Parsing TAGs and beyond

Éric de la Clergerie
Eric.De_La_Clergerie@inria.fr

http://alpage.inria.fr

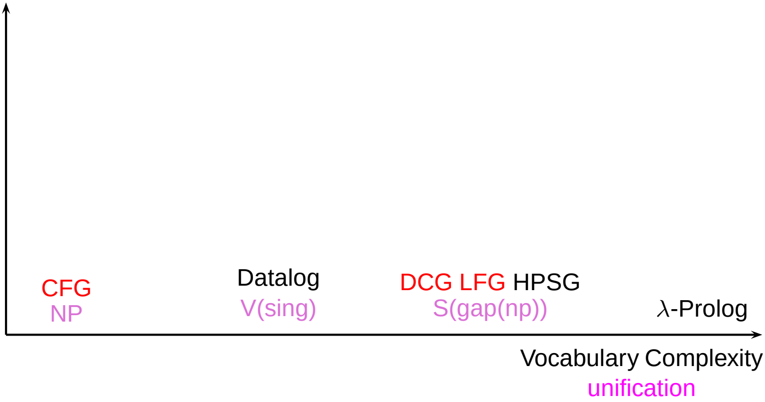
Tutorial TAG+9
Tuebingen, June 6th 2008

Why parsing TAGs ?

- TAGs are fun and linguistically important (**TAG**)
- TAGs are complex to parse, but they open the way for many variants and even more complex formalisms (**TAG+**)

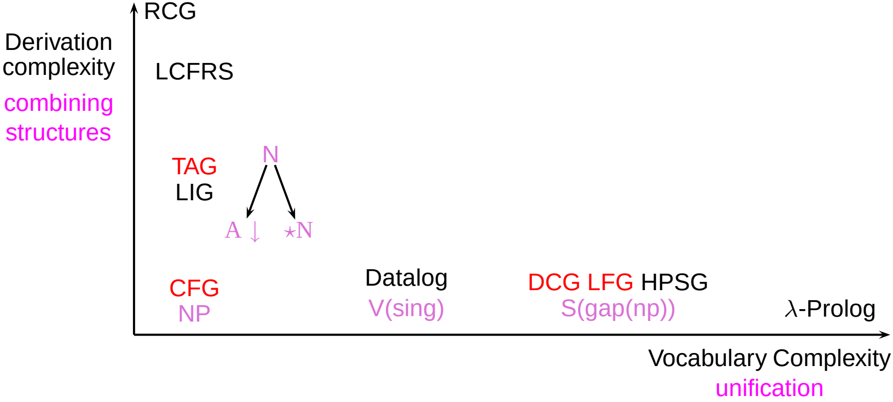
Why parsing TAGs ?

- TAGs are fun and linguistically important (**TAG**)
- TAGs are complex to parse, but they open the way for many variants and even more complex formalisms (**TAG+**)



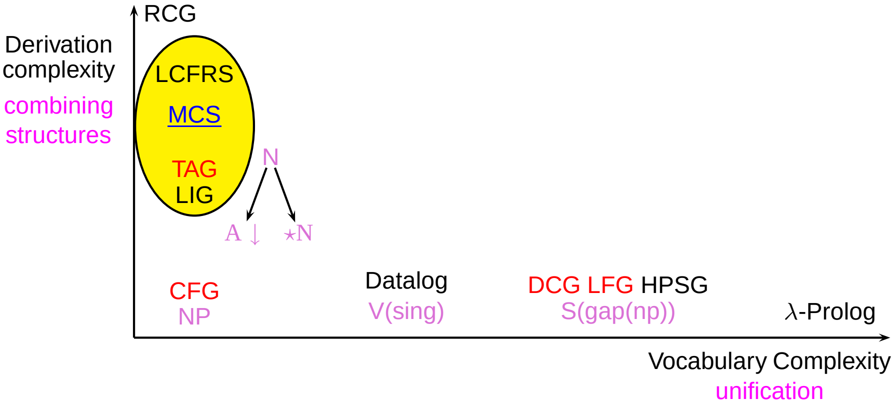
Why parsing TAGs ?

- TAGs are fun and linguistically important (**TAG**)
- TAGs are complex to parse, but they open the way for many variants and even more complex formalisms (**TAG+**)



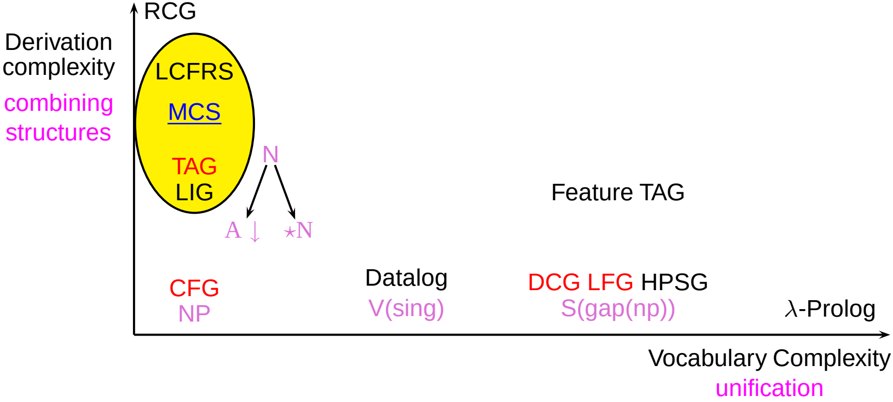
Why parsing TAGs ?

- TAGs are fun and linguistically important (**TAG**)
- TAGs are complex to parse, but they open the way for many variants and even more complex formalisms (**TAG+**)



Why parsing TAGs ?

- TAGs are fun and linguistically important (**TAG**)
- TAGs are complex to parse, but they open the way for many variants and even more complex formalisms (**TAG+**)



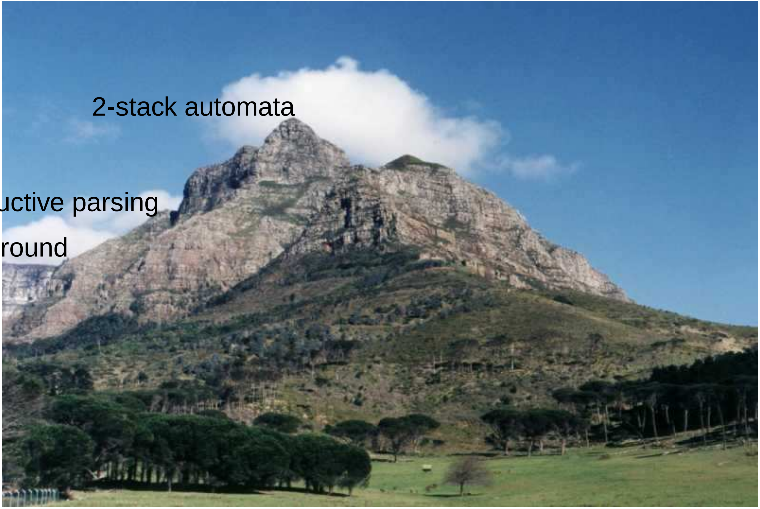
Background

Roadmap: climbing the devil's peak

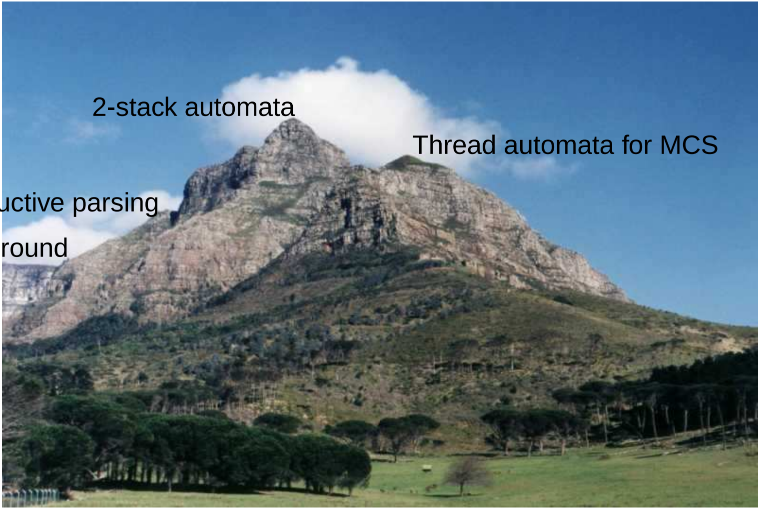
Deductive parsing

Background

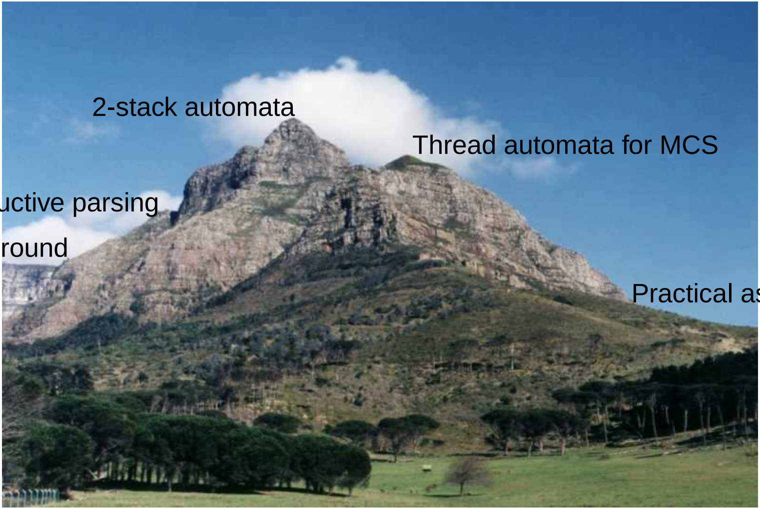
Roadmap: climbing the devil's peak



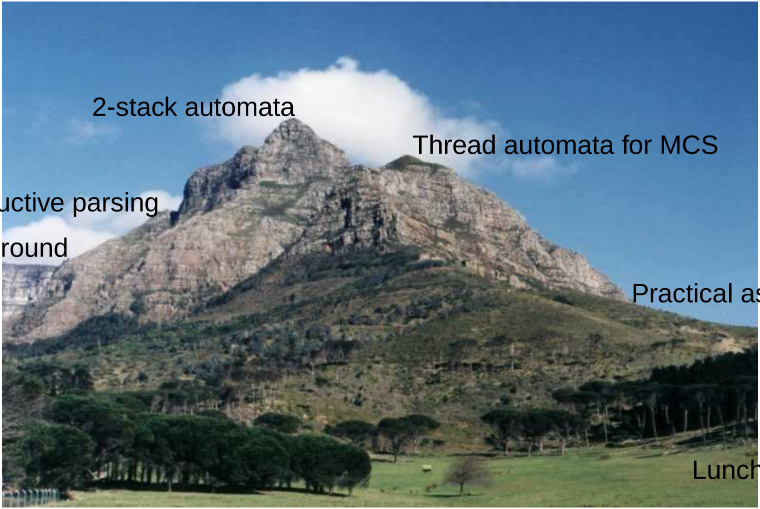
Roadmap: climbing the devil's peak



Roadmap: climbing the devil's peak



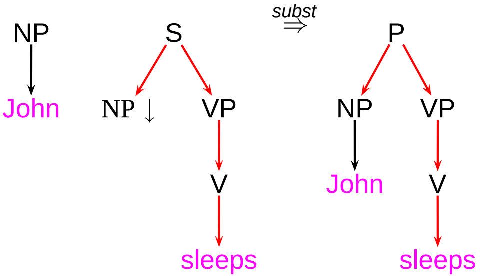
Roadmap: climbing the devil's peak



Outline

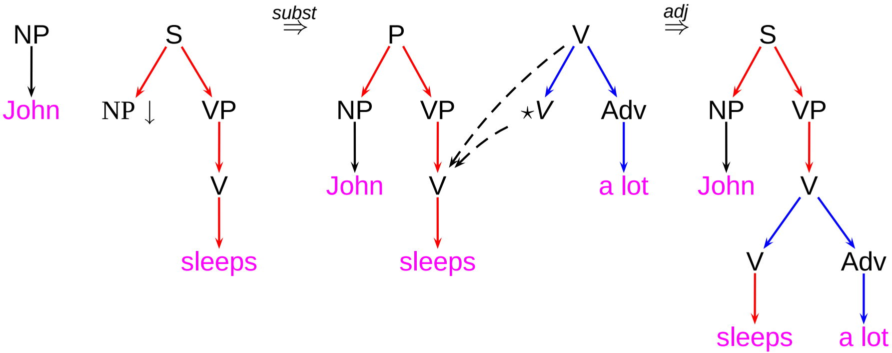
TAG: a small example

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

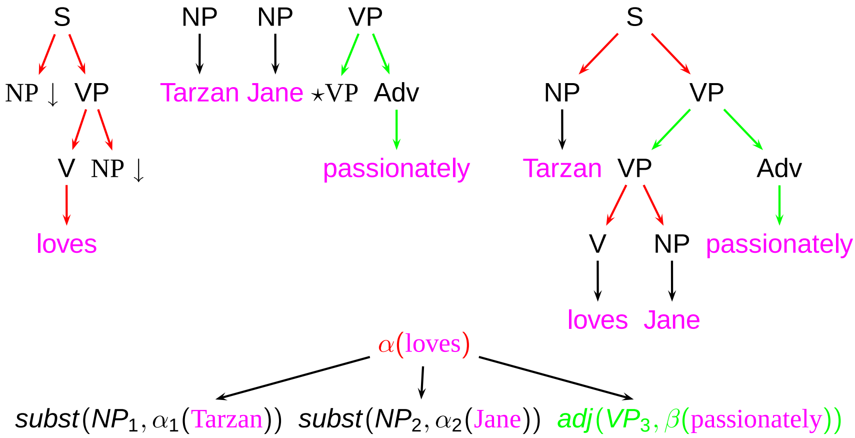


TAG: a small example

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



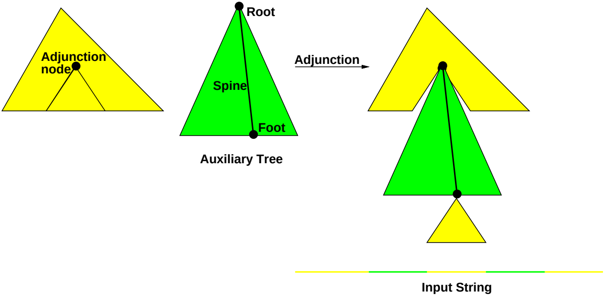
Derivation tree



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

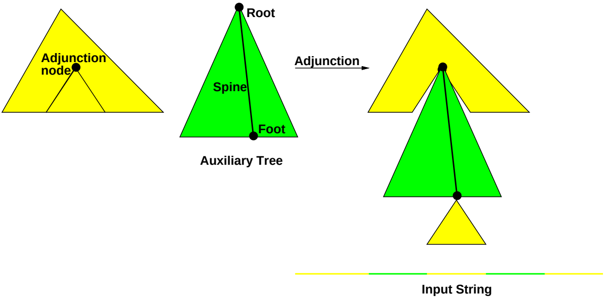
TAG complexity: Adjoining

TAG complexity: Adjoining



adjoining

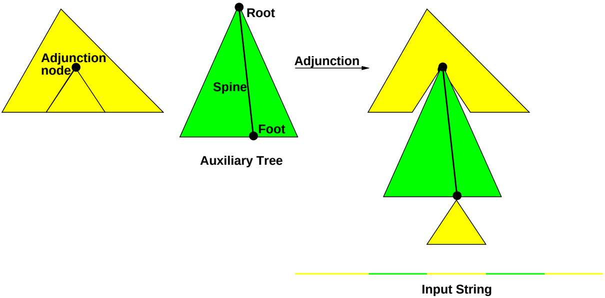
TAG complexity: Adjoining



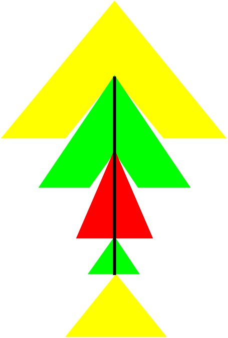
adjoining

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

TAG complexity: Adjoining



adjoining



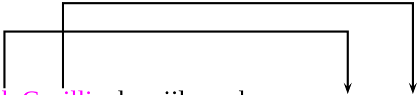
nested adjoining

- 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

Outline

Deductive parsing

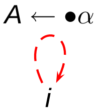
Formalization of chart parsing

Use of

- universe of tabulable **items**, representing (set of) partial parses
- items often build upon **dotted rules** $A_0 \leftarrow A_1 \ldots A_i \bullet A_{i+1} \ldots A_n$
- 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

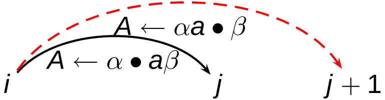
CKY as a deductive system (for CFGs)

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



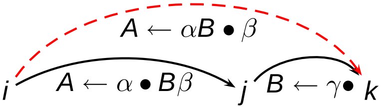
(Seed)

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



(Scan)

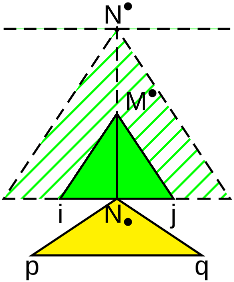
$$\frac{\langle i, j, A \leftarrow \alpha \bullet B\beta \rangle \quad \langle j, k, B \leftarrow \gamma\bullet \rangle}{\langle i, k, A \leftarrow \alpha B \bullet \beta \rangle}$$



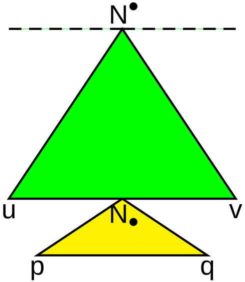
(Complete)

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.



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

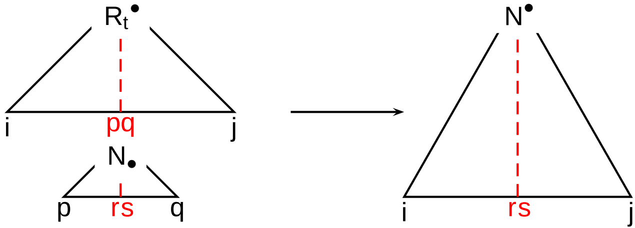


Without adjoining: $\langle N_\bullet, p, -, -, q\rangle$
With adjoining: $\langle N^\bullet, u, -, -, v\rangle$

Rule (Adjoin)

Gluing a sub-tree at a foot node.

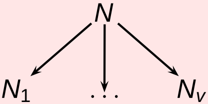
$$\frac{\langle \mathrm{N}_\bullet, p, r, s, q\rangle \; \langle \mathrm{R_t}^\bullet, i, p, q, j\rangle}{\langle \mathrm{N}^\bullet, i, r, s, j\rangle} \qquad \mathrm{label}(N) = \mathrm{label}(R_t) \qquad \text{(Adjoin)}$$



Rule (Complete)

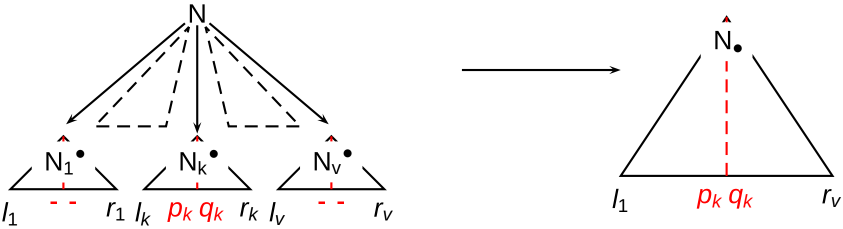
Gluing all node's children

$$\frac{\langle\, \mathrm{N_i}^{\bullet}, l_i, p_i, q_i, r_i \rangle}{\langle\, \mathrm{N}_{\bullet}, l_1, \cup p_i, \cup q_i, r_v \rangle}$$



and $l_{i+1} = r_i$ (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
3. node without adjoining

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 ($v = 2$) $\Rightarrow$ complexity $O(n^6)$

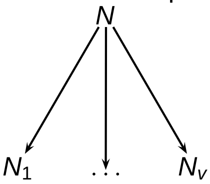
4 indexes per item $\Rightarrow$ Space complexity in $O(n^4)$ for a recognizer
$O(n^6)$ for a parser, keeping backpointers to parents

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

Prediction, dotted trees and dotted producctions

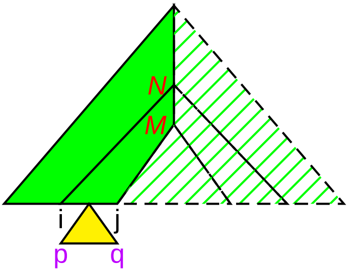
To mark prediction, new dotted trees [Shabes]: ${}^{\bullet}$N and ${}_{\bullet}$N

Alternative: equivalence with dotted productions



$\equiv N \leftarrow N_1 \dots N_v$

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



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

Non prefix valid Earley algorithm

- Glue a sub-tree at foot node F_t (not necessarily correct)

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

- Advance in recognition of N's children

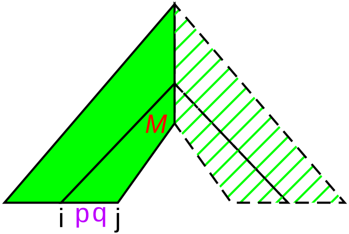
$$\frac{\langle N \leftarrow \alpha \bullet M\beta, i, u, v, j\rangle \;\; \langle M \leftarrow \gamma\bullet, j, r, s, k\rangle}{\langle N \leftarrow \alpha M \bullet \beta, i, u \cup r, v \cup s, k\rangle} \qquad \text{(Complete)}$$

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

Adjoining Prediction

Predict adjoining at M

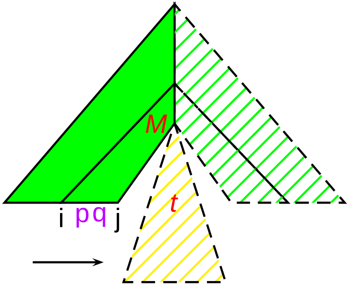
$$\frac{\langle N \leftarrow \alpha \bullet M\beta, i, p, q, j\rangle}{} \qquad \text{label}(M) = \text{label}(R_t) \qquad \text{(CallAdj)}$$



Adjoining Prediction

Predict adjoining at M

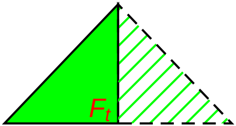
$$\frac{\langle N \leftarrow \alpha \bullet M\beta, i, p, q, j\rangle}{\langle \top \leftarrow \bullet R_t, j, -, -, j\rangle} \qquad \text{label}(M) = \text{label}(R_t) \qquad \text{(CallAdj)}$$



Foot Prediction

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

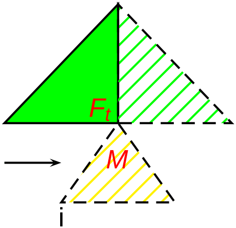
$$\frac{\langle F_t \leftarrow \bullet \bot, i, -, -, i \rangle}{} \qquad \text{label}(F_t) = \text{label}(M) \qquad \text{(CallFoot)}$$



Foot Prediction

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

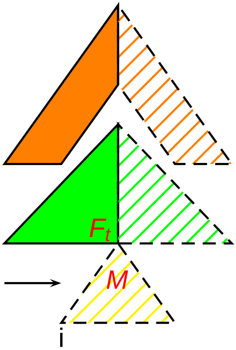
$$\frac{\langle F_t \leftarrow \bullet \perp, i, -, -, i \rangle}{\langle M \leftarrow \bullet \gamma, i, -, -, i \rangle} \qquad \text{label}(F_t) = \text{label}(M) \qquad \text{(CallFoot)}$$



Foot Prediction

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

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



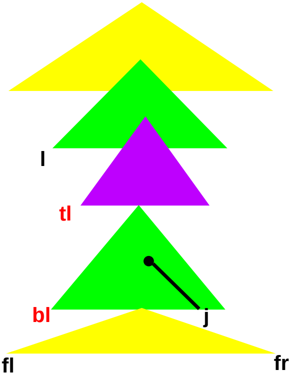
The prediction of M not related to the node M' having triggered the adjoining of t
$\Rightarrow$ Non prefix valid parsing strategy

Complexity

- Space complexity remains $O(n^4)$
- Dotted productions $\Rightarrow$ implicit binarization $\Rightarrow$ 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]

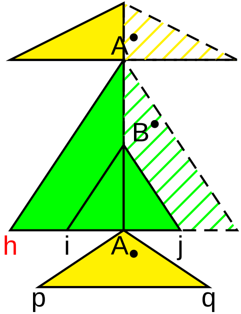
Complexities time in $O(n^9)$ and space in $O(n^6)$ due to 6-index items



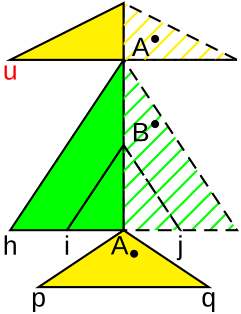
Actually, *tl* and *bl* may be avoided using dotted productions

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 on-going adjoining



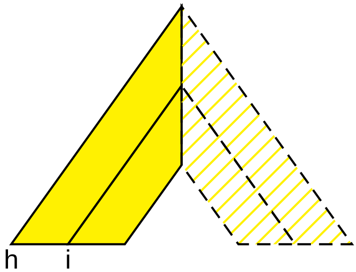
$\langle h, N \leftarrow \alpha B \bullet \beta, i, p, q, j \rangle$



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

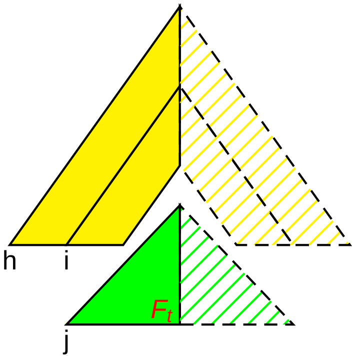
Foot prediction

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



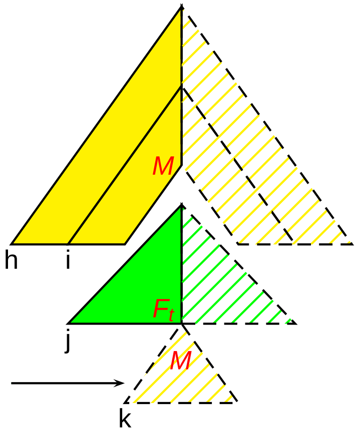
Foot prediction

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



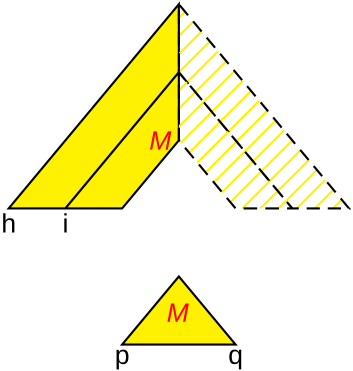
Foot prediction

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



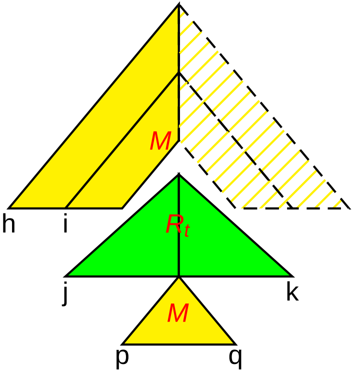
Adjoining return

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



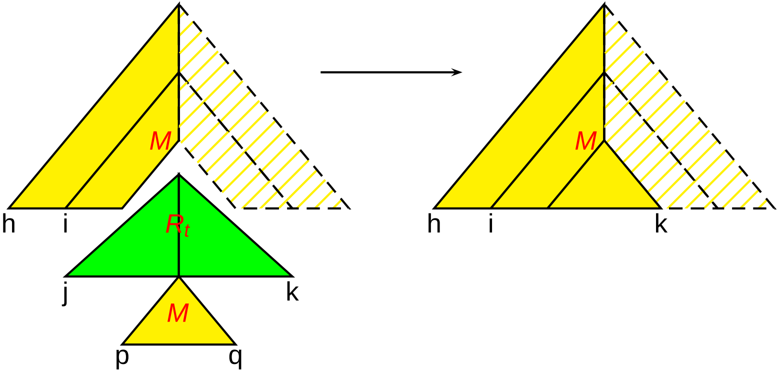
Adjoining return

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



Adjoining return

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



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 $(-, -)$
$\Rightarrow$ (Case analysis) splitting rule into 2 sub-rules $\Rightarrow O(n^8) \Rightarrow$ 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

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

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

Projection

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

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

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

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

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

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

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

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]?

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

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
$\Rightarrow$ need restrictions

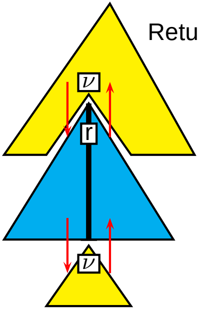
2-stack automata for TAGs

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) $\underline{\nu}_n = \bot$
$^\bullet$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

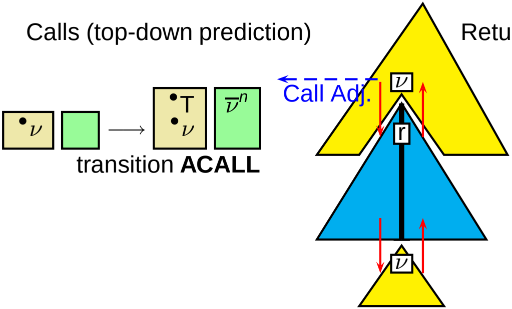
Auxiliary Stack: only to keep trace of uncompleted adjunctions

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)

Calls (top-down prediction)

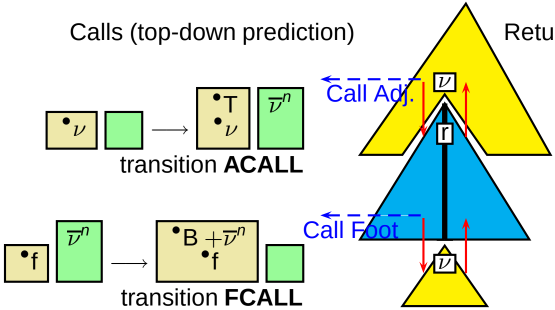
Returns (bottom-up propagation)



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



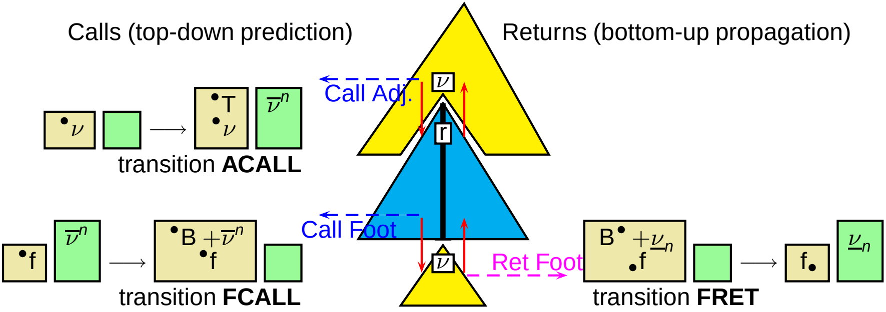
Solution: stack asymmetry

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) $\underline{\nu}_n = \bot$

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



2-stack automata for TAGs

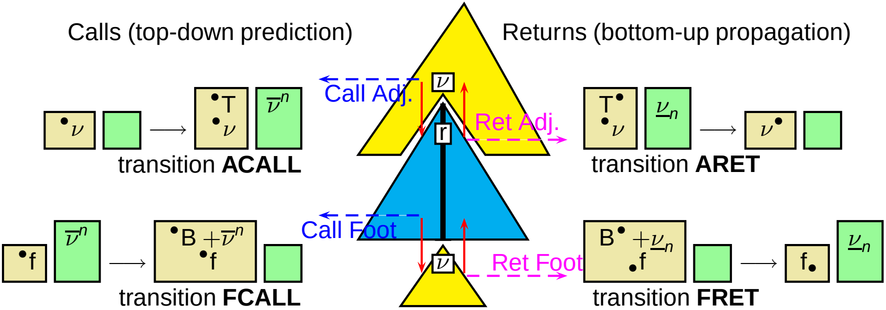
Solution: stack asymmetry

Master Stack: to keep trace of uncompleted tree traversals

Auxiliary Stack: only to keep trace of uncompleted adjunctions

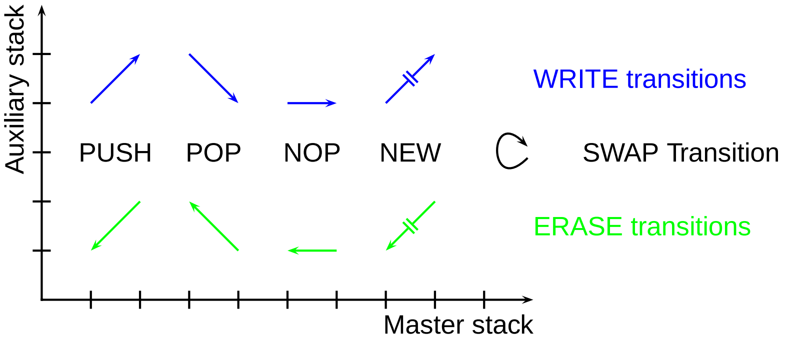
Adjunction info: (top-down) $\overline{\nu}^n = \nu$ and (bottom-up) $\underline{\nu}_n = \bot$

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



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.



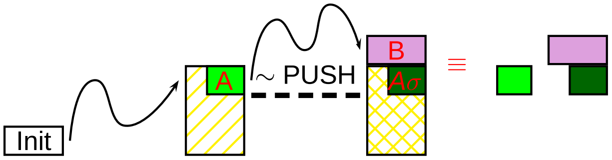
Context-Free derivation for PDAs

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

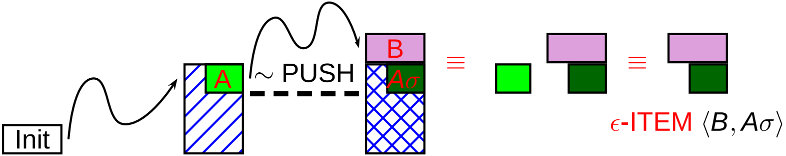
For PDAs, derivations broken into elementary Context-Free sub-derivations:



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

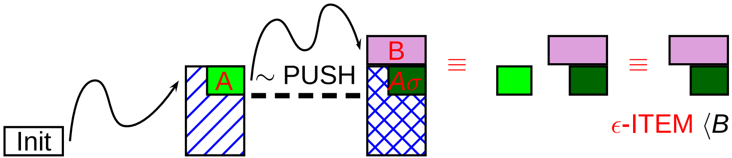
For PDAs, derivations broken into elementary Context-Free sub-derivations:



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:

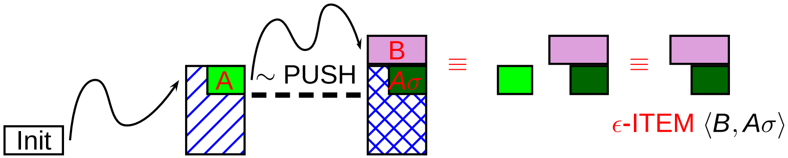


ϵ-ITEM $\langle B, A\sigma \rangle$

Context-Free derivation for PDAs

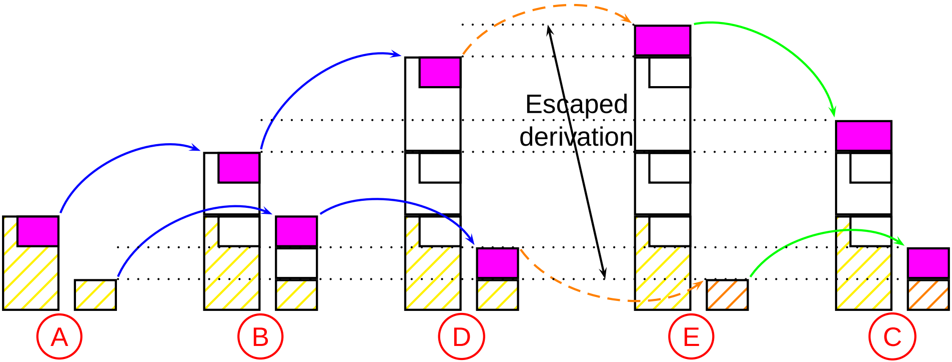
Dynamic Programming : Recursive decomposition of problems into elementary subproblems that may be combined, tabulated, and reused
eg the knapsack problem

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

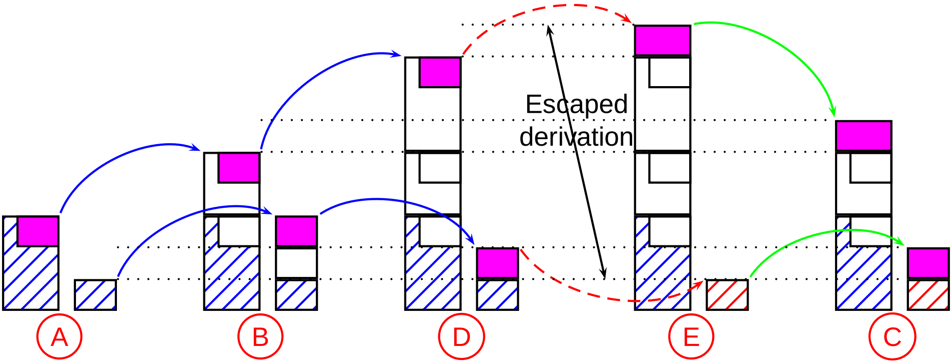


A is the fraction ϵ of information consulted to trigger the subderivation and not propagated to B.

(Escaped) CF derivations for 2SA



(Escaped) CF derivations for 2SA

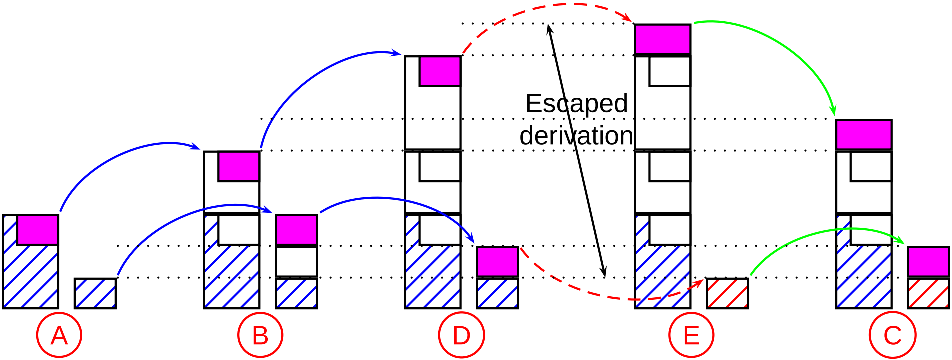


$\Rightarrow$ 5-point xCF items $AB[DE]C = \langle\epsilon A\rangle\langle\epsilon B, b\rangle[\langle\epsilon D, d\rangle\langle E\rangle]\langle C, c\rangle$
[TAG] $\rightsquigarrow \langle\epsilon A\rangle\langle\epsilon B\rangle[\langle\epsilon D\rangle\langle E\rangle]\langle C\rangle$

When no escaped part $\Rightarrow$ 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

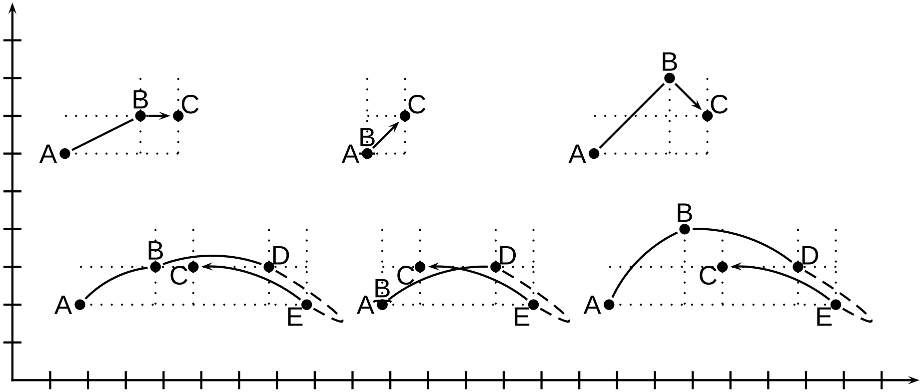
xCFs and TAGs



- *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 $\Rightarrow$ Space complexity in $O(n^5)$
SD-2SA restrictions & transition kinds $\Rightarrow$ 6 possible item shapes



Combining items and transitions

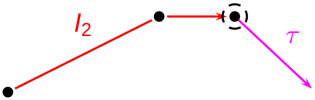
By graphically playing with items and transitions, we find 10 composition rules with $O(n^8)$ time complexity
may be split into 11 rules with $O(n^6)$ time complexity

(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

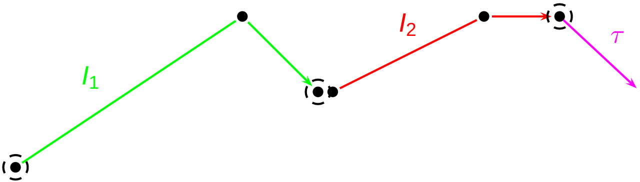
(Easy:) Write a POP mark: $I_1 + I_2 + \tau = I_3$



Combining items and transitions

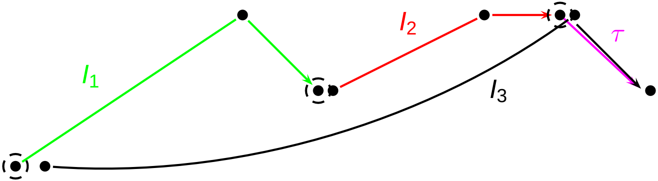
By graphically playing with items and transitions, we find 10 composition rules with $O(n^8)$ time complexity
may be split into 11 rules with $O(n^6)$ time complexity

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



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

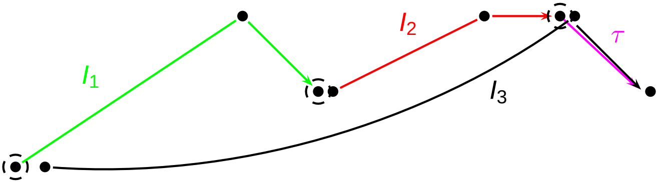
(Easy:) Write a POP mark: $I_1 + I_2 + \tau = I_3$



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: $I_1 + I_2 + \tau = I_3$



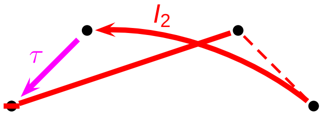
Consultation of 3 indexes ⦿ $\Rightarrow$ Complexity $O(n^3)$

Combining items and transitions (2)

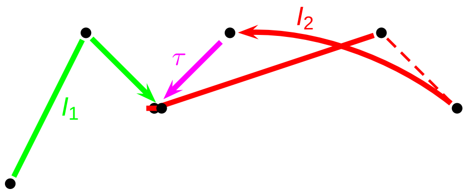
(complex:) Erasing a PUSH mark: $I_1 + I_2 + I_3 + \tau = I_4$
e.g. when returning from auxiliary tree (ending adjoining)

Combining items and transitions (2)

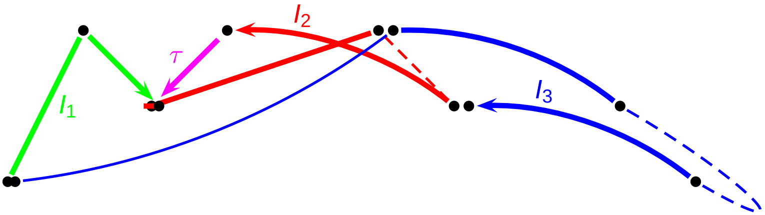
(complex:) Erasing a PUSH mark: $I_1 + I_2 + I_3 + \tau = I_4$
e.g. when returning from auxiliary tree (ending adjoining)



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

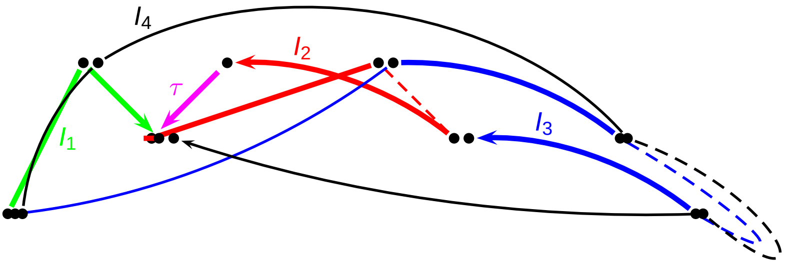


(complex:) Erasing a PUSH mark: $I_1 + I_2 + I_3 + \tau = I_4$
e.g. when returning from auxiliary tree (ending adjoining)



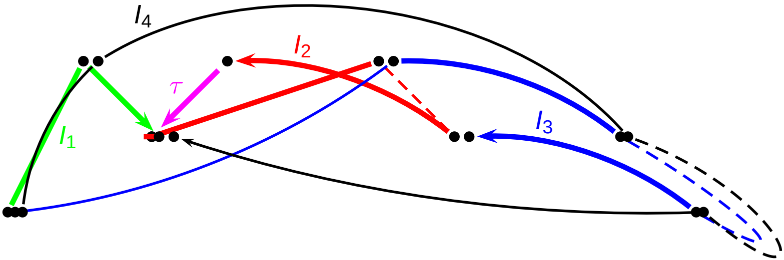
Combining items and transitions (2)

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



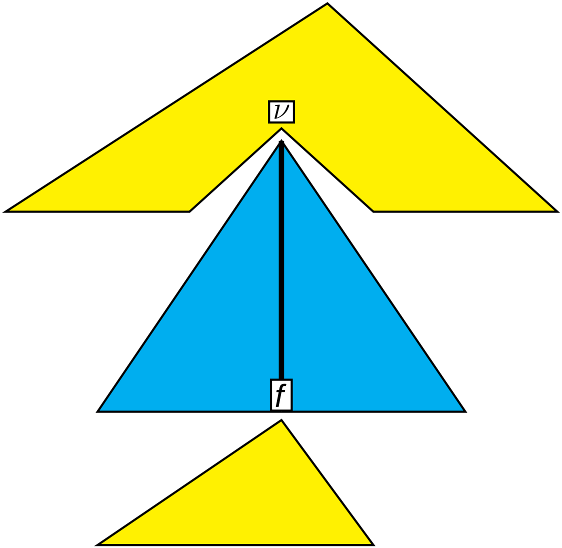
Combining items and transitions (2)

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

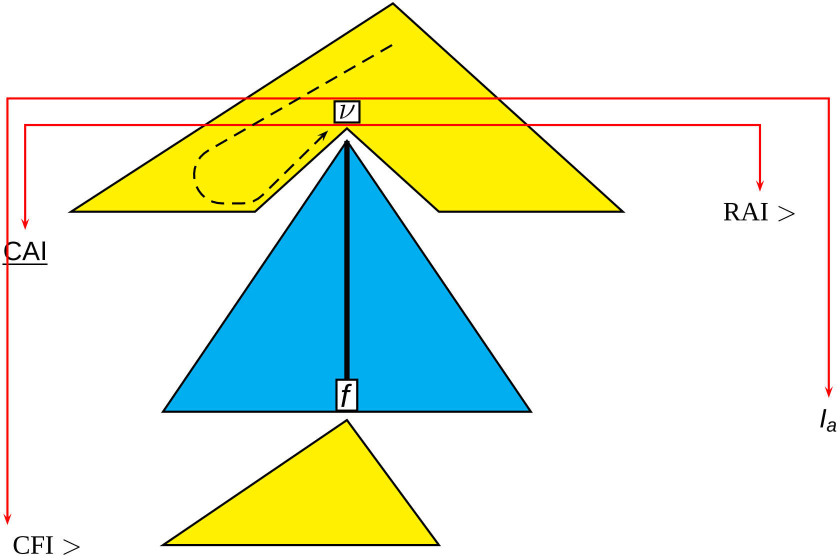


- Consultation of 8 indexes ⇒ Complexity $O(n^8)$
- need to decompose, project and use intermediary steps (as seen before)

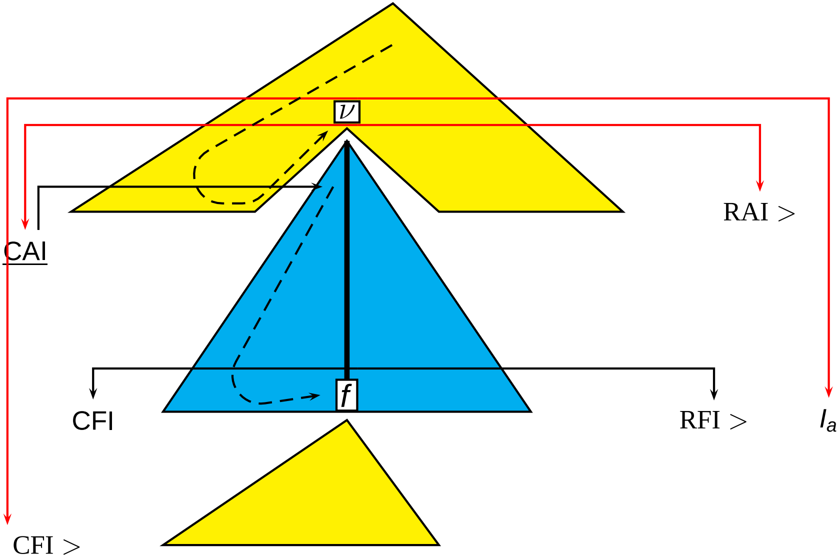
Cascade of partial evaluations [Devil's peak !]



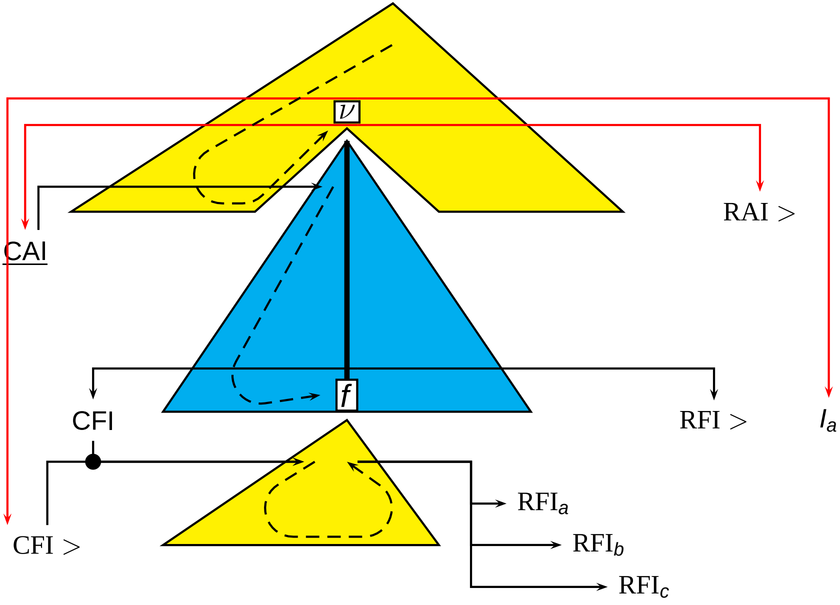
Cascade of partial evaluations [Devil's peak !]



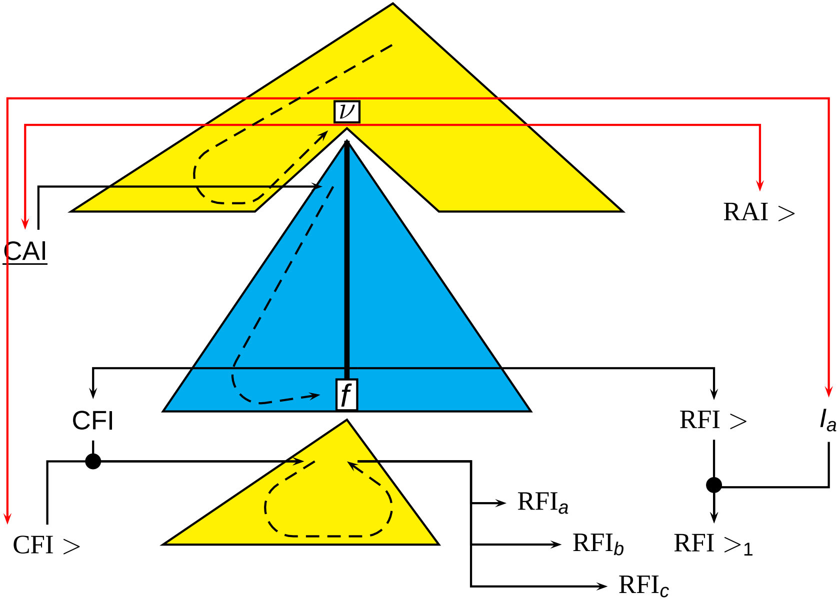
Cascade of partial evaluations [Devil's peak !]



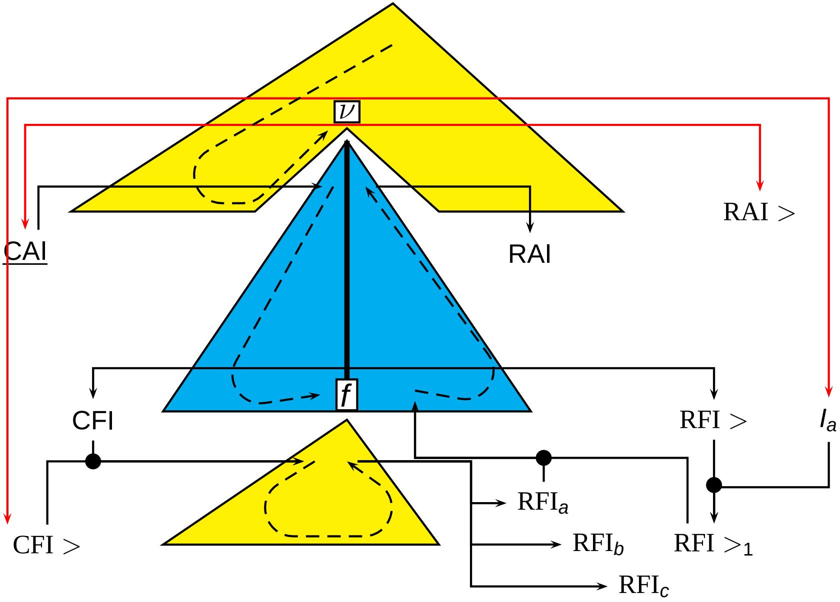
Cascade of partial evaluations [Devil's peak !]



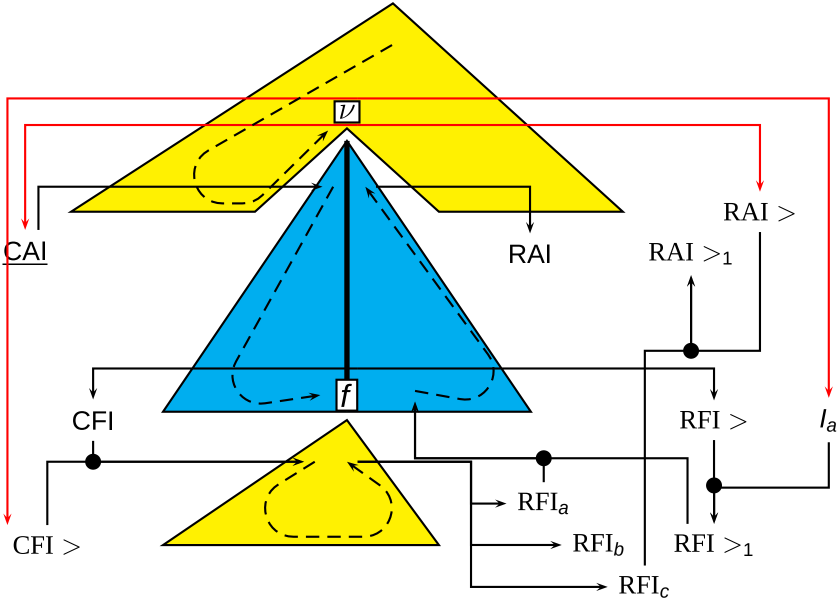
Cascade of partial evaluations [Devil's peak !]



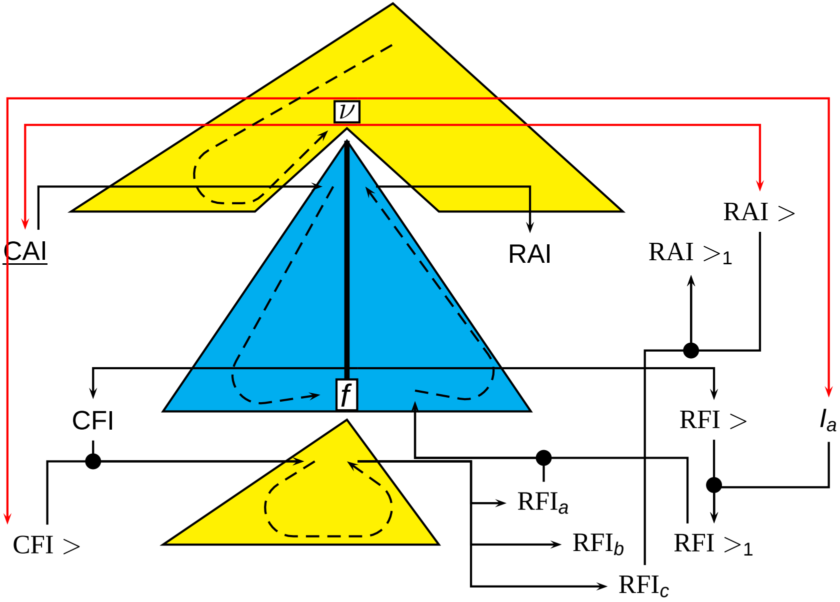
Cascade of partial evaluations [Devil's peak !]



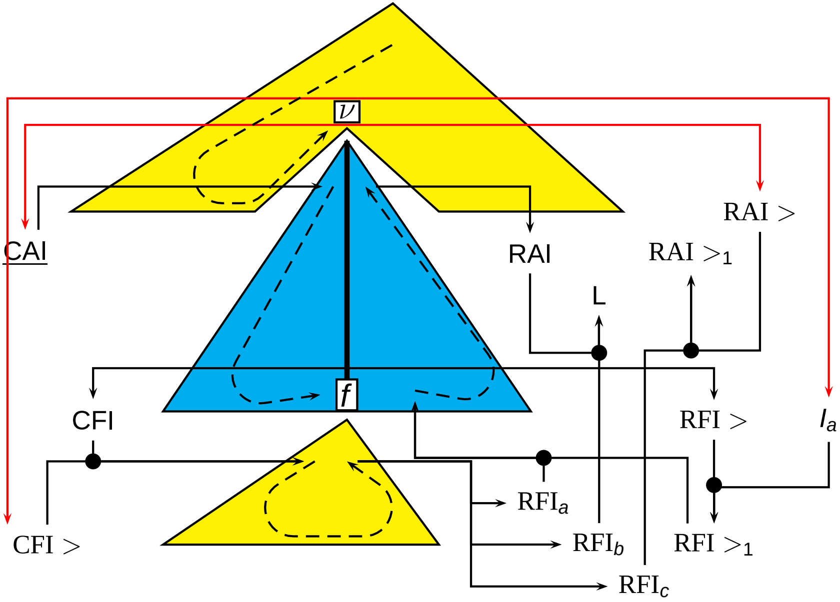
Cascade of partial evaluations [Devil's peak !]



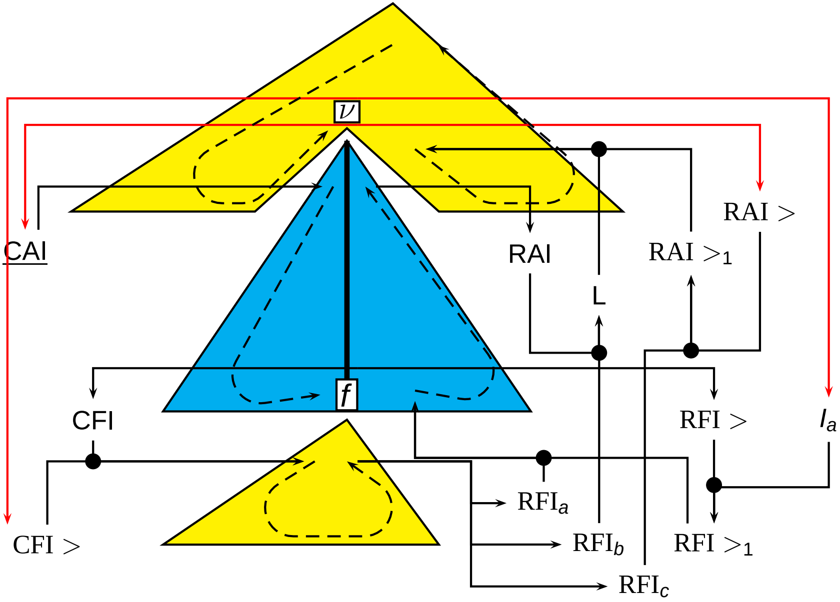
Cascade of partial evaluations [Devil's peak !]



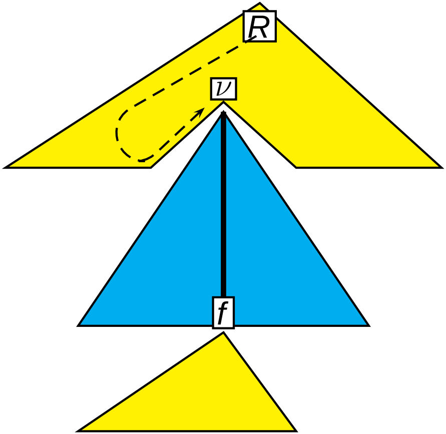
Cascade of partial evaluations [Devil's peak !]



Cascade of partial evaluations [Devil's peak !]

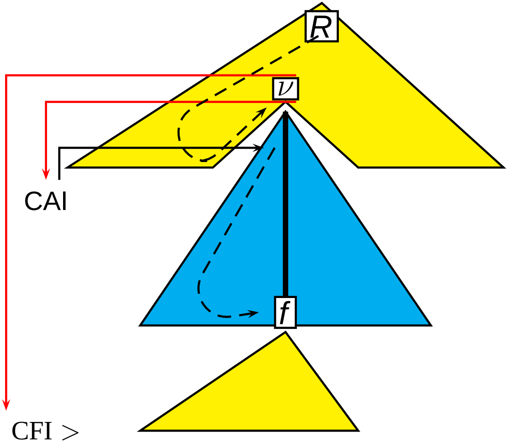


Simplified Cascade of partial evaluations



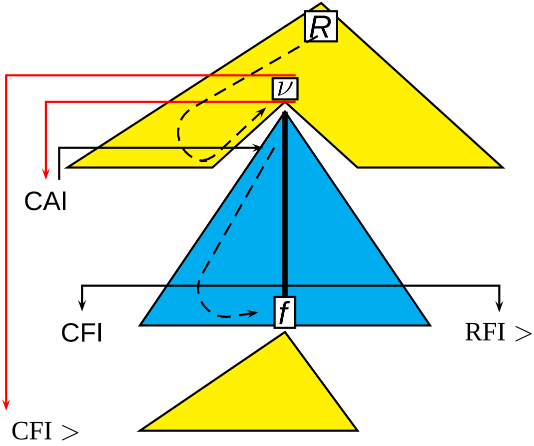
- Not the optimal worst case complexity (because yellow subtree traversed in the context of larger yellow subtree, keeping trace of unfinished adjoinings)
- But more efficient in practice !
- And suggesting extensions, based on the idea of continuation

Simplified Cascade of partial evaluations



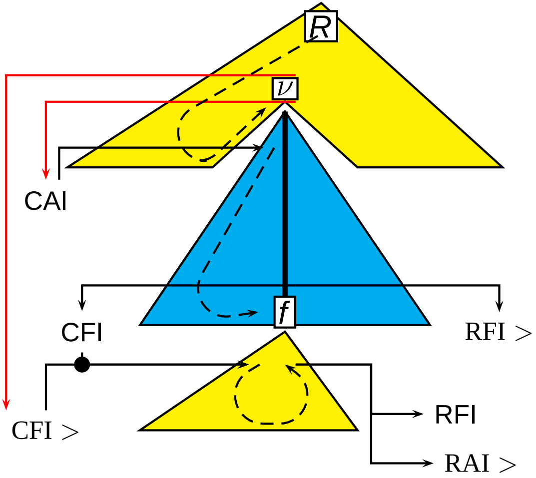
- Not the optimal worst case complexity (because yellow subtree traversed in the context of larger yellow subtree, keeping trace of unfinished adjoinings)
- But more efficient in practice !
- And suggesting extensions, based on the idea of continuation

Simplified Cascade of partial evaluations



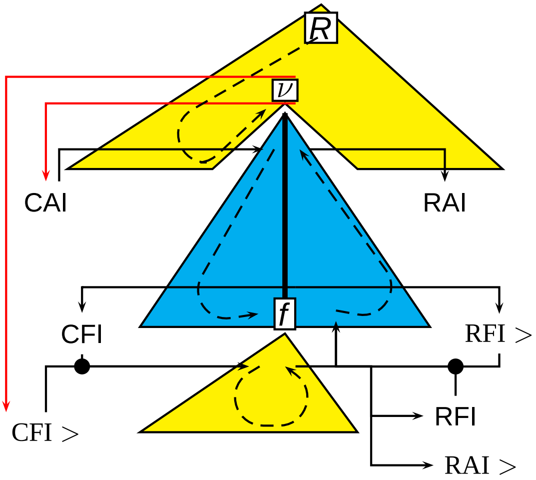
- Not the optimal worst case complexity (because yellow subtree traversed in the context of larger yellow subtree, keeping trace of unfinished adjoinings)
- But more efficient in practice !
- And suggesting extensions, based on the idea of continuation

Simplified Cascade of partial evaluations



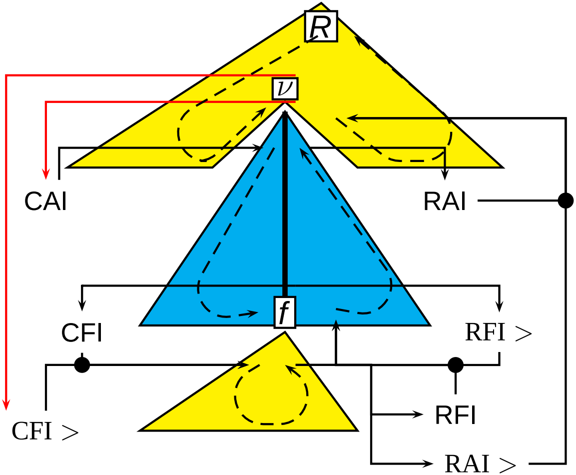
- Not the optimal worst case complexity (because yellow subtree traversed in the context of larger yellow subtree, keeping trace of unfinished adjoinings)
- But more efficient in practice !
- And suggesting extensions, based on the idea of continuation

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

Outline

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
- they generate languages satisfying the constant growth property

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

(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
- they generate languages satisfying the constant growth property

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

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

Some MCS languages:

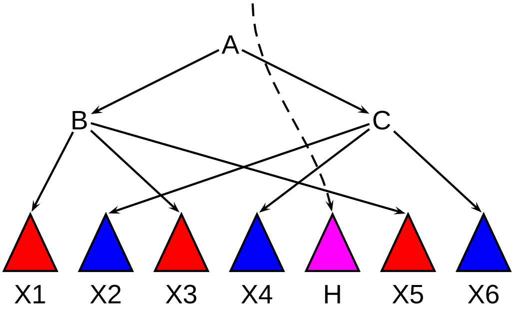
- TAGs and LIGs
- Local Multi Component TAGs (MC-TAGs Weir)
- Linear Context-Free Rewriting Systems (LCFRS Weir)
- Simple Range Concatenation Grammars (sRCG Boullier)

MCS: discontinuity and interleaving

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

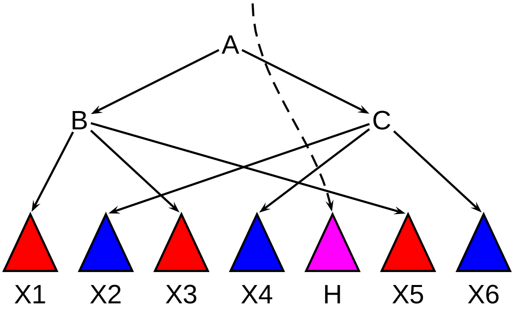
MCS: discontinuity and interleaving

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.

$$f(\langle x_1, x_3, x_5\rangle, \langle x_2, x_4, x_6\rangle) = \langle x_1x_2x_3x_4, x_5x_6\rangle$$

- **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 ","

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.

Thread Automata

Idea: Associate a thread p per constituent and

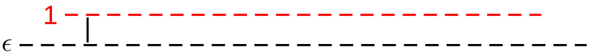
- create a subthread $p.u$ for a sub-constituent [PUSH]
- suspend thread at constituent discontinuity,
  and (resume) either the parent thread [SPOP]
  or some direct subthread [SPUSH]
- scan terminal [SWAP]
- delete thread after full recognition of a constituent [POP]

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

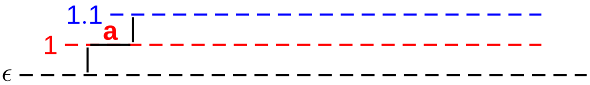


Thread Automata

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

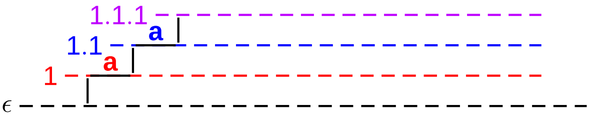


Thread Automata

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

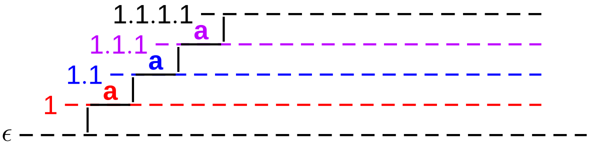


Thread Automata

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]

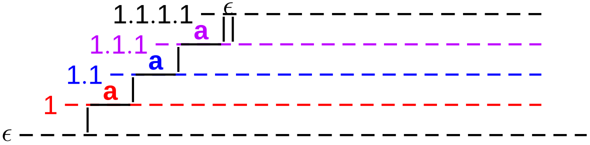
Recognize aaabbbccc $\in a^n b^n c^n$



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

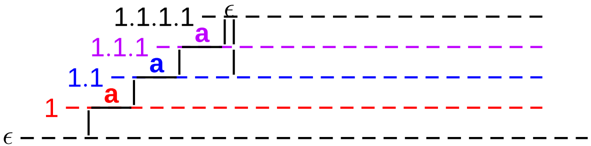


Thread Automata

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]

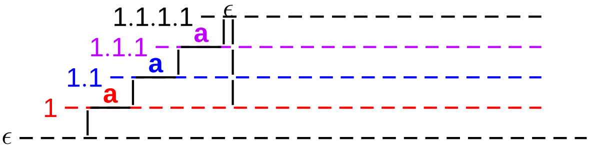
Recognize aaabbbccc $\in a^n b^n c^n$



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]

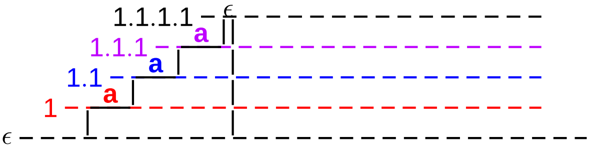
Recognize aaabbbccc $\in a^n b^n c^n$



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

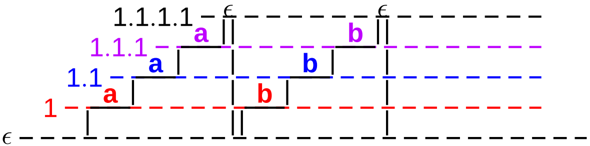


Thread Automata

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]

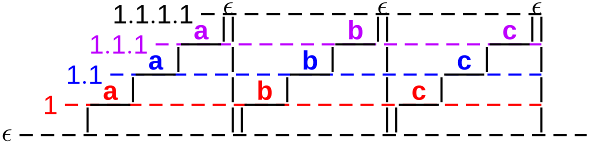
Recognize aaabbbccc $\in a^n b^n c^n$



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



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: $p.u \in \mathrm{dom}(\mathcal{S}) \Rightarrow p \in \mathrm{dom}(\mathcal{S})$
Note: stateless automata (but no problem for variants with states)

Triggering function $a = \kappa(A)$ Capture the amount of information needed to trigger transitions.
$\Rightarrow$ useful to get linear compexity $O(|G|)$ w.r.t. grammar size $|G|$

Driver function $u \in \delta(A)$ Drive thread creations and suspensions
$\Rightarrow$ 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.

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

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

$$\langle l, p, \mathcal{S} \cup p{:}B \rangle \mathrel{\underset{\mathcal{T}}{|\!\!-\!\!-}} \langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}C \rangle \quad (b, u) \in \kappa\delta(B) \wedge pu \not\in \mathrm{dor}$$

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

$$\langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}C \rangle \mathrel{\underset{\mathcal{T}}{|\!\!-\!\!-}} \langle l, p, \mathcal{S} \cup p{:}D \rangle \qquad pu \not\in \mathrm{dom}(\mathcal{S})$$

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

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

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

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

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:

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

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

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:

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

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

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

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:

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

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

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

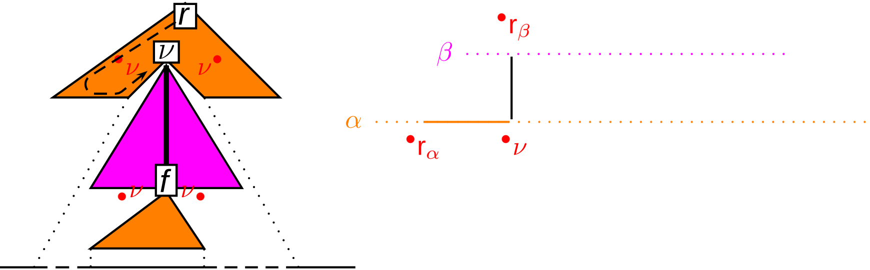
Complexity w.r.t. underlying grammar G: depends on the parsing strategy but generally possible to get $O(|G|)$, instead of $O(|G|^2)$

Parsing TAGs

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

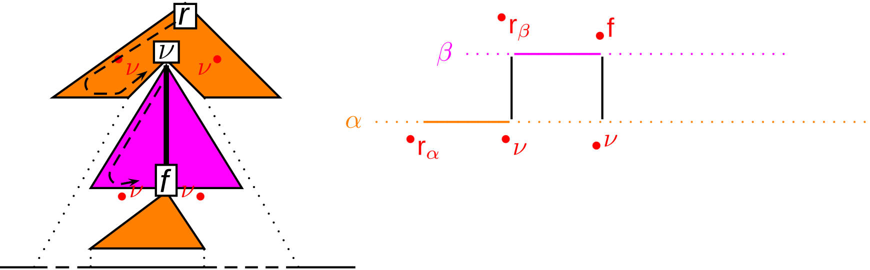
Parsing TAGs

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



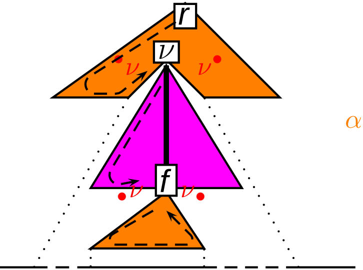
Parsing TAGs

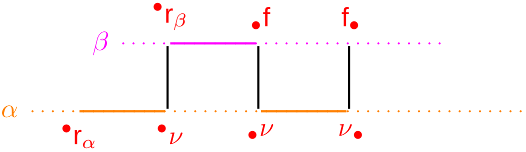
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node



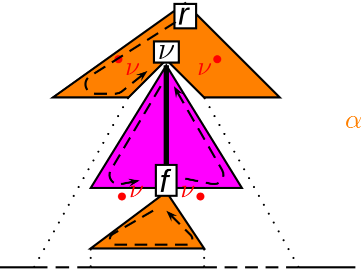
Parsing TAGs

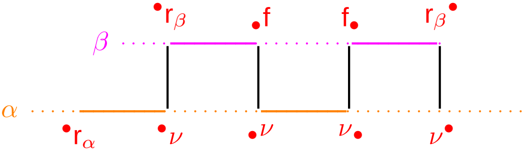
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node





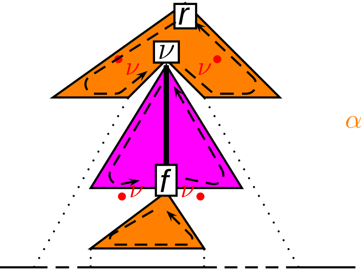
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node

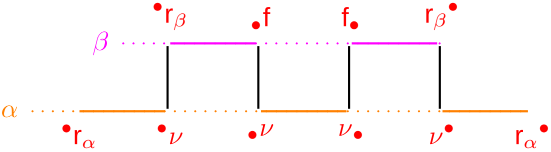




Parsing TAGs

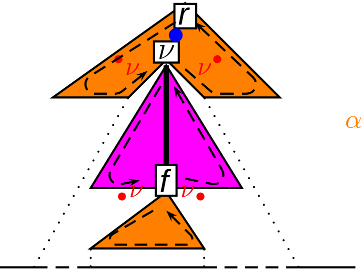
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node

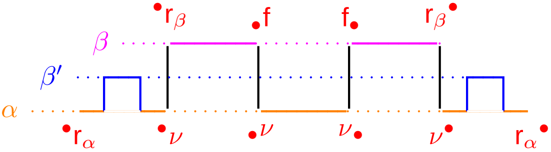




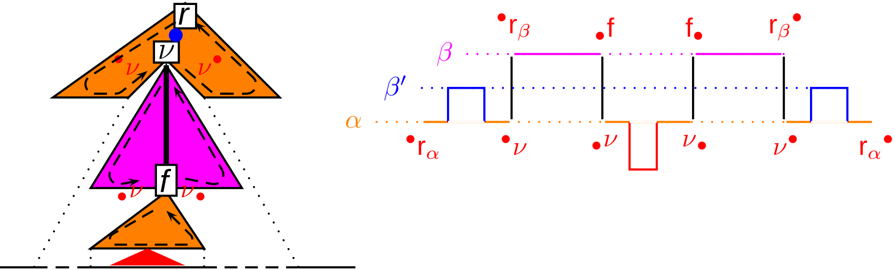
Parsing TAGs

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



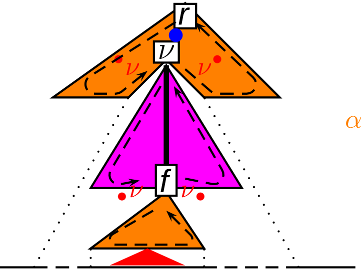


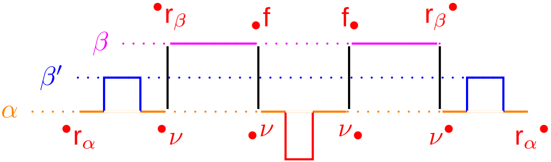
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node



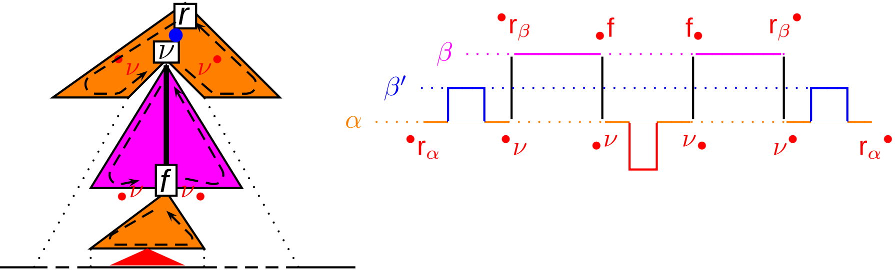
Parsing TAGs

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





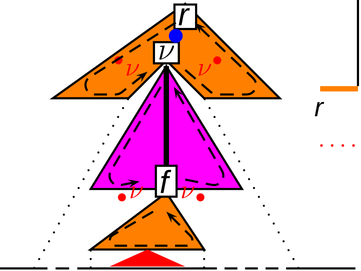
Idea: Assign a thread per elementary tree traversal (substitution or adjunction)
Suspend and return to parent thread to handle a foot node

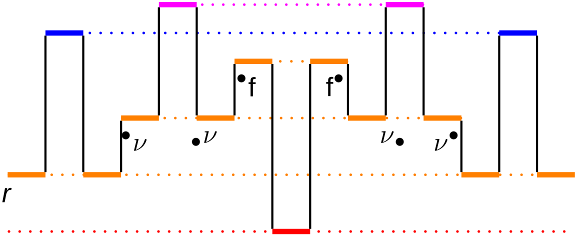


One thread per tree $h = 1, d = \max(\text{depth}(\text{trees}))$
$\Rightarrow$ $[s = 1 + d]$ space $O(n^{4+2d})$ and time $O(n^{5+2d})$

Parsing TAG: an alternate parsing strategy

Using more than one thread per elementary tree: 1 thread per subtree (∼ LIG)
⇒ implicit extraction of subtrees
⇒ implicit normal form (using a third kind of tree operation)
⇒ 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).

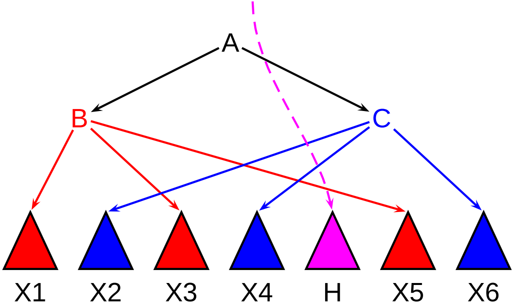
- if a thread p has $d + 1$ subthreads, add a new subthread $p.v$ that inherits d subthreads of p
- generally increases the number of parent suspensions h
- but may also exploit good topological properties, such as nesting (TAGs).

Parsing (ordered simple) RCG

Range Concatenation Grammars (Boullier)

$\gamma : A(X_1X_2X_3X_4, X_5X_6) \longrightarrow B(X_1, X_3, X_5)C(X_2, X_4, X_6)$

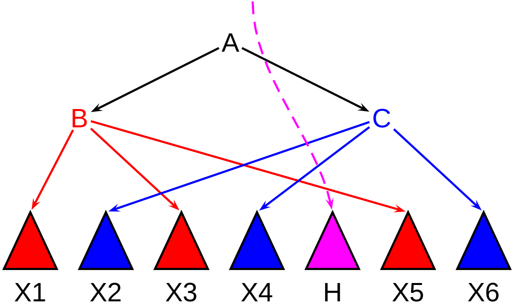
Ordered simple RCGs $\equiv$ Linear Context-Free Rewriting Systems (LCFRS)

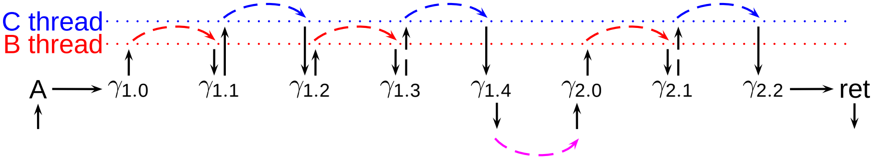


Range Concatenation Grammars (Boullier)

$\gamma: A(X_1X_2X_3X_4, X_5X_6) \longrightarrow B(X_1, X_3, X_5)C(X_2, X_4, X_6)$

Ordered simple RCGs ≡ Linear Context-Free Rewriting Systems (LCFRS)





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

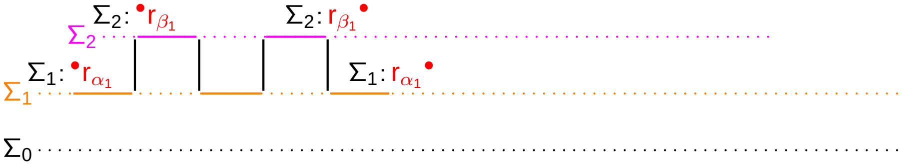
$$\begin{cases} \rho & \text{stack of dotted nodes of trees being traversed} \\ \sigma & \text{sequence of root nodes of trees already traversed} \end{cases}$$

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

- ρ stack of dotted nodes of trees being traversed
- σ sequence of root nodes of trees already traversed

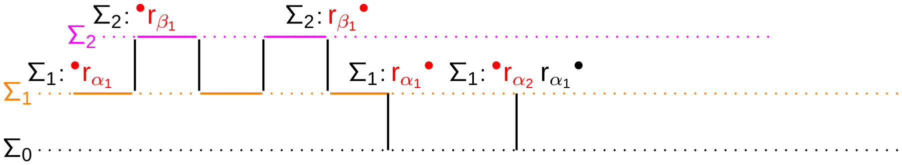
Eg.: Adjoin trees of set $\Sigma_2 = \{\beta_1, \beta_2\}$ on nodes of trees of set $\Sigma_1 = \{\alpha_1, \alpha_2\}$



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

- ρ stack of dotted nodes of trees being traversed
- σ sequence of root nodes of trees already traversed

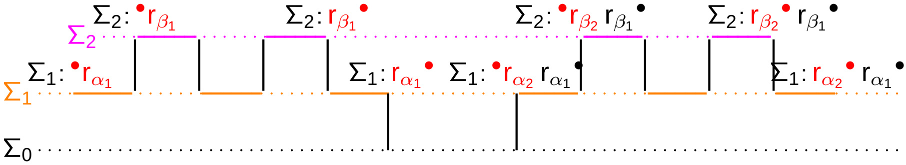
Eg.: Adjoin trees of set $\Sigma_2 = \{\beta_1, \beta_2\}$ on nodes of trees of set $\Sigma_1 = \{\alpha_1, \alpha_2\}$



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

- ρ stack of dotted nodes of trees being traversed
- σ sequence of root nodes of trees already traversed

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

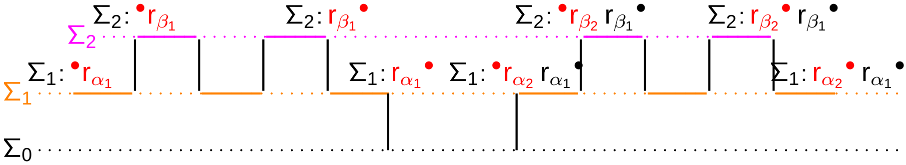


Parsing (set-local) MC-TAGs

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

- ρ stack of dotted nodes of trees being traversed
- σ sequence of root nodes of trees already traversed

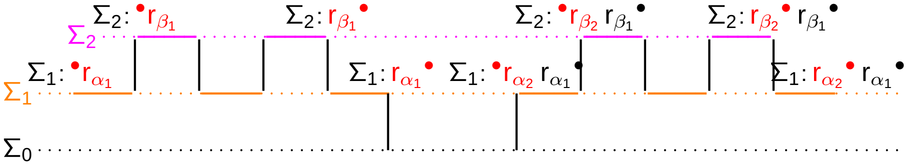
Eg.: Adjoin trees of set $\Sigma_2 = \{\beta_1, \beta_2\}$ on nodes of trees of set $\Sigma_1 = \{\alpha_1, \alpha_2\}$



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

- ρ stack of dotted nodes of trees being traversed
- σ sequence of root nodes of trees already traversed

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

- m max number of trees per set
- v max number of nodes per set

Outline

Dynamic Programming interpretation

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

Dynamic Programming interpretation

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

Use tabular techniques based on Dynamic Programming interpretation of TAs:

Dynamic Programming interpretation

Direct evaluation of TA $\rightsquigarrow$ 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

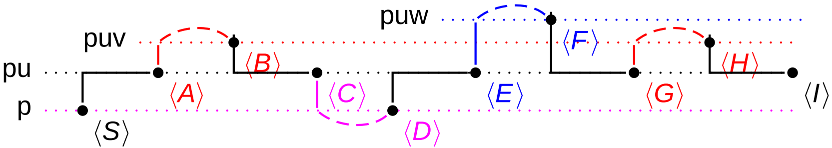
Start point	(current) Parent suspensions
(current) End point	(current) Subthread suspensions for **live** subthreads

Dynamic Programming – Items

DP interpretation of TA derivations:

(Tabulated) Item ≡ pertinent information about an (active) thread

Start point	(current) Parent suspensions
(current) End point	(current) Subthread suspensions for **live** subthreads

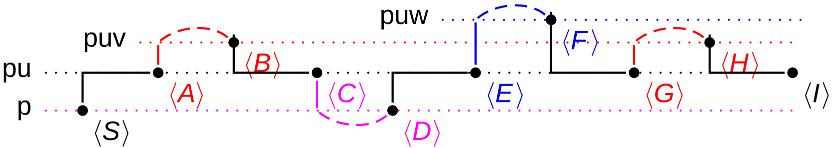


$\Rightarrow$ Item: $\langle s\rangle / v : \langle a\rangle\langle b\rangle, \bot : \langle c\rangle\langle d\rangle, w : \langle e\rangle\langle f\rangle, v : \langle g\rangle\langle h\rangle / \langle I\rangle$

Dynamic Programming – Items

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

Start point	(current) Parent suspensions
(current) End point	(current) Subthread suspensions for **live** subthreads



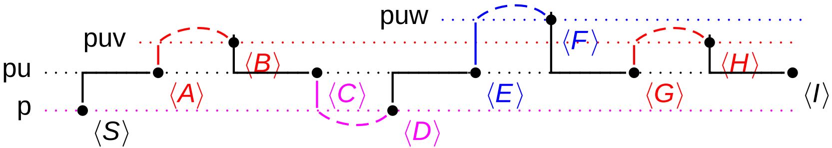
$\Rightarrow$ Item: $\langle s \rangle / v : \langle a \rangle \langle b \rangle, \perp : \langle c \rangle \langle d \rangle, w : \langle e \rangle \langle f \rangle, v : \langle g \rangle \langle h \rangle / \langle I \rangle$

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

Dynamic Programming – Items

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

Start point	(current) Parent suspensions
(current) End point	(current) Subthread suspensions for **live** subthreads



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

Projection $x = \kappa(X)$ used to trigger transition applications
$\Rightarrow$ 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)
  $\Rightarrow$ 1 index per suspension

Based on following model:

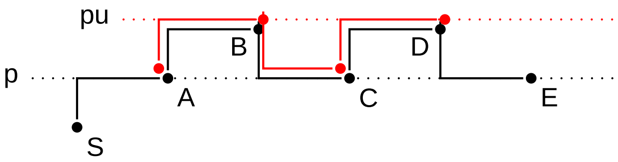
$$\frac{\text{parent item} \quad \text{son item} \quad \text{trans}}{\text{parent or son extension}} \qquad \{\text{fitting son and parent items}\}$$

Dynamic Programming – Application rules

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

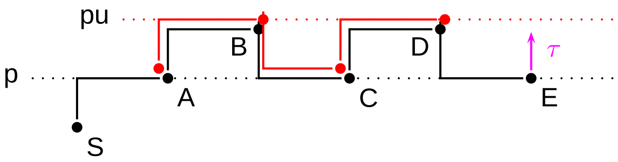


Dynamic Programming – Application rules

Based on following model:

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

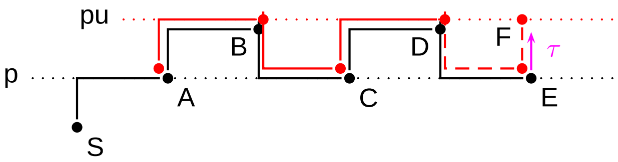
Case [SPUSH]: parent item down-extends son item



Based on following model:

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

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

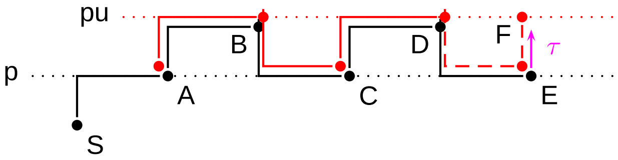


Dynamic Programming – Application rules

Based on following model:

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

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

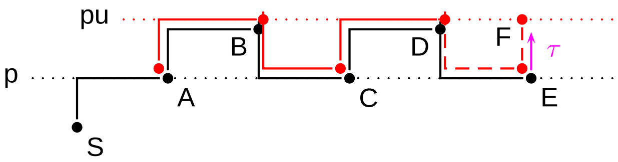


Dynamic Programming – Application rules

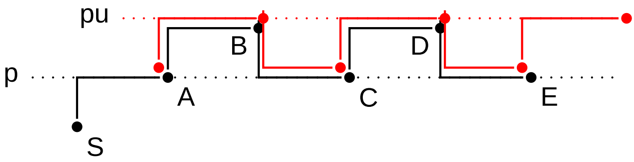
Based on following model:

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

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



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

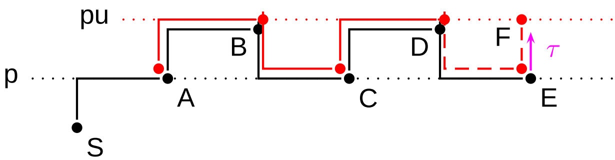


Dynamic Programming – Application rules

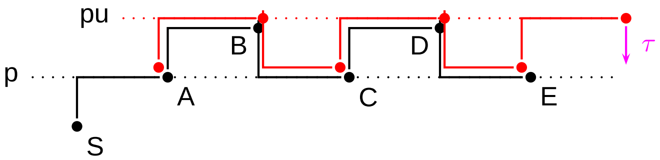
Based on following model:

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

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



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

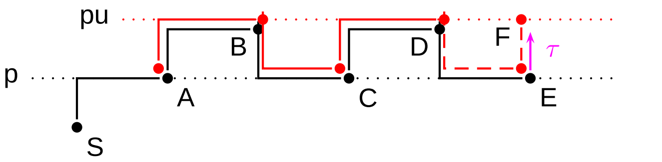


Dynamic Programming – Application rules

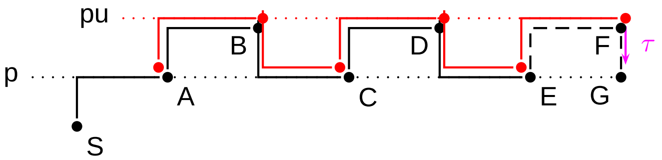
Based on following model:

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

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



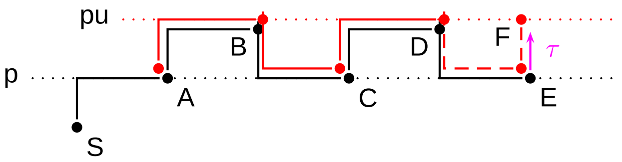
Case [SPOP]: son item up-extends parent item



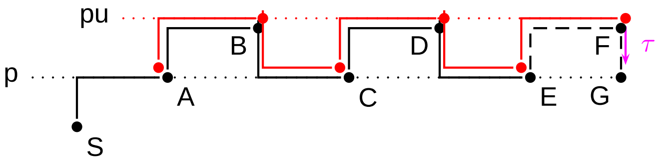
Based on following model:

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

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



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

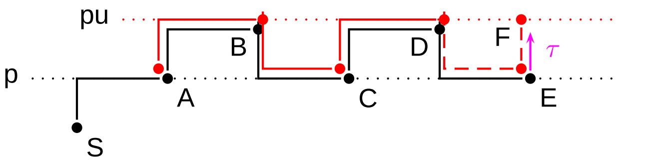


Based on following model:

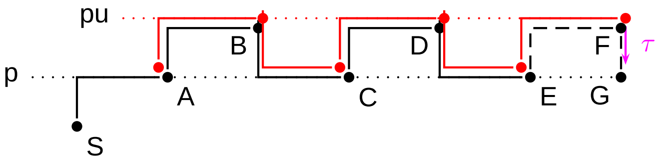
parent item son item trans
—
parent or son extension

{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

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

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

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

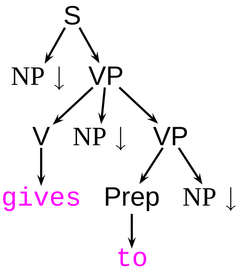
$$\frac{b[C] \longmapsto [b]D \quad I \quad \langle a \rangle / \mathcal{S} / \langle C \rangle^J}{\langle a \rangle / \mathcal{S}, \bot : \langle c \rangle \langle b \rangle / \langle D \rangle} \qquad \left\{ \begin{array}{l} I \searrow_{u} J \wedge I^{\bullet} = \langle B \rangle \\ (b,u) \in \kappa\delta(B) \wedge (c,\bot) \in \kappa\delta(C) \end{array} \right. \qquad \text{(SPUSH)}$$

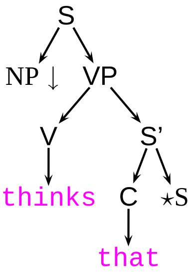
$$\frac{[B]c \longmapsto D[c] \quad \langle a \rangle / \mathcal{S} / \langle B \rangle^I \quad J}{\langle a \rangle / \mathcal{S}, u : \langle b \rangle \langle c \rangle / \langle D \rangle} \qquad \left\{ \begin{array}{l} J \nearrow^{u} I \wedge (b,u) \in \kappa\delta(B) \\ J^{\bullet} = \langle C \rangle \wedge (c,\bot) \in \kappa\delta(C) \end{array} \right. \qquad \text{(SPOP)}$$

Outline

Sub-Categorisation

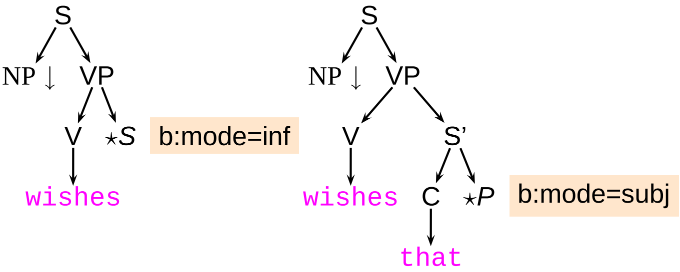
The extended domain of locality provided by trees allows (for instance) specifying the argument structure expected by a verb





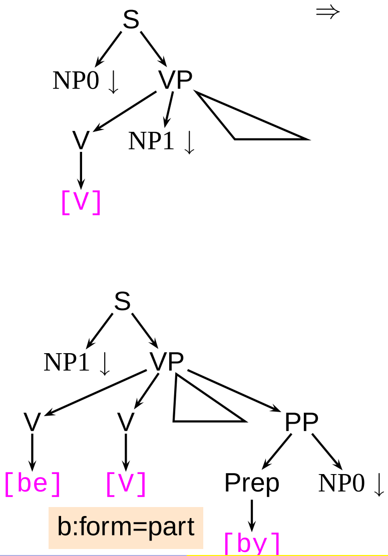
Feature TAGs

Node decoration `top` and `bot`:

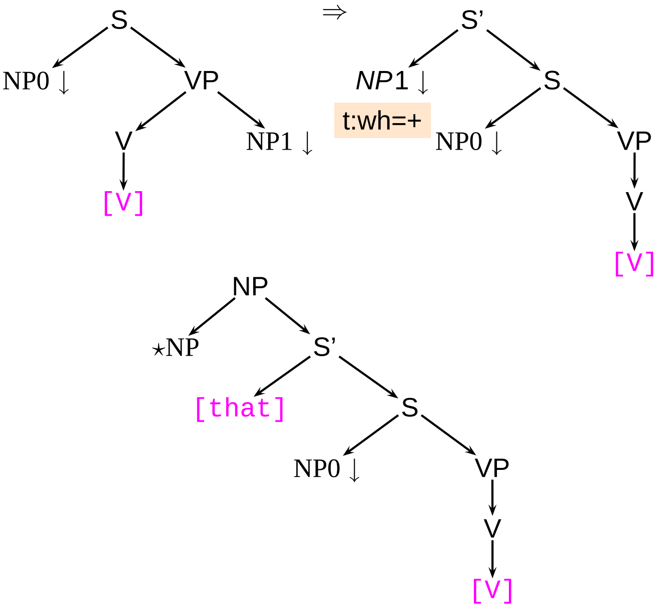


Family: passive

Transformation to handle passive voice



Family: Extraction



Real life problems

- Large grammar size, in terms of trees
  due to lexicalization and extended domain of locality
  $\Rightarrow$ several thousand tree schema, maybe more than ten thousands
  $\#\text{args}.\#\text{realizations}.\#\text{extractions}.\cdots$

- Complexity of adjoining

- Handling unification-based decorations (large feature structures)

Lexicalization

Lexicalized TAGs (LTAGs):

- each tree has to be anchored by a lexical (+ possibly lexical coanchors)
- the input words used to filter out non anchorable trees
- still many possible trees per words (specially for verbs)

Super- and Hyper- tagging

tagging words with information to anchor trees, depending on local contexts

Motivation: reducing the number of selected trees for a given word.

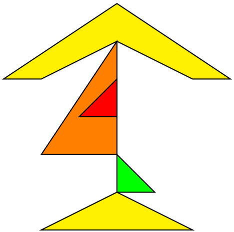
- **supertagging**: tree or family names
  but still many possible names per words (specially for verbs)

- **hypertagging**: (underspecified) feature structures, with feature characterizing syntactic properties (such as verb valence, diathesis, . . .)

promettre
$$\begin{bmatrix} \texttt{arg0} & \begin{bmatrix} \texttt{kind} & \text{subj} \mid \text{-} \\ \texttt{pcas} & \text{-} \end{bmatrix} \\ \texttt{arg1} & \begin{bmatrix} \texttt{kind} & \text{obj} \mid \text{scomp} \mid \text{-} \\ \texttt{pcas} & \text{-} \end{bmatrix} \\ \texttt{arg2} & \begin{bmatrix} \texttt{kind} & \text{prepobj} \mid \text{-} \\ \texttt{pcas} & \text{à} \mid \text{-} \end{bmatrix} \\ \texttt{refl} & \text{-} \end{bmatrix}$$

Hybrid TIG/TAG parsing

TIG are a TAG variant (Schabes) where one adjoining step can only insert material on left or right side of the adjoining node.



- Tree Insertion Grammars [TIG] have equivalent to CFGs (with $O(n^3)$ time complexity)
- Real life TAGs are mostly TIG and possible to automatically detect TIG and TAG parts of a grammar
  $\Rightarrow$ pay higher complexity only for wrapping adjoining
- May switch to multiple adjoining on nodes
  getting more natural derivation forests

Tree factorization

Idea : putting more in a single tree, because the trees share many common subparts

- defining more than one traversal path per tree (Harbush)
- using regular operators on trees:

disjunctions	$T[t_1;t_2] \equiv T[t_1] \cup T[t_2]$
repetitions	(Kleene Stars) $t@* \equiv \mathrm{kleene}_t(\epsilon) \cup \mathrm{kleene}_t(t, \mathrm{kleene}_t)$
interleaving	(free ordering between node sequences) $(t_1, t_2)\#\#t_3 \equiv (t_1, t_2, t_3; t_1, t_3, t_2; t_3, t_1, t_2)$
optionality	(optional node) $t? \equiv (t; \epsilon)$
guards	(guarded nodes) $T[G_+, t; G_-] \equiv T[t].\sigma_+ \cup T[\epsilon].\sigma_-$ guards: boolean formula over equation between feature structure paths

These operators

- do not modify expression power or complexity
- may be removed by expansion
  bur resulting trees exponential wrt number of operators
- more efficient to evaluate them without expansion
  $\Rightarrow$ more natural analysis
- very generic operators (not specific to TAGs, TIGs, or DCGs)

Handling node decorations

- good indexing mechanisms
- identifying related `top` ad **bottom** feature values that stay identical, even when adjoining
- efficient representations: for instance, bit vectors for finite sets of values (such as `mood` or `tense`)
- transformation into TAGs with no decoration but huge set of non-terminals but already dealing with large grammars !

Outline

French parser FRMG/DyALog: characteristics

French metagrammar FRMG with compiler generator DyALog;

- 2-SA based variant similar to non-optimal thread automata interpretation
- lexical filtering (even if the grammar is not 100% lexicalized)
- left-corner filtering
- hybrid TIG/TAG (almost 100%TIG), automatically detected by DyALog
- tree factoring (disjunction, Kleene stars, guards)
  thanks to generation from the meta-grammar
  $\Rightarrow$ 169 factorized trees
- (DyALog) bit vector for finite sets, table indexing, structure sharing
- no tagging, supertagging or hypertagging
  but planned when good training data for French
  and already using hypertags for tree anchoring
- returns shared derivation forests, converted into shared dependency forests

What is missing in this tutorial

- (n-best) shared derivation forests (Chiang & Huang)
- stochastic TIG/TAG parsing (Sarkar)
- machine learning techniques

Final words

I don't understand exactly what you are doing
but it doesn't look very useful

Romane, 8 years old