2(X).

Linear non-erasing positive RCGs equivalent to LCFRS
Full RCGs are PTIME (equivalent to Datalog)

Parsing MCS

- MCS have theoretical polynomial complexity $O(n^u)$ depending upon
  - degree of discontinuity, (also fanout, arity)
  - degree of interleaving, (also rank)

- But no uniform framework to express parsing strategies and tabular algorithms
  - operational device: Deterministic Tree Walking Transducer (Weir), but no tabular algorithm
  - operational formalism sRCG with tabular algorithm (Boullier) but not for prefix-valid strategies

Notion of Thread Automata to model discontinuity and interleaving through the suspension/resume of threads.

Tree Walking Automata

TWA may be used to check properties of (binary) trees
(by accepting or rejecting them)

A (non-deterministic) TWA is a tuple $A = (\mathcal{Q}, \Sigma, I, F, R, \delta)$ where

- $\mathcal{Q}$ is a finite set of states
- Σ a finite set of node labels
- $I, F, R \subset \mathcal{Q}$ the initial, accepting, rejecting states
- δ the finite set of transitions in $\mathcal{Q} \times \Sigma \times \mathrm{Pred} \times \mathrm{Dirs} \times \mathcal{Q}$ where
  - $\mathrm{Pred} \subset \{\mathrm{root}, \mathrm{left}, \mathrm{right}, \texttt{leaf}\}$ is a set of predicates for testing nodes
  - $\mathrm{Dirs} \subset \{\mathrm{stay}, \mathrm{up}, \mathrm{left}, \mathrm{right}\}$ a set of directions

  Deterministic TWA: $\delta : \mathcal{Q} \times \Sigma \times \mathrm{Pred} \mapsto \mathrm{Dirs} \times \mathcal{Q}$

Given a Σ-tree $\tau = (V, E)$, a configuration is given by $(\nu, q) \in V \times \mathcal{Q}$

Extensions: Pebble Automata (Engelfriet)

Similar to TWA, but emits strings when walking over a tree (in some tree set)

A deterministic TWD (Weir) is a tuple $T = (\mathcal{Q}, G, \Sigma_O, q_I, F, \delta)$ where

- $G = (\mathcal{N}, \Sigma_I, S, \mathcal{P})$ is a CFG
- Σ_O a finite set of output symbols
- $\delta : \mathcal{Q} \times (\mathcal{N} \cup \Sigma_I \cup \{\epsilon\}) \mapsto \mathrm{Dirs} \times \mathcal{Q} \times \Sigma_O$
  with $\mathrm{Dirs} = \{\mathrm{stay}, \mathrm{up}, \mathrm{down_1}, \ldots, \mathrm{down_n}\}$

A transition step given by

$$(q, \gamma, \nu, w) \stackrel{\star}{\vdash} (q', \gamma, \nu', w.v) \text{ if } \begin{cases} (q, \mathrm{dir}) = \delta(q, \mathrm{label}(\nu)) \\ \nu' = \mathrm{dir}(\nu) \end{cases}$$

The language generated by T defined as

$$L(T) = \{w | (q_I, \gamma, r_\gamma, \epsilon) \stackrel{\star}{\vdash} (q_f, \gamma, \uparrow, w)\}$$

with $q_f \in F$, γ a derivation tree for G with root r_γ
and $\uparrow$ a virtual node parent of r_γ

Weir's result: $L(\mathrm{DTWD}) = \mathrm{LCFRL}$

Thread Automata

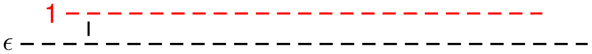
Idea: Associate a thread p per constituent and

- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity,
  and (resume) either the parent thread [SPOP]
  or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

Idea: Associate a thread p per constituent and

- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity,
  and (resume) either the parent thread [SPOP]
  or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

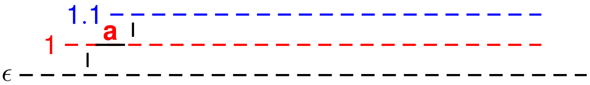
Recognize $aaabbbccc \in a^n b^n c^n$



Idea: Associate a thread p per constituent and

- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity,
  and (resume) either the parent thread [SPOP]
  or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

Recognize $aaabbbccc \in a^n b^n c^n$

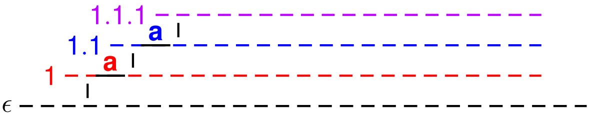


Thread Automata

Idea: Associate a thread p per constituent and

- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity,
  and (resume) either the parent thread [SPOP]
  or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

Recognize $aaabbbccc \in a^n b^n c^n$

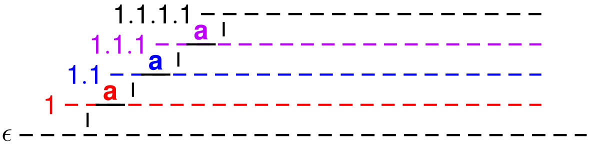


Thread Automata

Idea: Associate a thread p per constituent and

- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity, and (resume) either the parent thread [SPOP] or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

Recognize $aaabbbccc \in a^n b^n c^n$

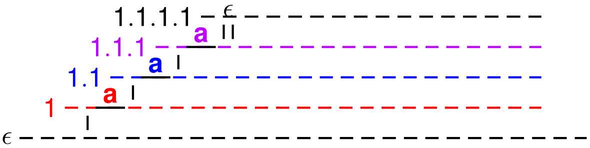


Thread Automata

Idea: Associate a thread p per constituent and

- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity,
  and (resume) either the parent thread [SPOP]
  or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

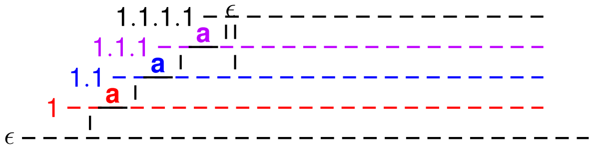
Recognize $aaabbbccc \in a^n b^n c^n$



Idea: Associate a thread p per constituent and

- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity,
  and (resume) either the parent thread [SPOP]
  or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

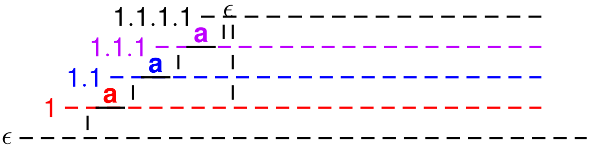
Recognize $aaabbbccc \in a^n b^n c^n$



Idea: Associate a thread p per constituent and

- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity,
  and (resume) either the parent thread [SPOP]
  or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

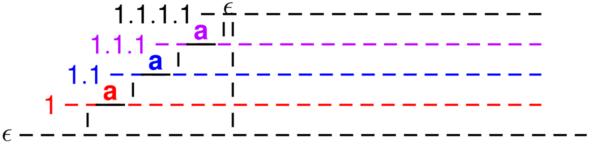
Recognize $aaabbbccc \in a^n b^n c^n$



Idea: Associate a thread p per constituent and

- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity,
  and (resume) either the parent thread [SPOP]
  or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

Recognize $aaabbbccc \in a^n b^n c^n$

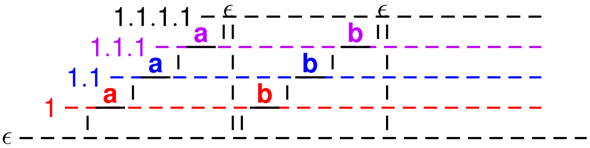


Thread Automata

Idea: Associate a thread p per constituent and

- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity,
  and (resume) either the parent thread [SPOP]
  or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

Recognize $aaabbbccc \in a^n b^n c^n$

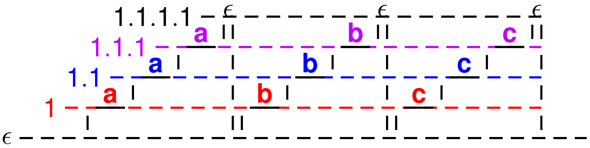


Thread Automata

Idea: Associate a thread p per constituent and

- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity,
  and (resume) either the parent thread [SPOP]
  or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

Recognize $aaabbbccc \in a^n b^n c^n$



Formal presentation of TA

Configuration $\langle$position l, active thread path p, thread store $\mathcal{S} = \{p_i{:}A_i\}\rangle$
$\mathcal{S}$ closed by prefix: $p.u \in \mathrm{dom}(\mathcal{S}) \Longrightarrow p \in \mathrm{dom}(\mathcal{S})$
Note: stateless automata (but no problem for variants with states)

Triggering function $a = \Phi(A)$ amount of information needed to trigger transitions.
$\Longrightarrow$ useful to get linear compexity $O(|G|)$ w.r.t. grammar size $|G|$
Default: Φ = Identity

Driver function $u \in \delta(A)$ Drive thread creations and suspensions
$\Longrightarrow$ reduce number of transitions
(TA variants without δ should be possible)

Formal presentation of TA (cont'd)

SWAP $B \overset{\alpha}{\longmapsto} C$: Changes the content of the active thread, possibly scanning a terminal.

$$\langle l, p, \mathcal{S} \cup p{:}B \rangle \underset{\mathcal{T}}{\mid\!\!-\!\!-} \langle l + |\alpha|, p, \mathcal{S} \cup p{:}C \rangle \qquad a_l = \alpha \text{ if } \alpha \neq \epsilon$$

PUSH $b \longmapsto [b]C$: Creates a new subthread (unless present)

$$\langle l, p, \mathcal{S} \cup p{:}B \rangle \underset{\mathcal{T}}{\mid\!\!-\!\!-} \langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}C \rangle \quad (b, u) \in \Phi\delta(B) \wedge pu \notin \mathrm{do}$$

POP $[B]C \longmapsto D$: Terminates thread pu (if no existing subthreads).

$$\langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}C \rangle \underset{\mathcal{T}}{\mid\!\!-\!\!-} \langle l, p, \mathcal{S} \cup p{:}D \rangle \qquad pu \notin \mathrm{dom}(\mathcal{S})$$

SPUSH $b[C] \longmapsto [b]D$: Resumes the subthread pu (if already created)

$$\langle l, p, \mathcal{S} \cup p{:}B \cup pu{:}C \rangle \underset{\mathcal{T}}{\mid\!\!-\!\!-} \langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}D \rangle \quad (b, u^s) \in \Phi\delta(B)$$

SPOP $[B]c \longmapsto D[c]$: Resumes the parent thread p of pu

$$\langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}C \rangle \underset{\mathcal{T}}{\mid\!\!-\!\!-} \langle l, p, \mathcal{S} \cup p{:}D \cup pu{:}C \rangle \quad (c, \perp) \in \Phi\delta(C)$$

Characterizing Thread Automata

Key parameters:

- h maximal number of suspensions to the parent thread
  h finite ensures termination (of tabular parsing)
- d maximal number of simultaneously *alive* subthreads
- l maximal number of subthreads
- s maximal number of suspensions (parent + alive subthreads)
  $s \leq h + dh \leq h + lh$

Characterizing Thread Automata

Key parameters:

- h maximal number of suspensions to the parent thread
  h finite ensures termination (of tabular parsing)
- d maximal number of simultaneously *alive* subthreads
- l maximal number of subthreads
- s maximal number of suspensions (parent + alive subthreads)
  $s \leq h + dh \leq h + lh$

Worst-case Complexity:

$$\left.\begin{array}{l}\text{space } O(n^u)\\ \text{time } O(n^{1+u})\end{array}\right\} \text{where} \left\{\begin{array}{l} u = 2 + s + x \\ x = \min(s, (l-d)(h+1))\end{array}\right.$$

$$\Longrightarrow \left\{\begin{array}{l}\text{space between } O(n^{2+2s}) \text{ and [when } l = d \text{] } O(n^{2+s})\\ \text{time between } O(n^{3+2s)}) \text{ and [when } l = d \text{] } O(n^{3+s})\end{array}\right.$$

Characterizing Thread Automata

Key parameters:

- h maximal number of suspensions to the parent thread
  h finite ensures termination (of tabular parsing)
- d maximal number of simultaneously *alive* subthreads
- l maximal number of subthreads
- s maximal number of suspensions (parent + alive subthreads)
  $s \leq h + dh \leq h + lh$

Worst-case Complexity:

$$\left.\begin{array}{l}\text{space } O(n^u)\\ \text{time } O(n^{1+u})\end{array}\right\} \text{where} \left\{\begin{array}{l} u = 2 + s + x \\ x = \min(s, (l-d)(h+1))\end{array}\right.$$

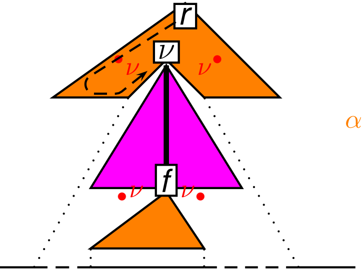
$$\Longrightarrow \left\{\begin{array}{l}\text{space between } O(n^{2+2s}) \text{ and [when } l = d] \; O(n^{2+s}) \\ \text{time between } O(n^{3+2s}) \text{ and [when } l = d] \; O(n^{3+s})\end{array}\right.$$

Push-Down Automata (PDA) for CFG $\equiv$ TA(h=0,d=1,s=0)
$\Longrightarrow$ space $O(n^2)$ and time $O(n^3)$

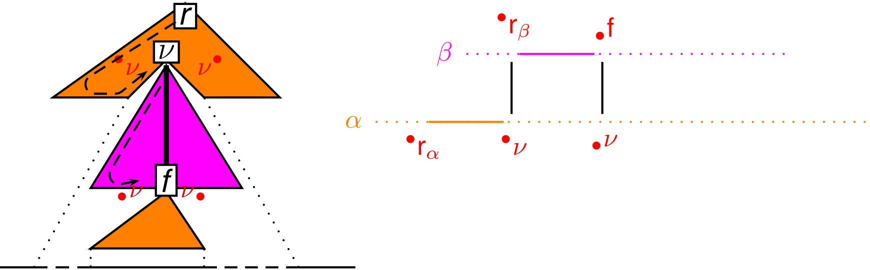
Parsing TAGs

Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node

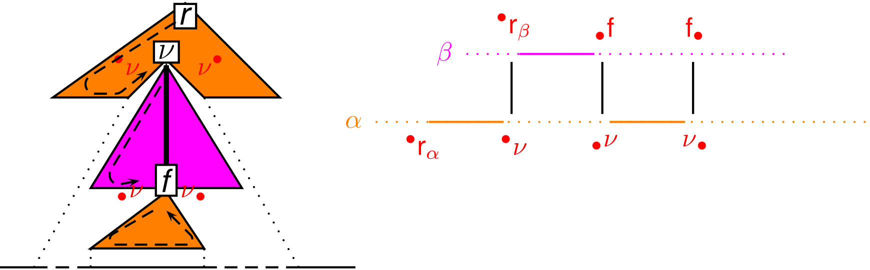
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node



Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node

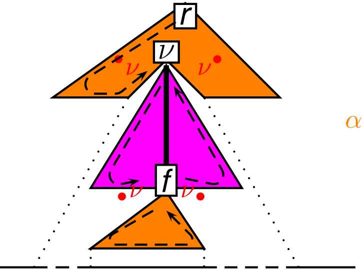


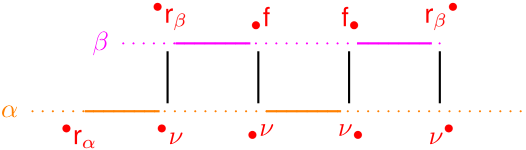
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node



Parsing TAGs

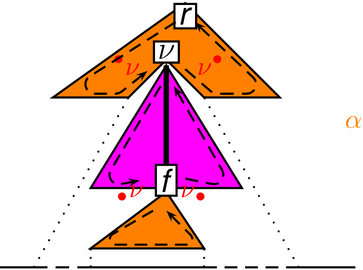
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node

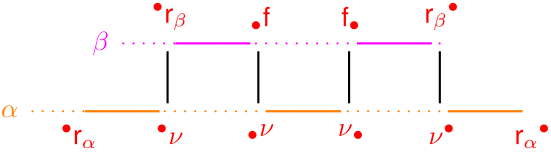




Parsing TAGs

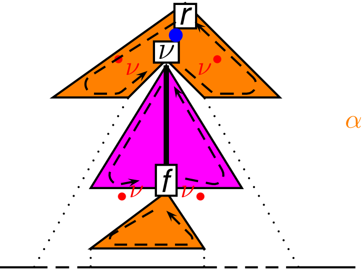
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node

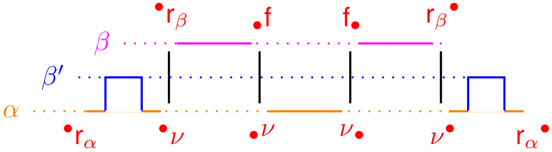




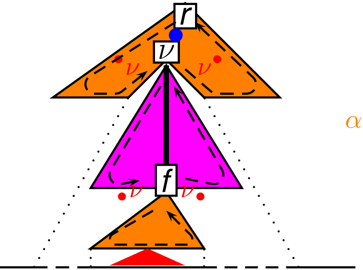
Parsing TAGs

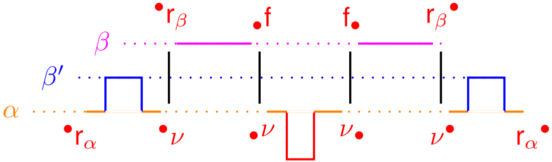
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node





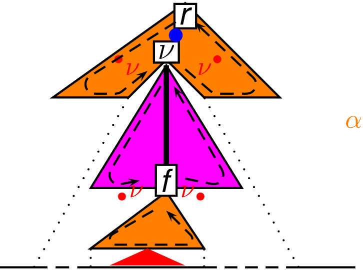
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node

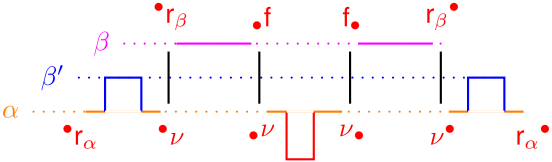




Parsing TAGs

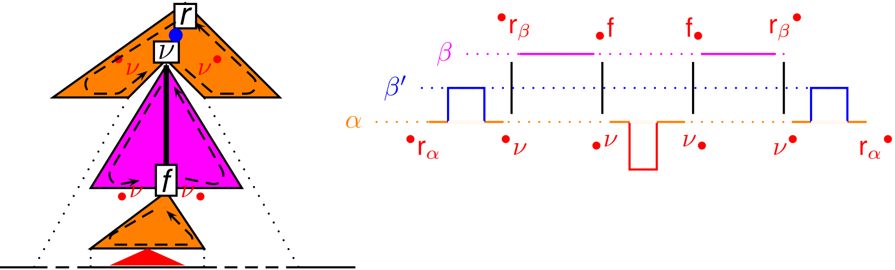
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node





Parsing TAGs

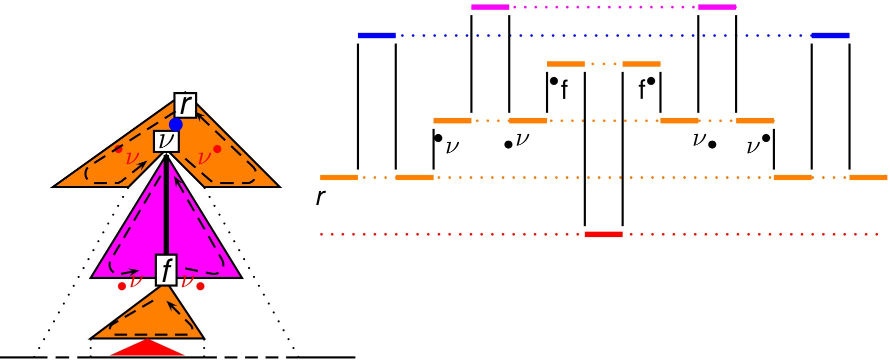
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node



One thread per tree $h = 1, d = \max(\mathrm{depth}(\mathrm{trees}))$
$\Longrightarrow [s = 1 + d]$ space $O(n^{4+2d})$ and time $O(n^{5+2d})$

Parsing TAG: an alternate parsing strategy

Using more than one thread per elementary tree: 1 thread per subtree ($\sim$ LIG)
$\Longrightarrow$ implicit extraction of subtrees
$\Longrightarrow$ implicit normal form (using a third kind of tree operation)
$\Longrightarrow$ usual n^6 time complexity



Note: Similar to a TAG encoding in RCG proposed by Boullier

Using less threads

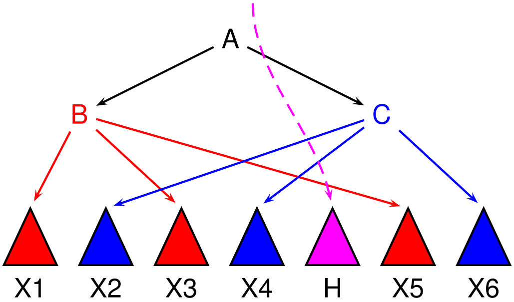
Always possible to reduce the number of live subthreads (down to 2).

- if a thread p has $d + 1$ subthreads, add a new subthread $p.v$ that inherits d subthreads of p
- generally increases the number of parent suspensions h
- but may also exploit good topological properties, such as well-nesting (TAGs).

Range Concatenation Grammars (Boullier)

$\gamma: \; A(X_1 X_2 X_3 X_4, X_5 X_6) \longrightarrow B(X_1, X_3, X_5) C(X_2, X_4, X_6)$

Ordered simple RCGs $\equiv$ Linear Context-Free Rewriting Systems (LCFRS)

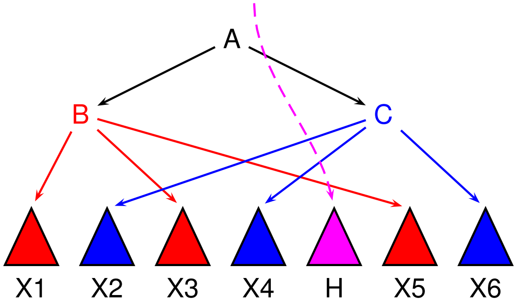


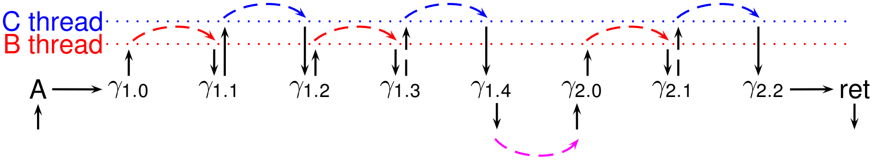
Parsing (ordered simple) RCG

Range Concatenation Grammars (Boullier)

$\gamma : A(X_1 X_2 X_3 X_4, X_5 X_6) \longrightarrow B(X_1, X_3, X_5) C(X_2, X_4, X_6)$

Ordered simple RCGs $\equiv$ Linear Context-Free Rewriting Systems (LCFRS)





Idea: assign a thread to traverse (in any order) the elementary trees of a set Σ, using extended dotted nodes $\Sigma{:}\rho\sigma$ where

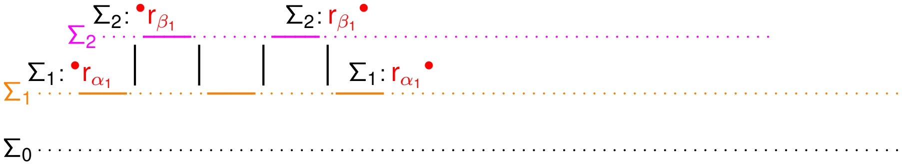
$$\begin{cases} \rho \text{ stack of dotted nodes of trees being traversed} \\ \sigma \text{ sequence of root nodes of trees already traversed} \end{cases}$$

Parsing (set-local) MC-TAGs

Idea: assign a thread to traverse (in any order) the elementary trees of a set Σ, using extended dotted nodes $\Sigma{:}\rho\sigma$ where

$$\begin{cases} \rho \text{ stack of dotted nodes of trees being traversed} \\ \sigma \text{ sequence of root nodes of trees already traversed} \end{cases}$$

Eg.: Adjoin trees of set $\Sigma_2 = \{\beta_1, \beta_2\}$ on nodes of trees of set $\Sigma_1 = \{\alpha_1, \alpha_2\}$

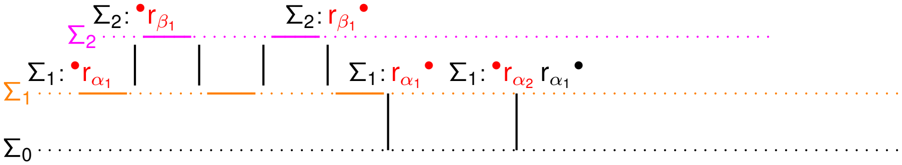


Parsing (set-local) MC-TAGs

Idea: assign a thread to traverse (in any order) the elementary trees of a set Σ, using extended dotted nodes $\Sigma{:}\rho\sigma$ where

$$\begin{cases} \rho \text{ stack of dotted nodes of trees being traversed} \\ \sigma \text{ sequence of root nodes of trees already traversed} \end{cases}$$

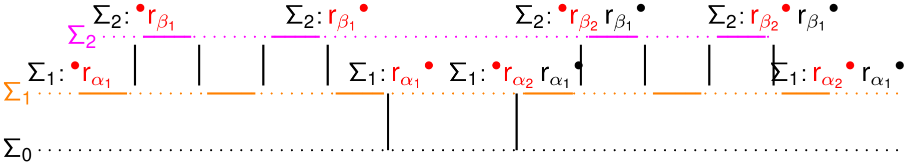
Eg.: Adjoin trees of set $\Sigma_2 = \{\beta_1, \beta_2\}$ on nodes of trees of set $\Sigma_1 = \{\alpha_1, \alpha_2\}$



Idea: assign a thread to traverse (in any order) the elementary trees of a set Σ, using extended dotted nodes $\Sigma{:}\rho\sigma$ where

$$\begin{cases} \rho \text{ stack of dotted nodes of trees being traversed} \\ \sigma \text{ sequence of root nodes of trees already traversed} \end{cases}$$

Eg.: Adjoin trees of set $\Sigma_2 = \{\beta_1, \beta_2\}$ on nodes of trees of set $\Sigma_1 = \{\alpha_1, \alpha_2\}$

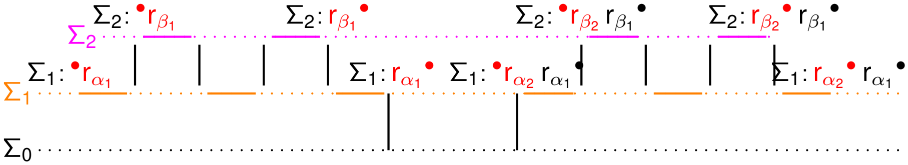


Parsing (set-local) MC-TAGs

Idea: assign a thread to traverse (in any order) the elementary trees of a set Σ, using extended dotted nodes $\Sigma{:}\rho\sigma$ where

$$\begin{cases} \rho \text{ stack of dotted nodes of trees being traversed} \\ \sigma \text{ sequence of root nodes of trees already traversed} \end{cases}$$

Eg.: Adjoin trees of set $\Sigma_2 = \{\beta_1, \beta_2\}$ on nodes of trees of set $\Sigma_1 = \{\alpha_1, \alpha_2\}$

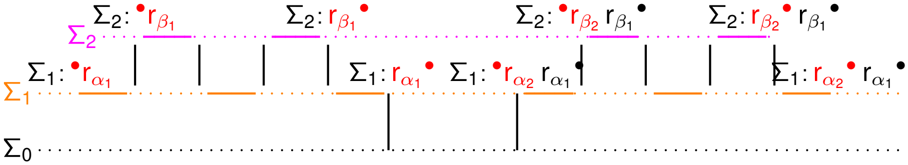


Parsing (set-local) MC-TAGs

Idea: assign a thread to traverse (in any order) the elementary trees of a set Σ, using extended dotted nodes $\Sigma{:}\rho\sigma$ where

$$\begin{cases} \rho \text{ stack of dotted nodes of trees being traversed} \\ \sigma \text{ sequence of root nodes of trees already traversed} \end{cases}$$

Eg.: Adjoin trees of set $\Sigma_2 = \{\beta_1, \beta_2\}$ on nodes of trees of set $\Sigma_1 = \{\alpha_1, \alpha_2\}$



Time complexity $O(n^{3+2(m+v)})$ where $\begin{cases} m \text{ max number of trees per set} \\ v \text{ max number of nodes per set} \end{cases}$

Outline

Dynamic Programming interpretation

Direct evaluation of TA $\leadsto$ exponential complexity and non-termination

Use tabular techniques based on Dynamic Programming interpretation of TAs:

Principle: Identification of a class of subderivations that

- may be tabulated as compact items, removing non-pertinent information
- may be combined together and with transitions to retrieve all derivations

Methodology followed for PDAs (CFGs) and 2SAs (TAGs)

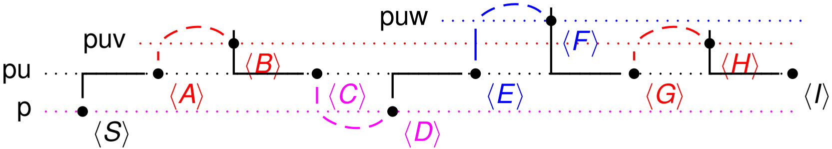
Dynamic Programming – Items

DP interpretation of TA derivations:
(Tabulated) Item $\equiv$ pertinent information about an (active) thread

1– Start point	3– (current) Parent suspensions
2– (current) End point	4– (current) Subthread suspensions for **live** subthreads

DP interpretation of TA derivations:
(Tabulated) Item $\equiv$ pertinent information about an (active) thread

1– Start point
2– (current) End point
3– (current) Parent suspensions
4– (current) Subthread suspensions for **live** subthreads

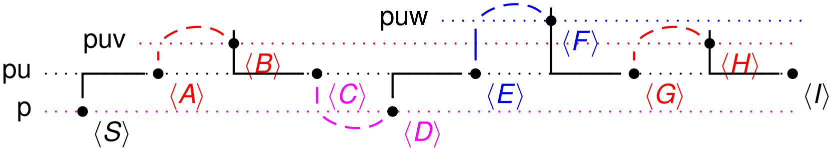


$\Longrightarrow$ Item: $\langle s\rangle / v : \langle a\rangle\langle b\rangle, \perp : \langle c\rangle\langle d\rangle, w : \langle e\rangle\langle f\rangle, v : \langle g\rangle\langle h\rangle / \langle I\rangle$

Dynamic Programming – Items

DP interpretation of TA derivations:
(Tabulated) Item $\equiv$ pertinent information about an (active) thread

1– Start point
2– (current) End point
3– (current) Parent suspensions
4– (current) Subthread suspensions for **live** subthreads



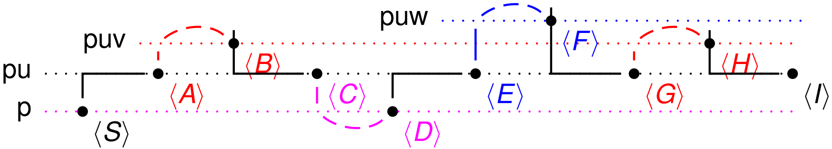
$\Longrightarrow$ Item: $\langle s\rangle / v : \langle a\rangle\langle b\rangle, \perp : \langle c\rangle\langle d\rangle, w : \langle e\rangle\langle f\rangle, v : \langle g\rangle\langle h\rangle / \langle I\rangle$

Projection $x = \Phi(X)$ used to trigger transition applications
$\Longrightarrow$ easy way to get complexity $O(|G|)$

Dynamic Programming – Items

DP interpretation of TA derivations:
(Tabulated) Item $\equiv$ pertinent information about an (active) thread

1– Start point
2– (current) End point
3– (current) Parent suspensions
4– (current) Subthread suspensions for **live** subthreads



$\Longrightarrow$ Item: $\langle s\rangle / v : \langle a\rangle\langle b\rangle, \perp : \langle c\rangle\langle d\rangle, w : \langle e\rangle\langle f\rangle, v : \langle g\rangle\langle h\rangle / \langle I\rangle$

Projection $x = \Phi(X)$ used to trigger transition applications
$\Longrightarrow$ easy way to get complexity $O(|G|)$

Space complexity:

- at most 2 indices per suspensions + start + end = $2(1+s) \leq 2(1+h+dh)$
- Scanning parts generally of fixed length (independent of n)
  $\Longrightarrow$ 1 index per suspension

Dynamic Programming – Application rules

Based on following model:

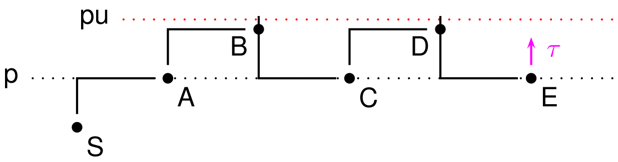
$$\frac{\text{parent item} \quad \textcolor{red}{\text{son item}} \quad \textcolor{magenta}{\text{trans}}}{\text{parent or son extension}} \quad \{\text{fitting son and parent items}\}$$

Dynamic Programming – Application rules

Based on following model:

$$\frac{\text{parent item} \quad \textcolor{red}{\text{son item}} \quad \textcolor{magenta}{\text{trans}}}{\text{parent or son extension}} \qquad \{\text{fitting son and parent items}\}$$

Case [SPUSH]: parent item down-extends son item

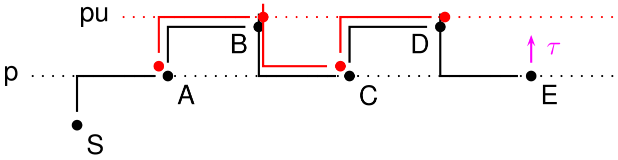


Dynamic Programming – Application rules

Based on following model:

$$\frac{\text{parent item} \quad \text{son item} \quad \text{trans}}{\text{parent or son extension}} \qquad \{\text{fitting son and parent items}\}$$

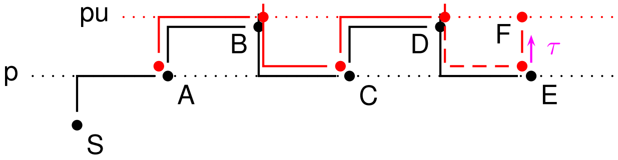
Case [SPUSH]: parent item down-extends son item



Based on following model:

$$\frac{\text{parent item} \quad \text{son item} \quad \text{trans}}{\text{parent or son extension}} \quad \{\text{fitting son and parent items}\}$$

Case [SPUSH]: parent item down-extends son item

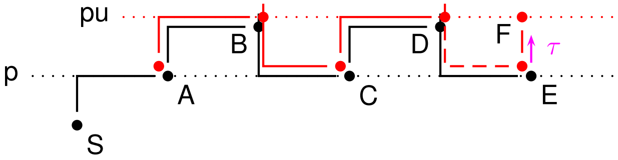


Dynamic Programming – Application rules

Based on following model:

$$\frac{\text{parent item} \quad \text{son item} \quad \text{trans}}{\text{parent or son extension}} \qquad \{\text{fitting son and parent items}\}$$

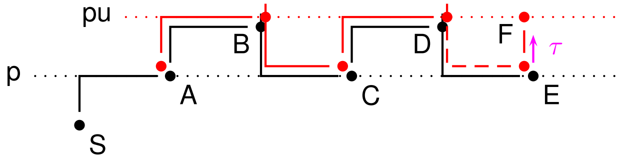
Case [SPUSH]: parent item down-extends son item



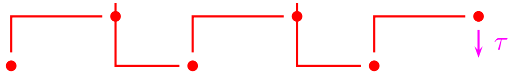
Based on following model:

$$\frac{\text{parent item} \quad \text{son item} \quad \text{trans}}{\text{parent or son extension}} \qquad \{\text{fitting son and parent items}\}$$

Case [SPUSH]: parent item down-extends son item



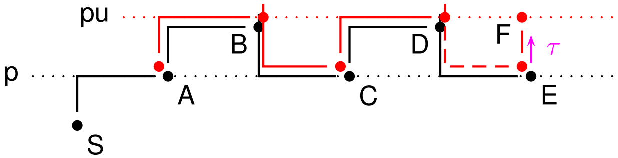
Case [SPOP]: son item up-extends parent item



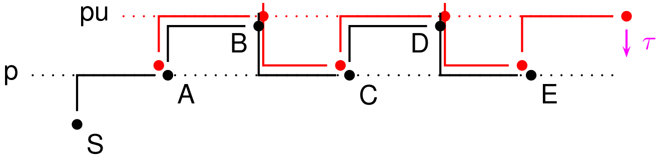
Based on following model:

$$\frac{\text{parent item} \quad \text{son item} \quad \text{trans}}{\text{parent or son extension}} \quad \{\text{fitting son and parent items}\}$$

Case [SPUSH]: parent item down-extends son item



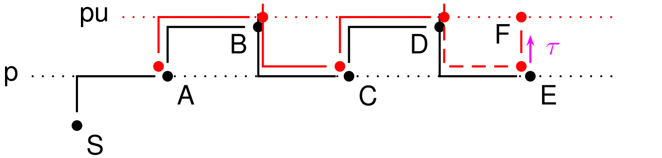
Case [SPOP]: son item up-extends parent item



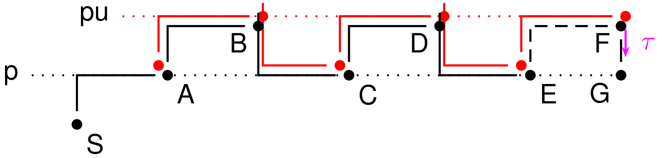
Based on following model:

$$\frac{\text{parent item} \quad \text{son item} \quad \text{trans}}{\text{parent or son extension}} \qquad \{\text{fitting son and parent items}\}$$

Case [SPUSH]: parent item down-extends son item



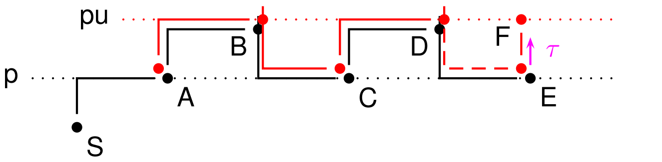
Case [SPOP]: son item up-extends parent item



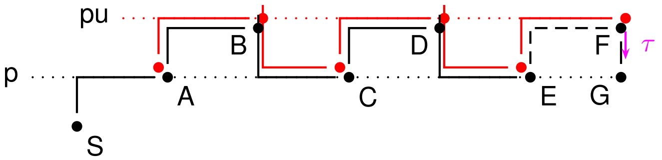
Based on following model:

$$\frac{\text{parent item} \quad \text{son item} \quad \text{trans}}{\text{parent or son extension}} \qquad \{\text{fitting son and parent items}\}$$

Case [SPUSH]: parent item down-extends son item



Case [SPOP]: son item up-extends parent item

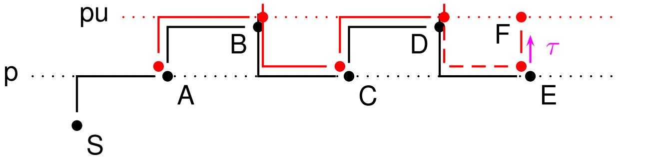


Dynamic Programming – Application rules

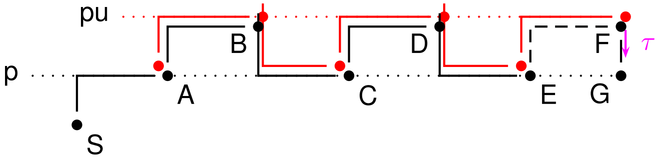
Based on following model:

$$\frac{\text{parent item} \quad \text{son item} \quad \text{trans}}{\text{parent or son extension}} \qquad \{\text{fitting son and parent items}\}$$

Case [SPUSH]: parent item down-extends son item



Case [SPOP]: son item up-extends parent item



Time complexity: all indices of parent item + end position of son item
ignore indices of son item not related to parent suspensions

Dynamic Programming: Rules

$$\frac{B \overset{\alpha}{\longmapsto} C \quad \langle a\rangle/\mathcal{S}/\langle B\rangle}{\langle a\rangle/\mathcal{S}/\langle C\rangle} \qquad a_r = \alpha \text{ if } \alpha \neq \epsilon \qquad \text{(SWAP)}$$

$$\frac{b \longmapsto [b]C \quad \star/\star/\langle B\rangle^I}{\langle b\rangle//\langle C\rangle} \qquad \{\ (b,u) \in \Phi\delta(B) \wedge u \notin \mathrm{ind}(I) \qquad \text{(PUSH)}$$

$$\frac{[B]C \longmapsto D \quad \langle a\rangle/\mathcal{S}/\langle B\rangle^I \quad J}{\langle a\rangle/\mathcal{S}_{/u}/\langle D\rangle} \qquad \begin{cases} J \nearrow^{u} I \wedge (b,u) \in \Phi\delta(B) \\ J^\bullet = \langle C\rangle \wedge \mathrm{ind}(J) \subset \{\perp\} \end{cases} \qquad \text{(POP)}$$

$$\frac{b[C] \longmapsto [b]D \quad I \quad \langle a\rangle/\mathcal{S}/\langle C\rangle^J}{\langle a\rangle/\mathcal{S},\perp:\langle c\rangle\langle b\rangle//\langle D\rangle} \qquad \begin{cases} I \searrow_{u} J \wedge I^\bullet = \langle B\rangle \\ (b,u) \in \Phi\delta(B) \wedge (c,\perp) \in \Phi\delta(C) \end{cases} \qquad \text{(SPUSH)}$$

$$\frac{[B]c \longmapsto D[c] \quad \langle a\rangle/\mathcal{S}/\langle B\rangle^I \quad J}{\langle a\rangle/\mathcal{S},u:\langle b\rangle\langle c\rangle//\langle D\rangle} \qquad \begin{cases} J \nearrow^{u} I \wedge (b,u) \in \Phi\delta(B) \\ J^\bullet = \langle C\rangle \wedge (c,\perp) \in \Phi\delta(C) \end{cases} \qquad \text{(SPOP)}$$