Test Suite Generation

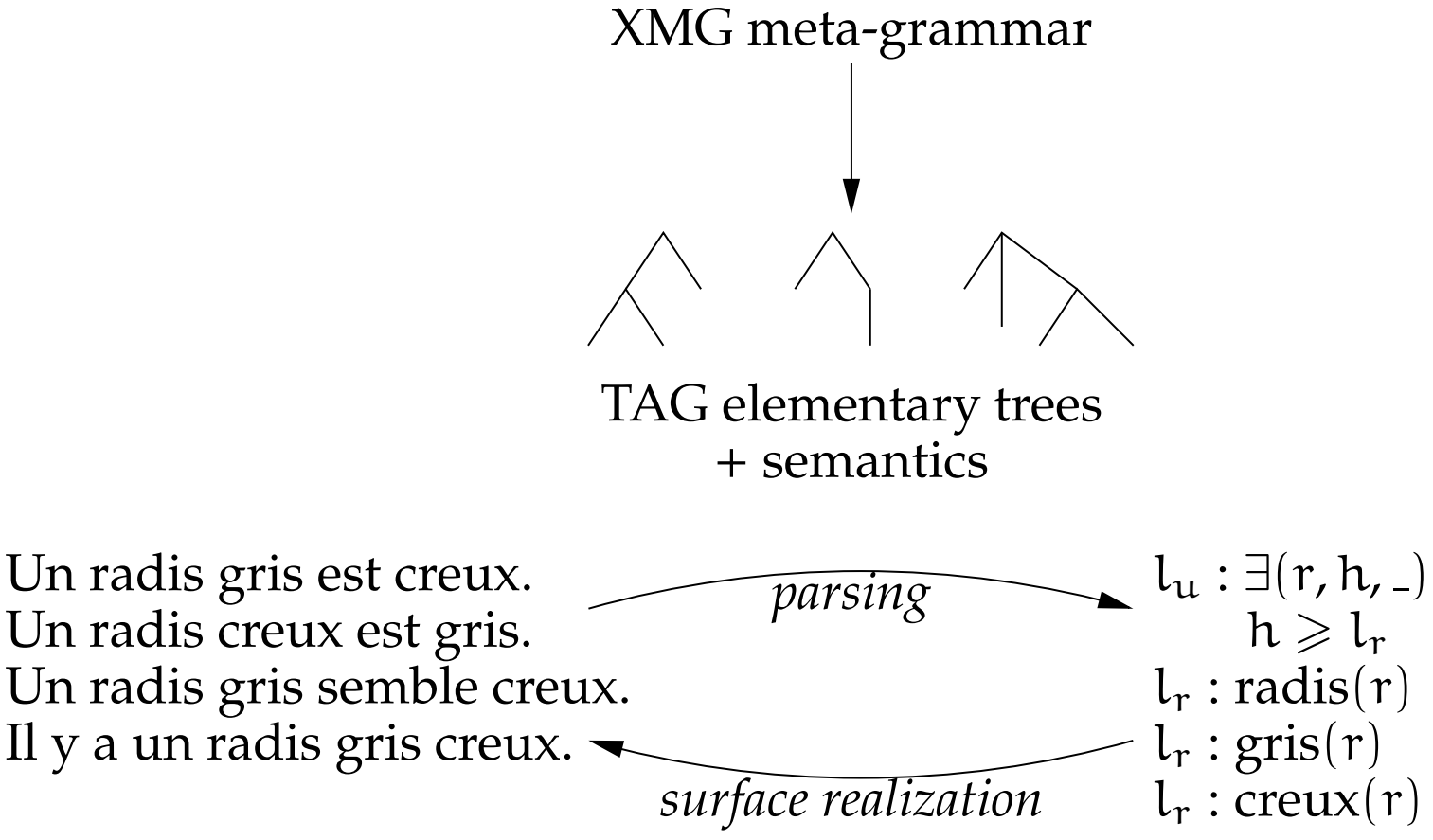
Sylvain Schmitz

LSV, ENS Cachan & CNRS, France

LSV Seminairs, Cachan, September 23, 2008

SemFraG

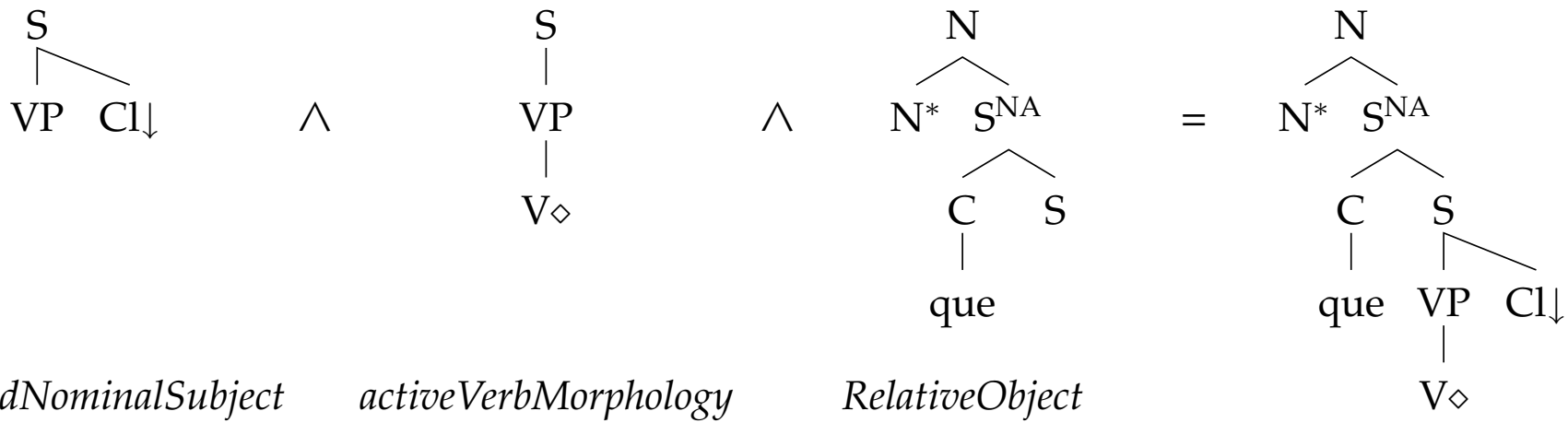
Crabbé [2005], Gardent [2008]



Meta Grammar

XMG, Crabbé [2005] and many others

CanonicalSubject *activeVerbMorphology* *CanonicalObject*



InvertedNominalSubject *activeVerbMorphology* *RelativeObject*

Grammar Evaluation

Against TSNLP [Lehmann and Oepen, 1996]

- undergeneration
  - 76% of grammatical sentences parsed
- overgeneration
  - 83% of agrammatical sentences rejected
  - 1.65 parses per correct sentence on average
- how can we improve the grammar?

Error Mining for Undergeneration

van Noord [2004], Sagot and Éric de la Clergerie [2006]

1. parse a large corpus of correct sentences
2. failures indicate coverage issues
3. statistical analysis identifies a probable culprit for each failure
4. might attempt to provide corrections

Error Mining for Undergeneration

van Noord [2004], Sagot and Éric de la Clergerie [2006]

1. parse a large corpus of correct sentences
2. failures indicate coverage issues
3. statistical analysis identifies a probable culprit for each failure
4. might attempt to provide corrections

For Overgeneration?

Which test suite for pass/failure?

- a TreeBank
- a corpus of incorrect sentences
- sentences generated from the grammar
  - which input?
  - generation from logic formulae is NP-complete

For Overgeneration?

Which test suite for pass/failure?

- a TreeBank
- a corpus of incorrect sentences
- sentences generated from the grammar
  - which input?
  - generation from logic formulae is NP-complete

For Overgeneration?

Which test suite for pass/failure?

- a TreeBank
- a corpus of incorrect sentences
- sentences generated from the grammar
  - which input?
  - generation from logic formulae is NP-complete

For Overgeneration?

Which test suite for pass/failure?

- a TreeBank
- a corpus of incorrect sentences
- sentences generated from the grammar
  - which input?
  - generation from logic formulae is NP-complete

“Exhaustive” Generation

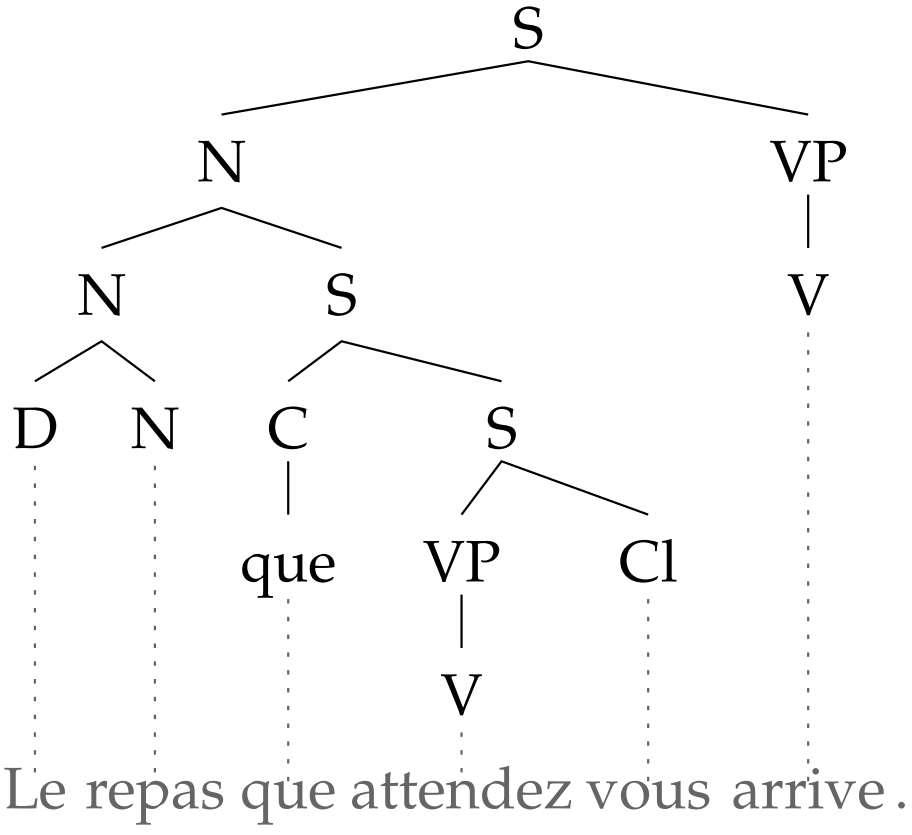
- not in terms of elementary trees (about 6,000)
- in terms of *linguistic phenomena*
  - grammar compiled from a meta grammar
  - compilation traces
  - 110 classes match linguistic phenomena

Guided Generation

Input:
bag of classes

{ *InvertedNominalSubject*, *RelativeObject* }

Output:
set of trees

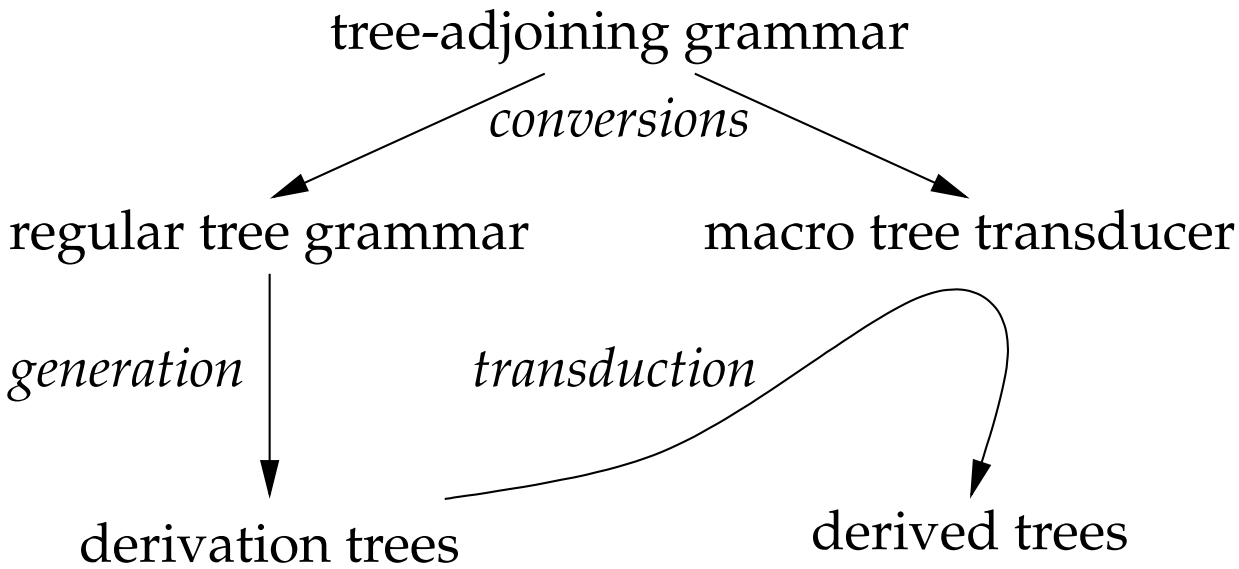


Example of a Test Suite

*Jean qu'attends-tu est grand.
Jean qui attend se lave.
Le chat noir est grand.
Il le faut.
Beaucoup de chats noirs se lavent.
Jean qu'il attend agit.
*Le chat est avec beaucoup de poils grand.
Jean qui agit est grand.
*Le repas qu'attendez-vous arrive.
*Jean est avec chat grand.

Digression: 2-level TAG Syntax

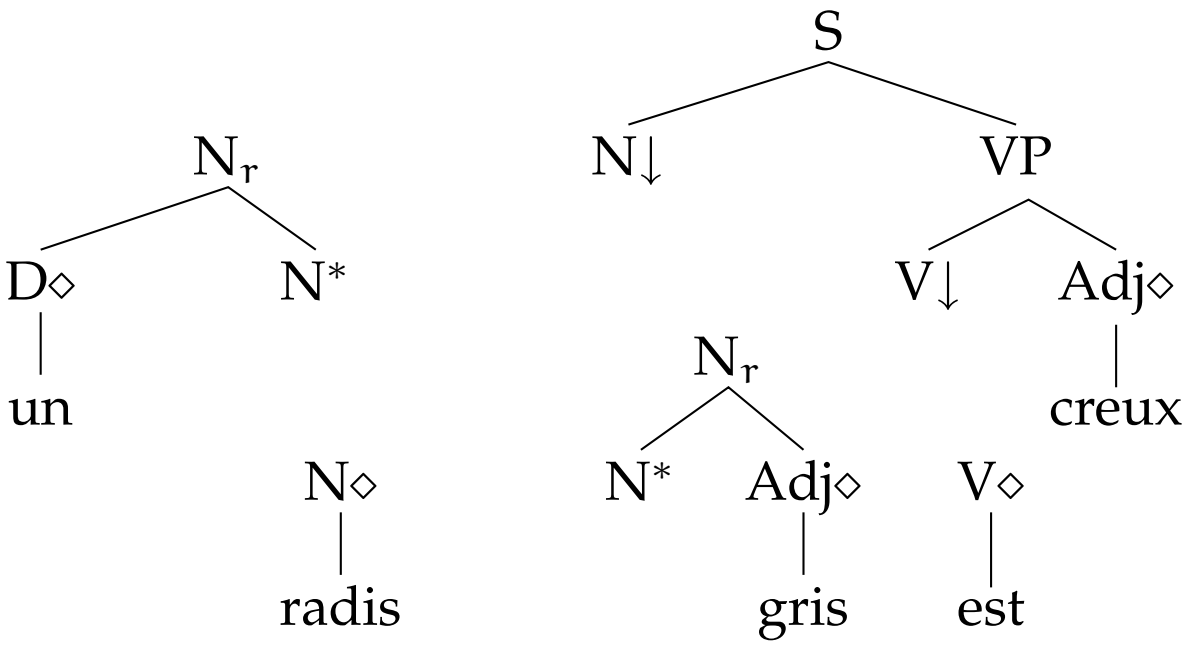
Shieber [2006] and many others



Tree Adjoining Grammars

Joshi et al. [1975], Joshi and Schabes [1997]

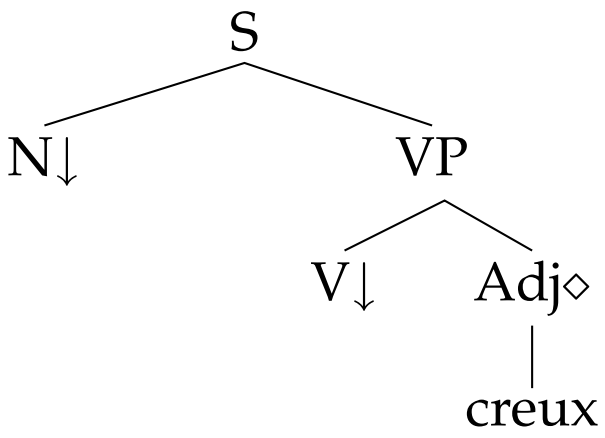
Elementary trees:



Tree Adjoining Grammars

Joshi et al. [1975], Joshi and Schabes [1997]

Two operations:

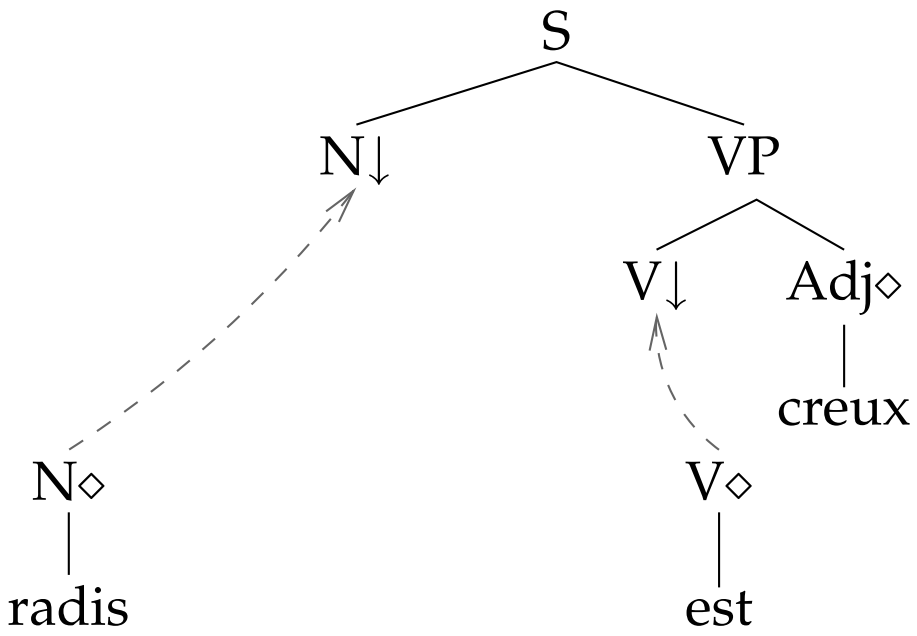


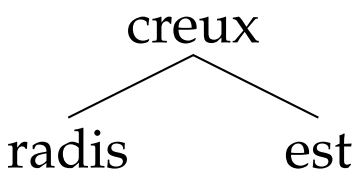
creux

Tree Adjoining Grammars

Joshi et al. [1975], Joshi and Schabes [1997]

Two operations: substitution:

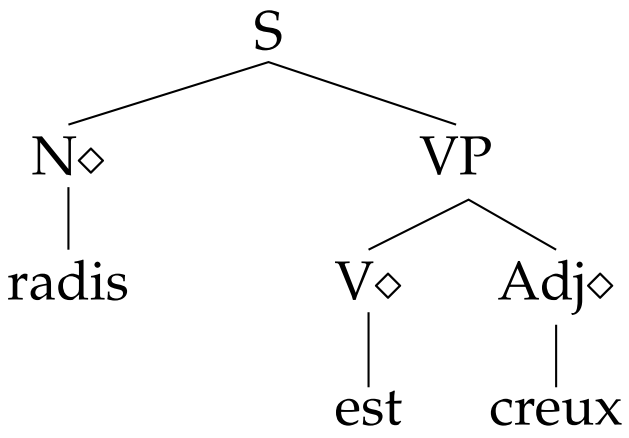


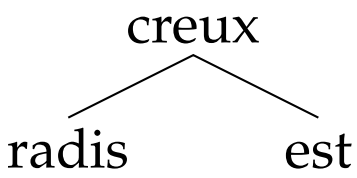


Tree Adjoining Grammars

Joshi et al. [1975], Joshi and Schabes [1997]

Two operations: substitution:

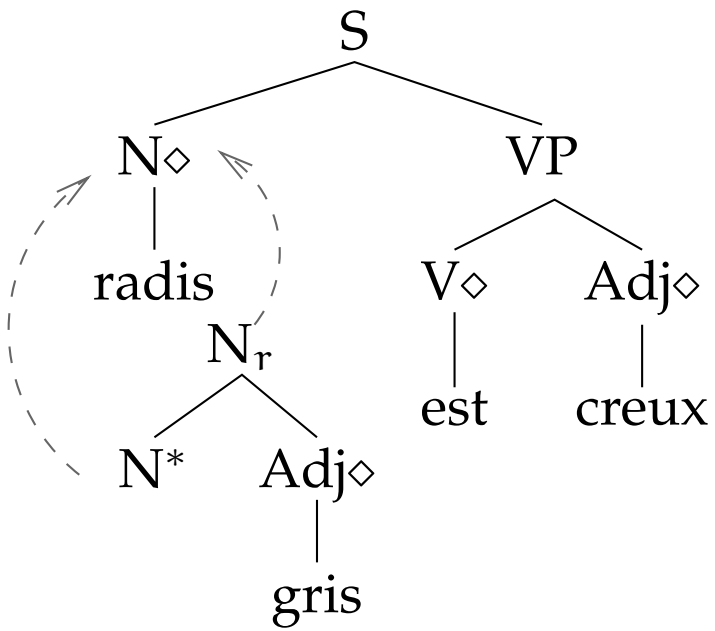


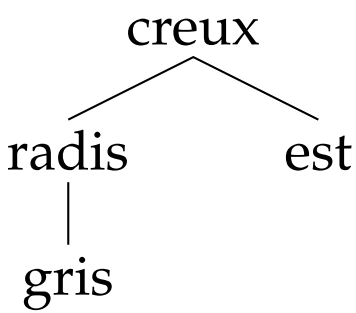


Tree Adjoining Grammars

Joshi et al. [1975], Joshi and Schabes [1997]

Two operations: adjunction:

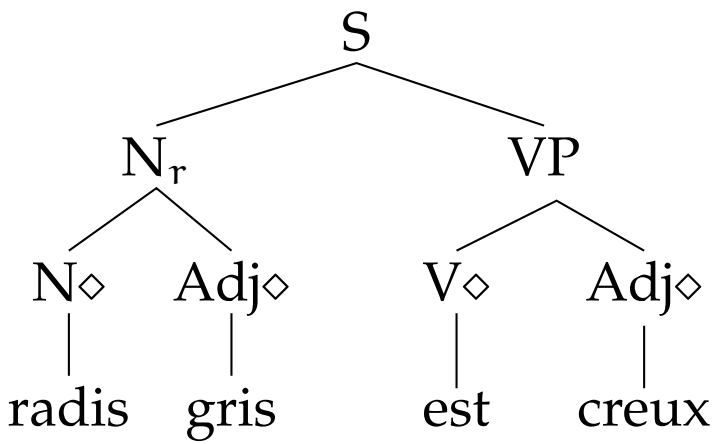


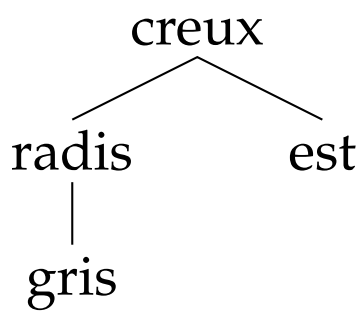


Tree Adjoining Grammars

Joshi et al. [1975], Joshi and Schabes [1997]

Two operations: adjunction:

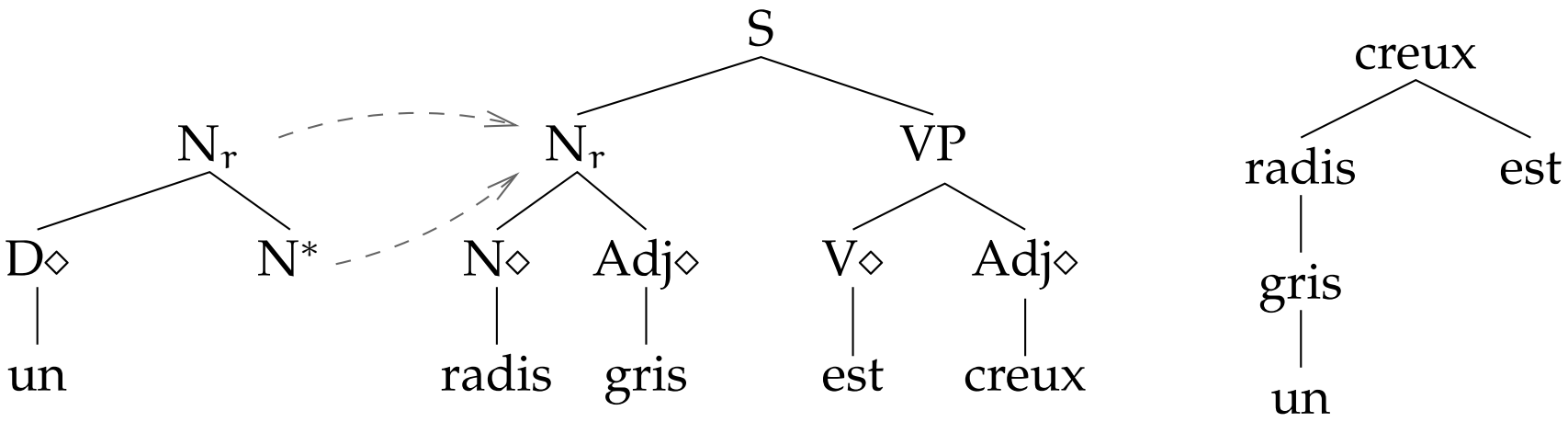




Tree Adjoining Grammars

Joshi et al. [1975], Joshi and Schabes [1997]

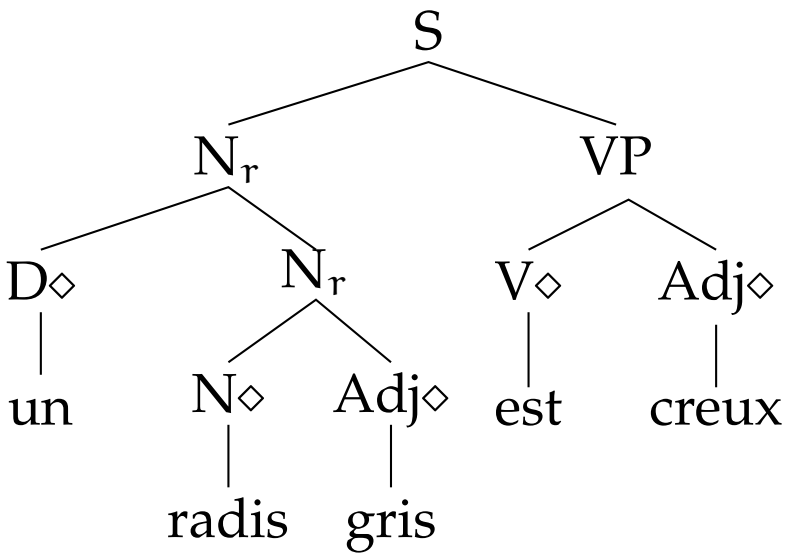
Two operations: adjunction:

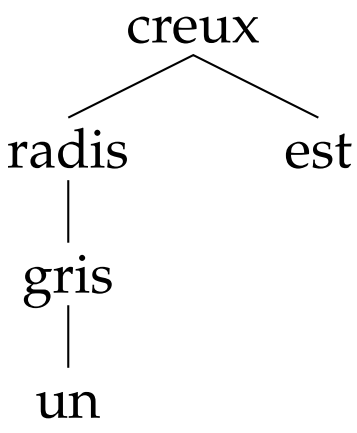


Tree Adjoining Grammars

Joshi et al. [1975], Joshi and Schabes [1997]

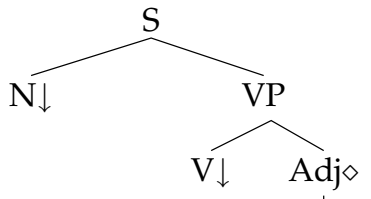
Derived and derivation trees:





Derivation Tree Language (1)

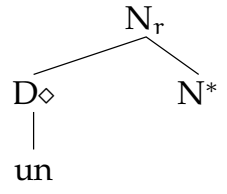
(reduced) Regular tree grammar



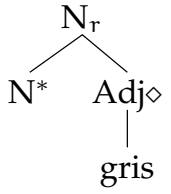
$$S_I \rightarrow \text{creux}(S_A, N_I, VP_A, V_I, Adj_A)$$

$$N_I \rightarrow \text{radis}(N_A)$$

$$V_I \rightarrow \text{est}(V_A)$$



$$N_A \rightarrow \text{un}(N_A, D_A)$$



$$N_A \rightarrow \text{gris}(N_A, Adj_A)$$

$$Adj_A \rightarrow \varepsilon(),\ D_A \rightarrow \varepsilon(),\ N_A \rightarrow \varepsilon(),\ S_A \rightarrow \varepsilon(),\ V_A \rightarrow \varepsilon(),\ VP_A \rightarrow \varepsilon()$$

Derivation Tree Language (1)

(reduced) Regular tree grammar

$$S_I \rightarrow \text{creux}(S_A, N_I, VP_A, V_I, Adj_A)$$

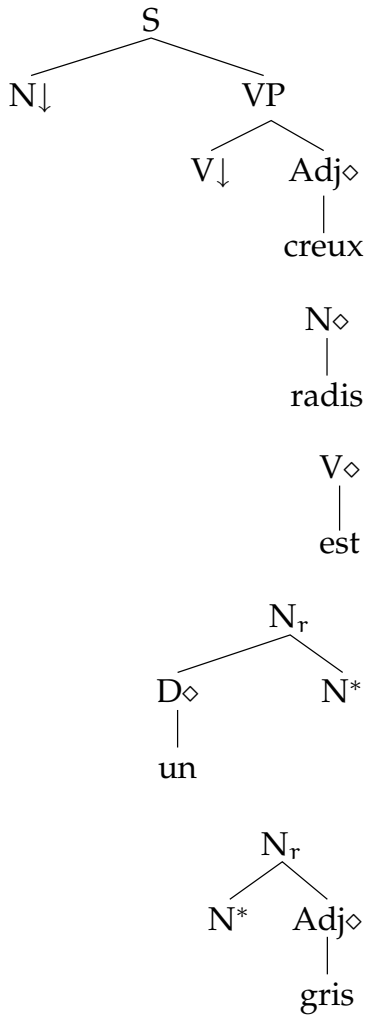
$$N_I \rightarrow \text{radis}(N_A)$$

$$V_I \rightarrow \text{est}(V_A)$$

$$N_A \rightarrow \text{un}(N_A, D_A)$$

$$N_A \rightarrow \text{gris}(N_A, Adj_A)$$

$$Adj_A \rightarrow \varepsilon(),\ D_A \rightarrow \varepsilon(),\ N_A \rightarrow \varepsilon(),\ S_A \rightarrow \varepsilon(),\ V_A \rightarrow \varepsilon(),\ VP_A \rightarrow \varepsilon()$$



From Derivations to Derived Trees

Macro tree transducer

$$q_I(\text{creux}(t_1, t_2)) \rightarrow S(q_I(t_1),\ VP(q_I(t_2),\ \text{Adj}(\text{creux})))$$

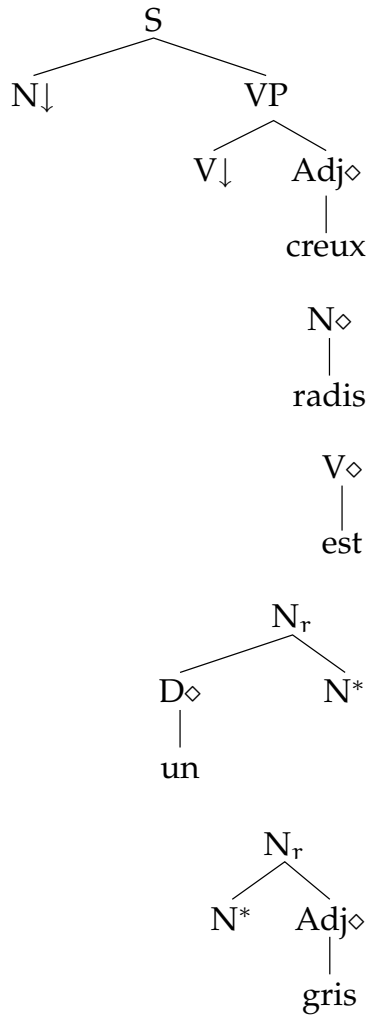
$$q_I(\text{radis}(t_1)) \rightarrow q_A(t_1,\ N(\text{radis}))$$

$$q_I(\text{est}()) \rightarrow V(\text{est})$$

$$q_A(\text{un}(t_1),\ y) \rightarrow q_A(t_1,\ N(D(\text{un}),\ y))$$

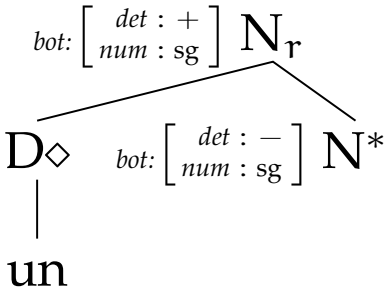
$$q_A(\text{gris}(t_1),\ y) \rightarrow q_A(t_1,\ N(y,\ \text{Adj}(\text{gris})))$$

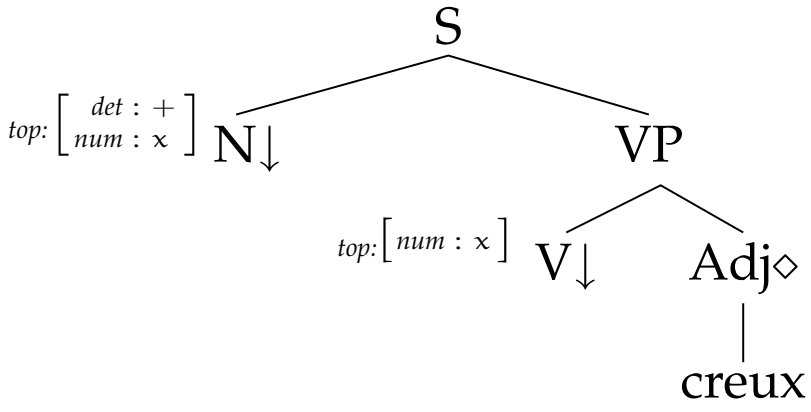
$$q_A(\varepsilon(),\ y) \rightarrow y$$

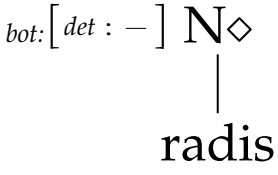


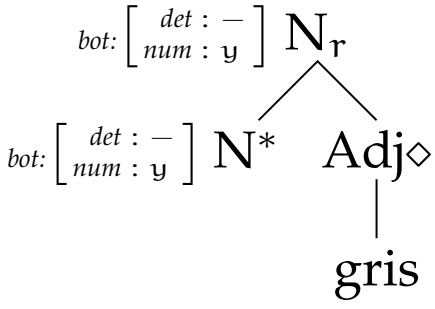
Feature-Based TAG

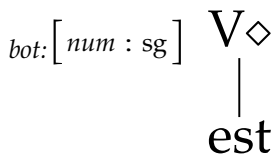
Vijay-Shanker and Joshi [1988]





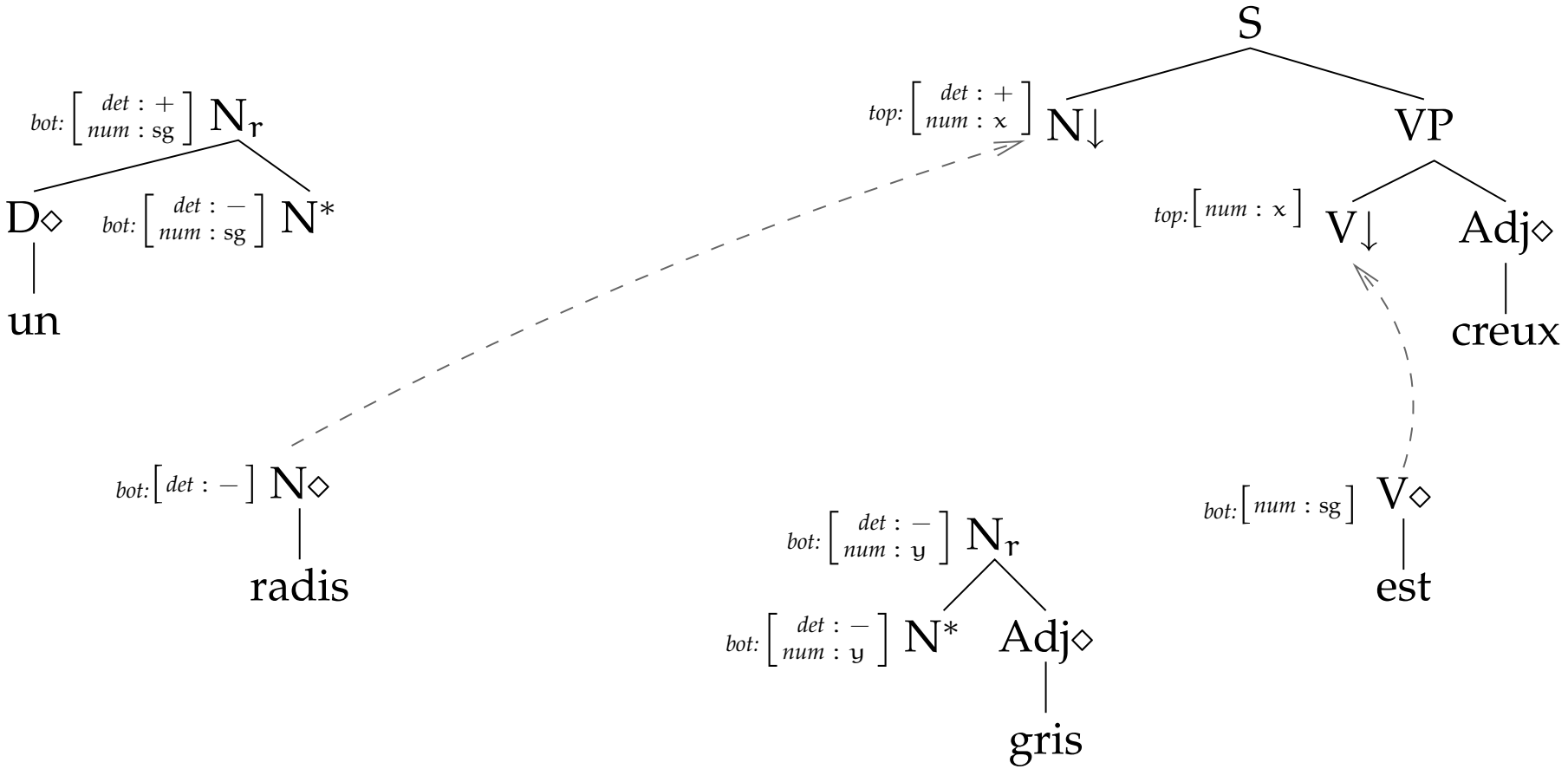






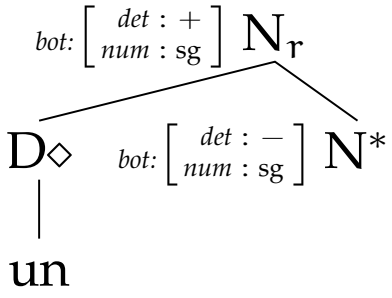
Feature-Based TAG

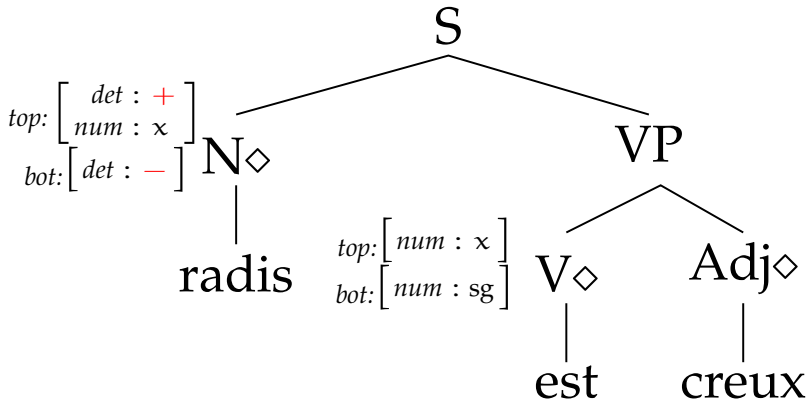
Vijay-Shanker and Joshi [1988]

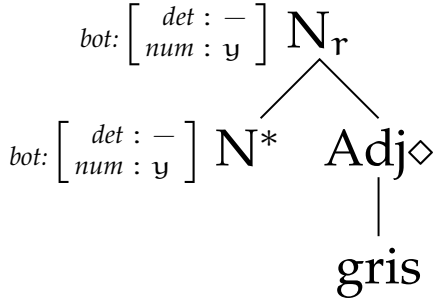


Feature-Based TAG

Vijay-Shanker and Joshi [1988]

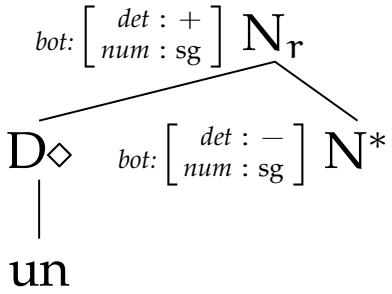


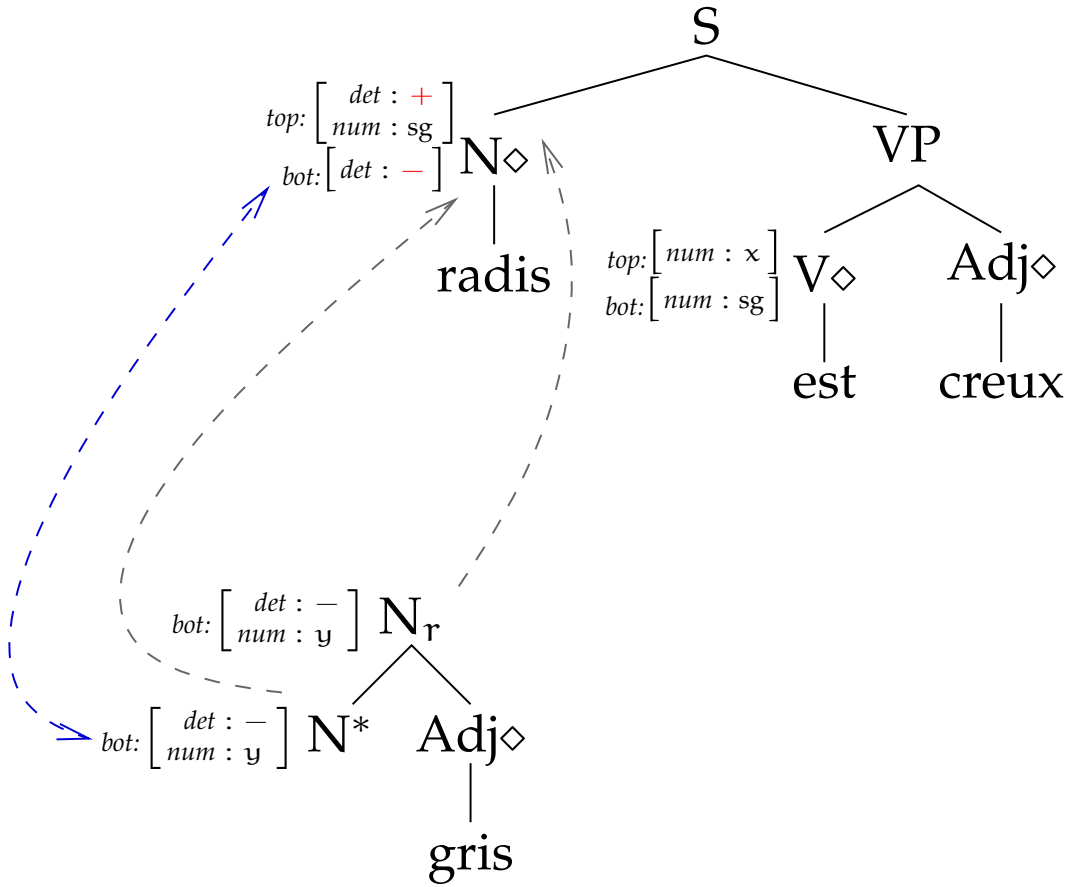




Feature-Based TAG

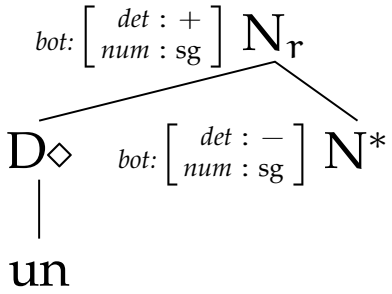
Vijay-Shanker and Joshi [1988]

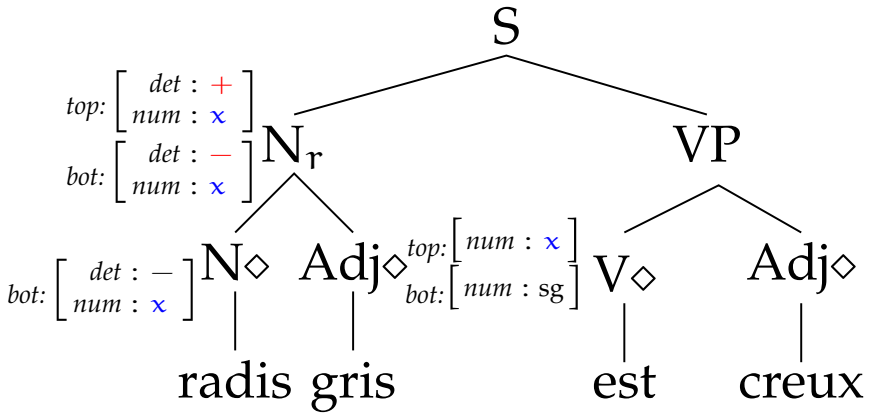




Feature-Based TAG

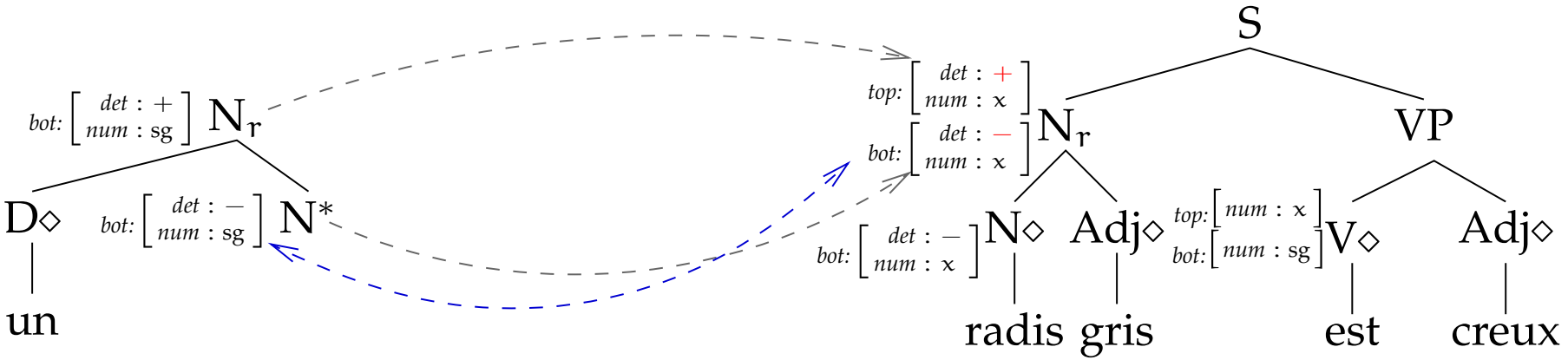
Vijay-Shanker and Joshi [1988]





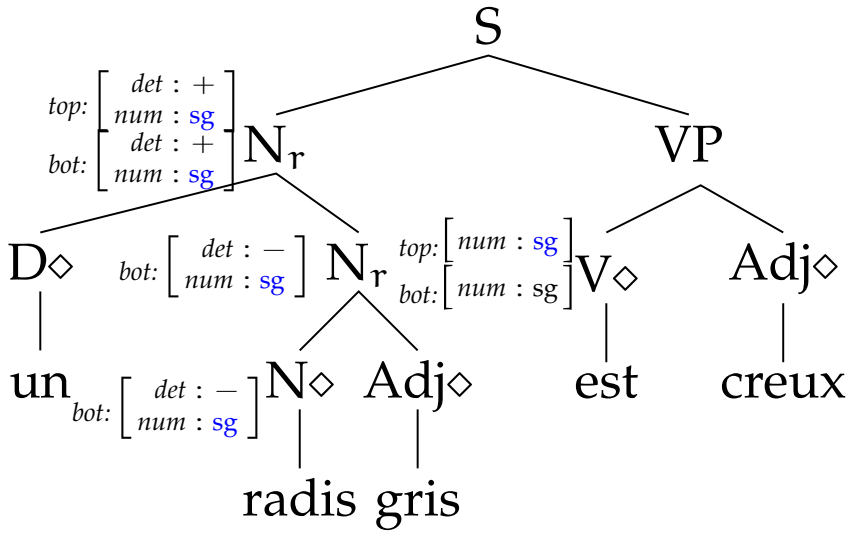
Feature-Based TAG

Vijay-Shanker and Joshi [1988]



Feature-Based TAG

Vijay-Shanker and Joshi [1988]



Features in RTG

- $\mathcal{D}$: set of feature structures
- work on terms over $N \times \mathcal{D}$
- rules of form $(A, d) \to a((B_1, d'_1), \ldots, (B_n, d'_n))$
- rewrites $(s, e) \Rightarrow (t, e')$ maintain a global environment for substitutions in feature structures

$$s = C[(A, d')],\ t = C[a((B_1, \sigma(d'_1)), \ldots, (B_n, \sigma(d'_n)))]$$
$$\sigma = \mathsf{mgu}(d, e(d')),\ e' = \sigma \circ e$$

- language

$$L(G) = \{t \in T(\mathcal{F}) \mid \exists e, ((S, \top), id) \Rightarrow^* (t, e)\}$$

Derivation Tree Language (2)

$$(S_I, \top) \rightarrow \text{creux}\left(N_I\left[top : \begin{bmatrix} det : + \\ num : x \end{bmatrix}\right], V_I\left[top : [num : x]\right]\right)$$

$$N_I\left[top : t\right] \rightarrow \text{radis}\left(N_A\begin{bmatrix} top : t \\ bot : [det : -] \end{bmatrix}\right)$$

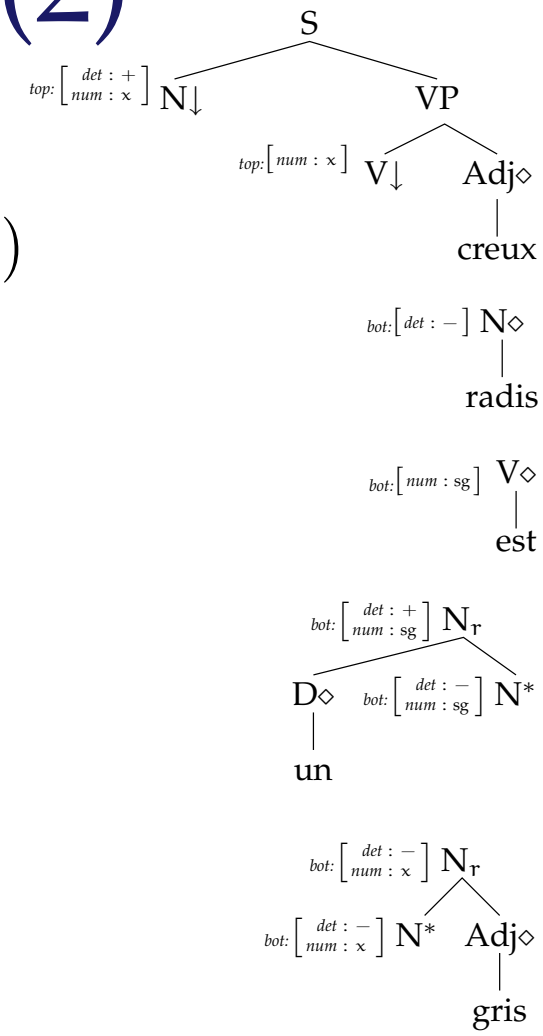
$$V_I\left[top : t\right] \rightarrow \text{est}\left(V_A\begin{bmatrix} top : t \\ bot : [num : sg] \end{bmatrix}\right)$$

$$N_A\begin{bmatrix} top : t \\ bot : \begin{bmatrix} det : - \\ num : sg \end{bmatrix} \end{bmatrix} \rightarrow \text{un}\left(N_A\begin{bmatrix} top : t \\ bot : \begin{bmatrix} det : + \\ num : sg \end{bmatrix} \end{bmatrix}\right)$$

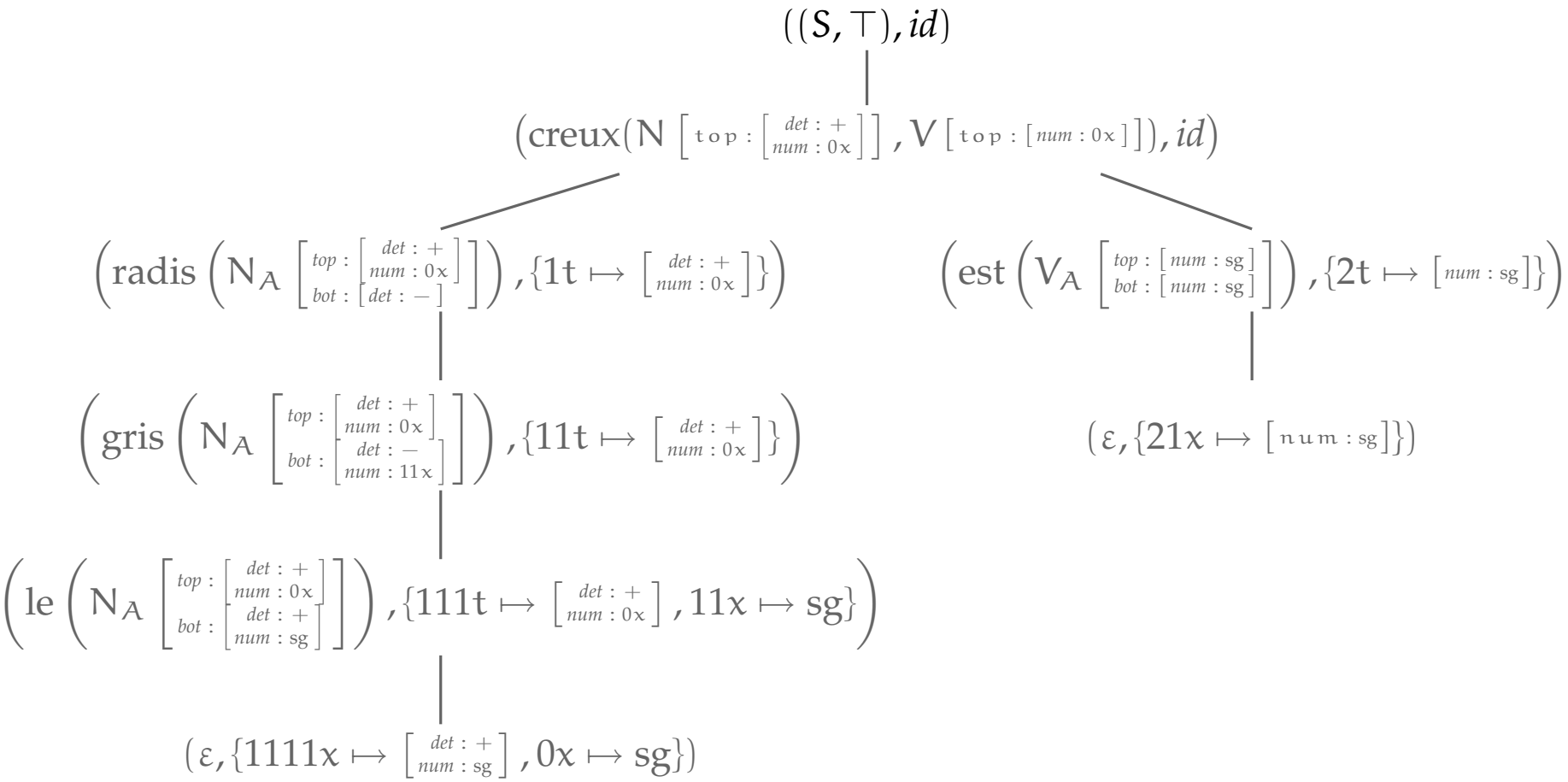
$$N_A\begin{bmatrix} top : t \\ bot : \begin{bmatrix} det : - \\ num : x \end{bmatrix} \end{bmatrix} \rightarrow \text{gris}\left(N_A\begin{bmatrix} top : t \\ bot : \begin{bmatrix} det : - \\ num : x \end{bmatrix} \end{bmatrix}\right)$$

$$N_A\begin{bmatrix} top : x \\ bot : x \end{bmatrix} \rightarrow \varepsilon()$$

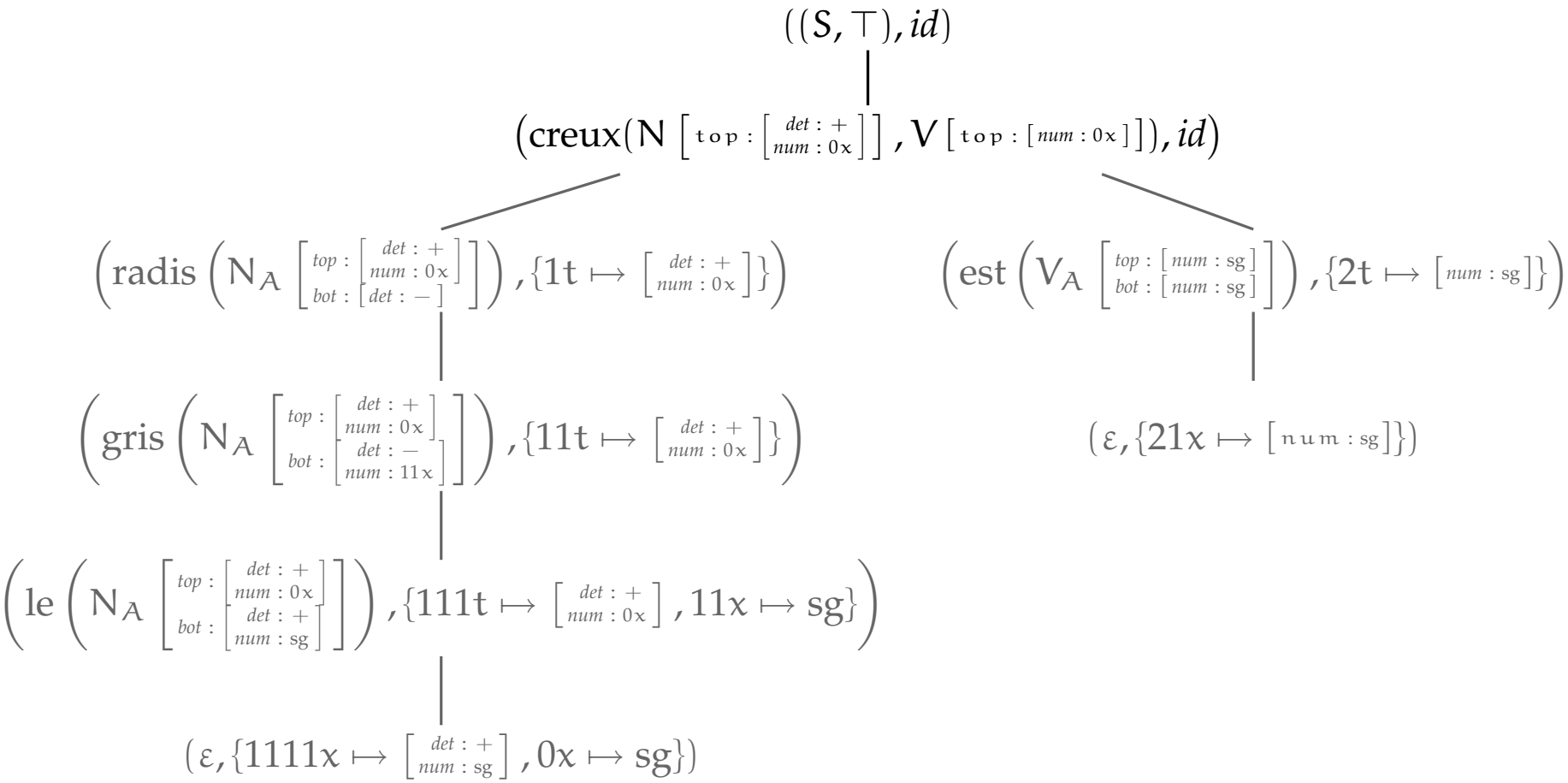
$$V_A\begin{bmatrix} top : x \\ bot : x \end{bmatrix} \rightarrow \varepsilon()$$



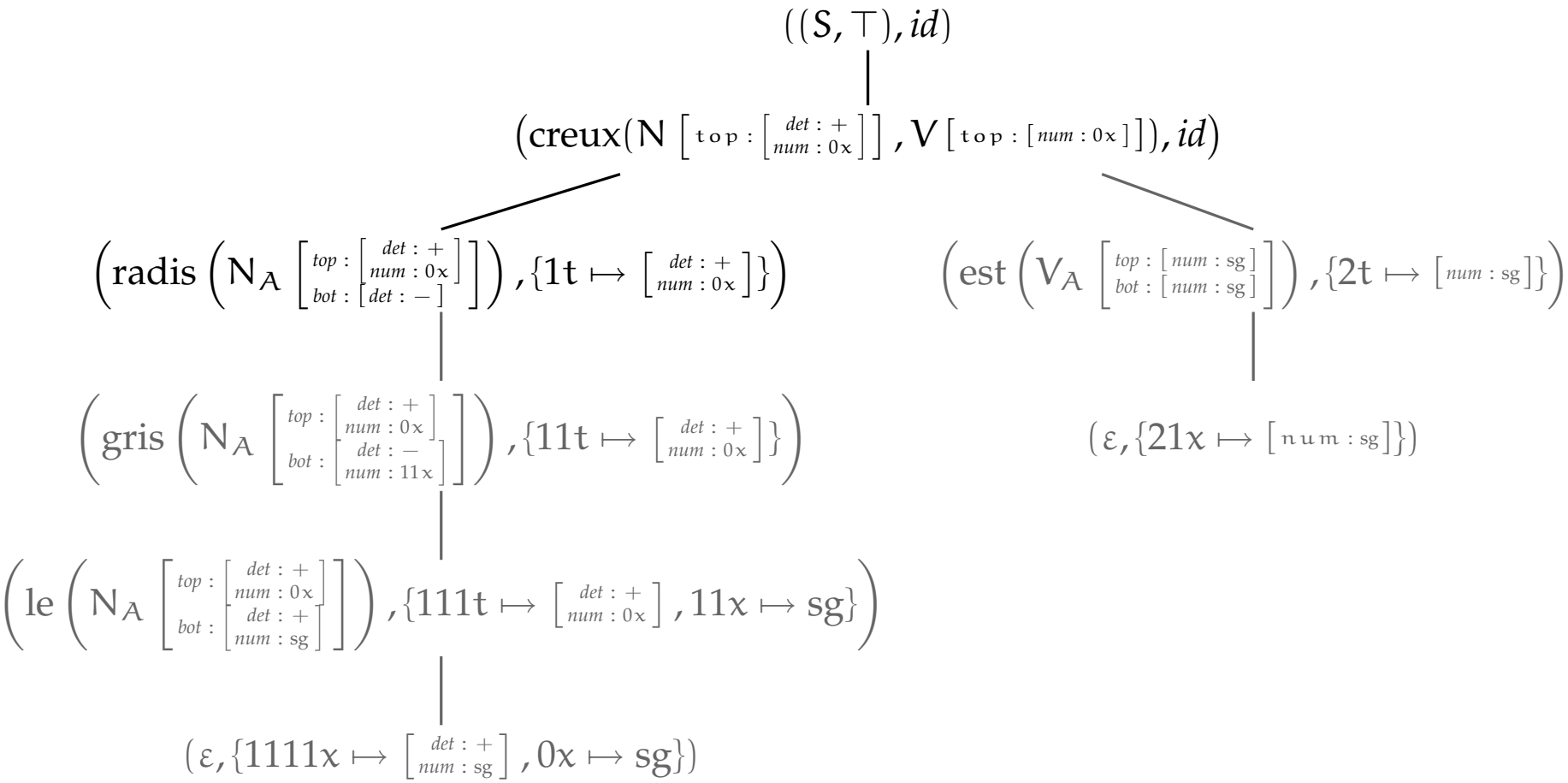
Example


((S, ⊤), id)
(creux(N [top : [det : +, num : 0x]], V [top : [num : 0x]]), id)
(radis (N_A [top : [det : +, num : 0x], bot : [det : −]]), {1t ↦ [det : +, num : 0x]})
(est (V_A [top : [num : sg], bot : [num : sg]]), {2t ↦ [num : sg]})
(gris (N_A [top : [det : +, num : 0x], bot : [det : −, num : 11x]]), {11t ↦ [det : +, num : 0x]})
(ε, {21x ↦ [num : sg]})
(le (N_A [top : [det : +, num : 0x], bot : [det : +, num : sg]]), {111t ↦ [det : +, num : 0x], 11x ↦ sg})
(ε, {1111x ↦ [det : +, num : sg], 0x ↦ sg})

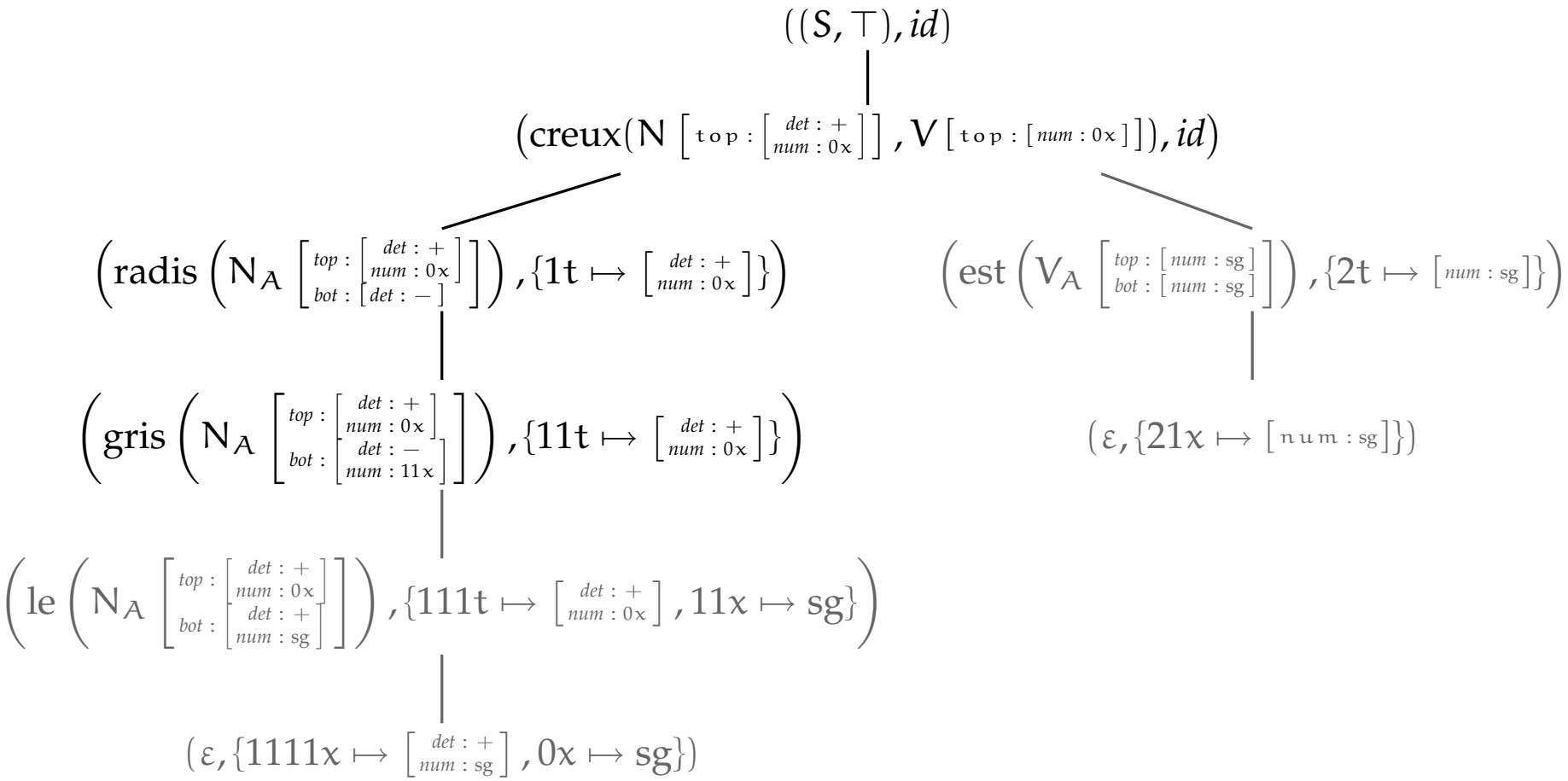
Example


((S, ⊤), id)
(creux(N [top : [det : +, num : 0x]], V [top : [num : 0x]]), id)
(radis (N_A [top : [det : +, num : 0x], bot : [det : −]]), {1t ↦ [det : +, num : 0x]})
(est (V_A [top : [num : sg], bot : [num : sg]]), {2t ↦ [num : sg]})
(gris (N_A [top : [det : +, num : 0x], bot : [det : −, num : 11x]]), {11t ↦ [det : +, num : 0x]})
(ε, {21x ↦ [num : sg]})
(le (N_A [top : [det : +, num : 0x], bot : [det : +, num : sg]]), {111t ↦ [det : +, num : 0x], 11x ↦ sg})
(ε, {1111x ↦ [det : +, num : sg], 0x ↦ sg})

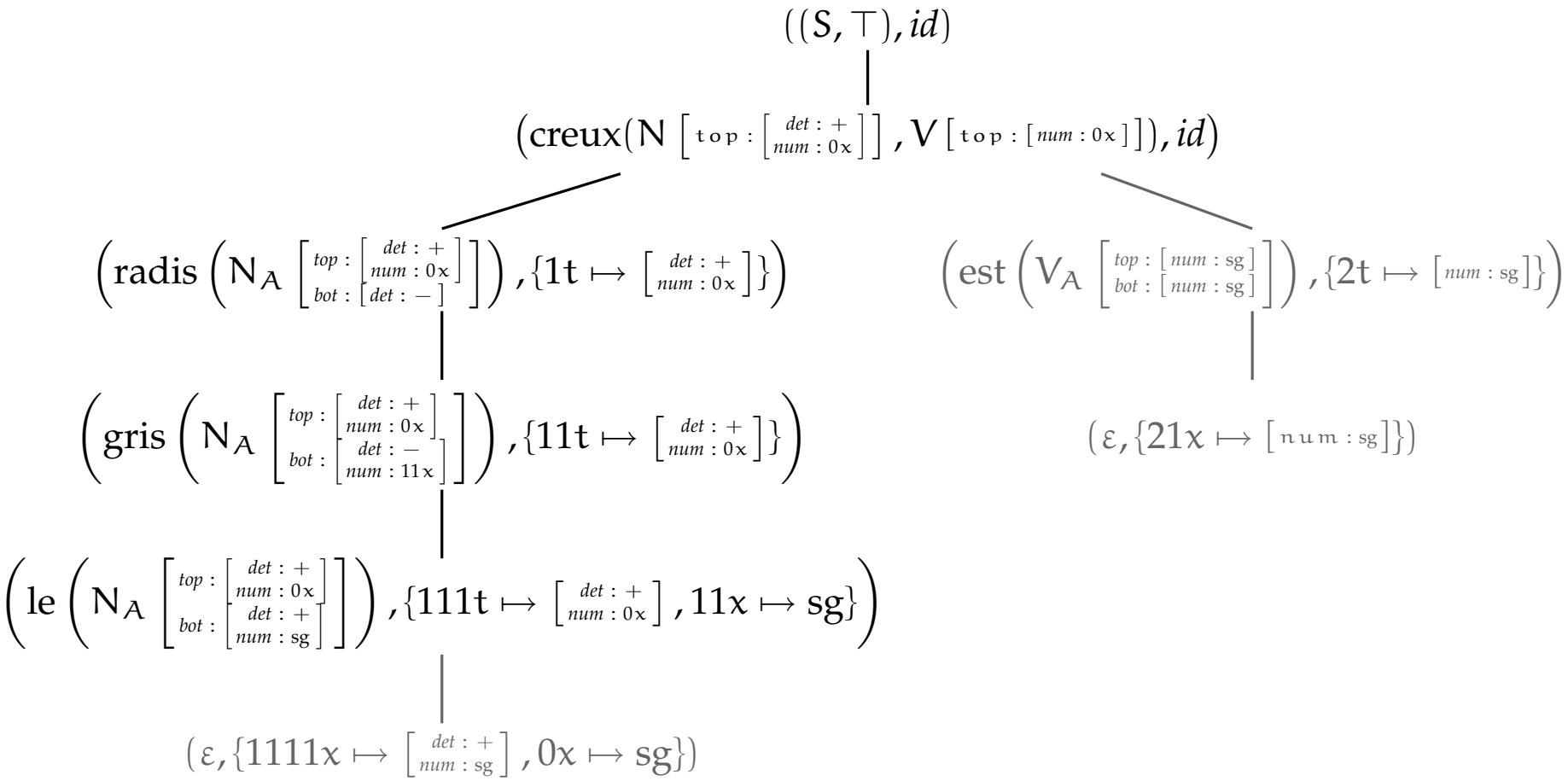
Example


((S, ⊤), id)
(creux(N [top : [det : +, num : 0x]], V [top : [num : 0x]]), id)
(radis (N_A [top : [det : +, num : 0x], bot : [det : −]]), {1t ↦ [det : +, num : 0x]})
(est (V_A [top : [num : sg], bot : [num : sg]]), {2t ↦ [num : sg]})
(gris (N_A [top : [det : +, num : 0x], bot : [det : −, num : 11x]]), {11t ↦ [det : +, num : 0x]})
(ε, {21x ↦ [num : sg]})
(le (N_A [top : [det : +, num : 0x], bot : [det : +, num : sg]]), {111t ↦ [det : +, num : 0x], 11x ↦ sg})
(ε, {1111x ↦ [det : +, num : sg], 0x ↦ sg})

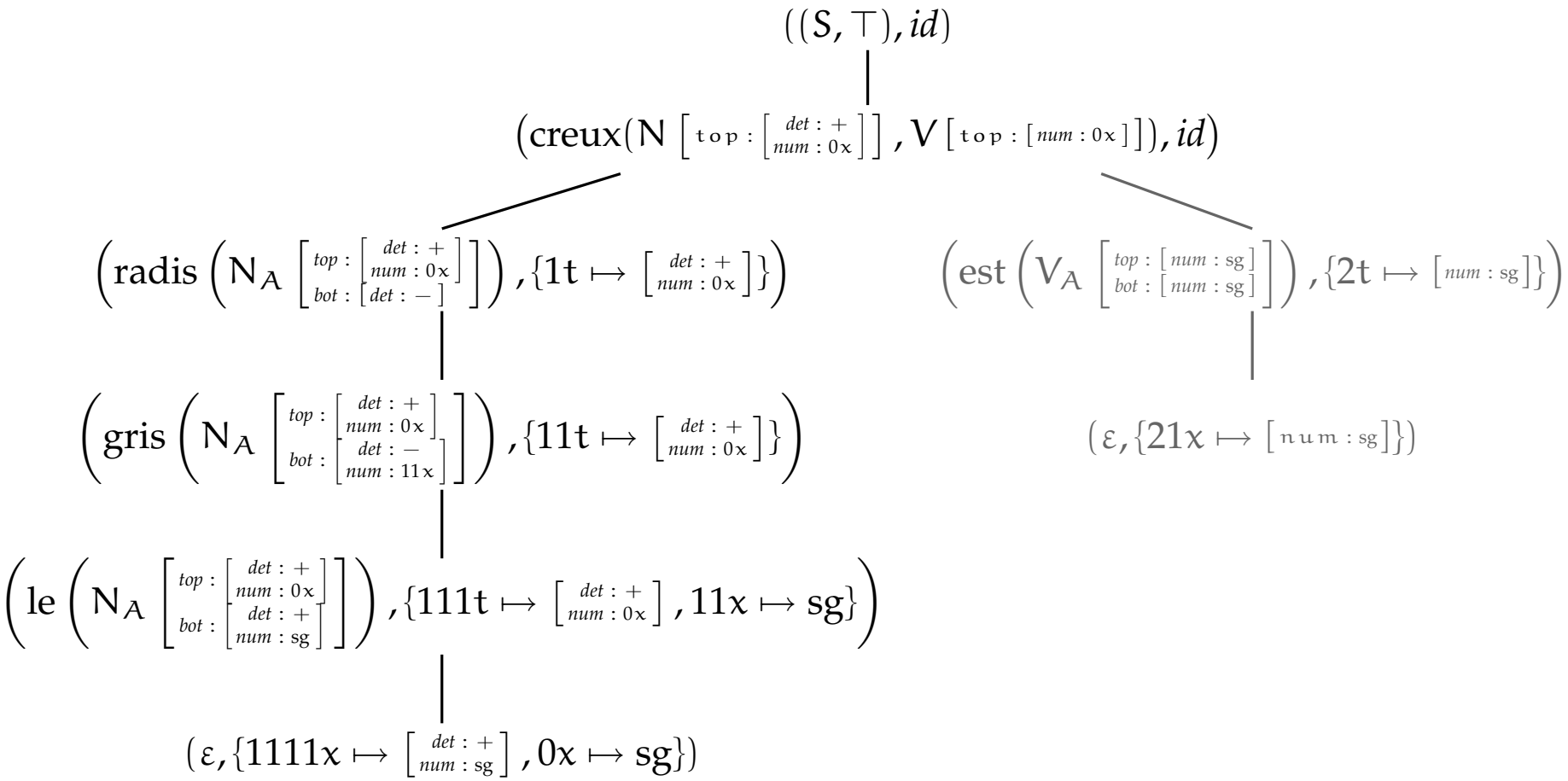
Example


((S, ⊤), id)
(creux(N [top : [det : +, num : 0x]], V [top : [num : 0x]]), id)
(radis (N_A [top : [det : +, num : 0x], bot : [det : −]]), {1t ↦ [det : +, num : 0x]})
(est (V_A [top : [num : sg], bot : [num : sg]]), {2t ↦ [num : sg]})
(gris (N_A [top : [det : +, num : 0x], bot : [det : −, num : 11x]]), {11t ↦ [det : +, num : 0x]})
(ε, {21x ↦ [num : sg]})
(le (N_A [top : [det : +, num : 0x], bot : [det : +, num : sg]]), {111t ↦ [det : +, num : 0x], 11x ↦ sg})
(ε, {1111x ↦ [det : +, num : sg], 0x ↦ sg})

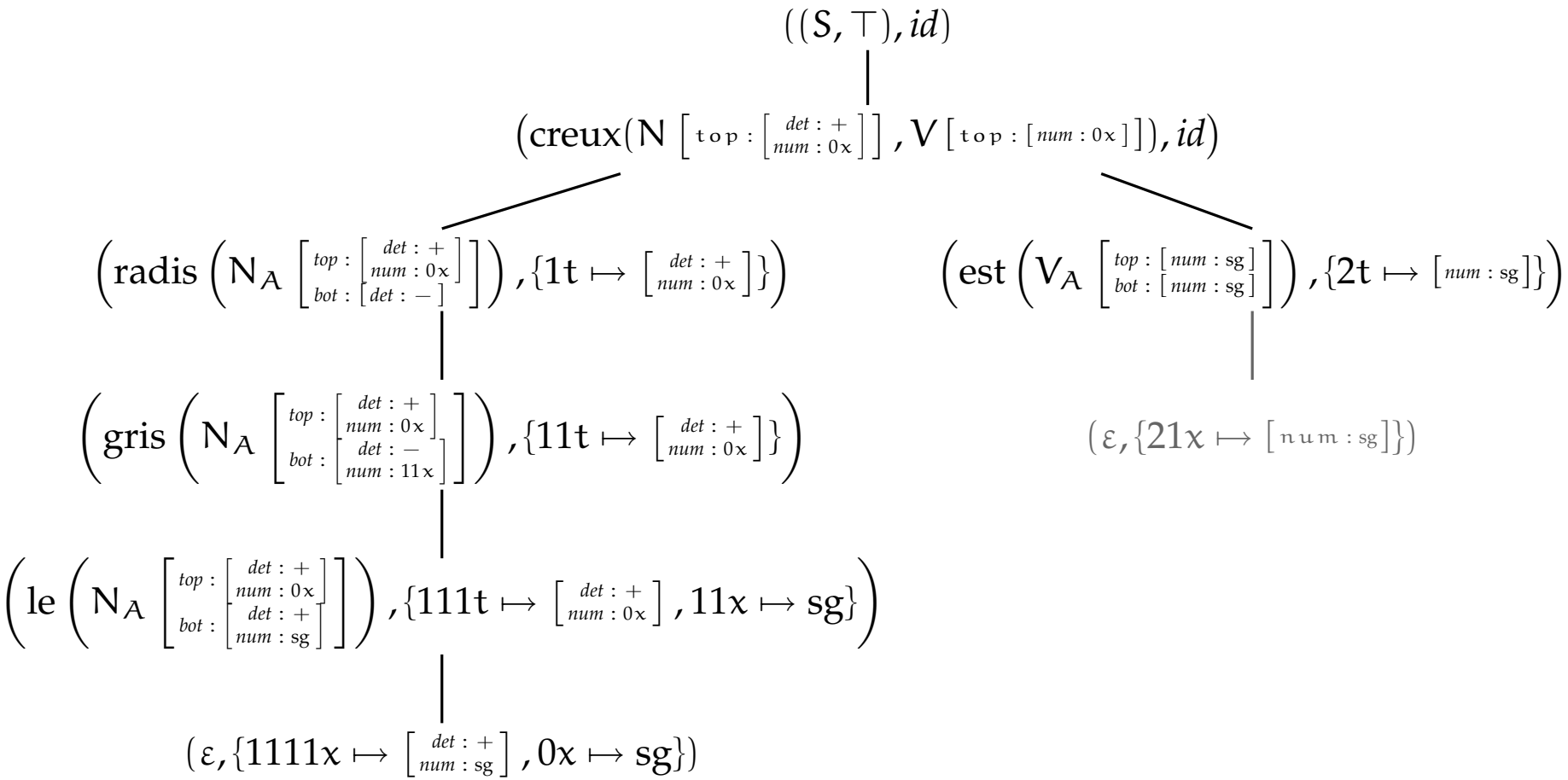
Example


((S, ⊤), id)
(creux(N [top : [det : +, num : 0x]], V [top : [num : 0x]]), id)
(radis (N_A [top : [det : +, num : 0x], bot : [det : −]]), {1t ↦ [det : +, num : 0x]})
(est (V_A [top : [num : sg], bot : [num : sg]]), {2t ↦ [num : sg]})
(gris (N_A [top : [det : +, num : 0x], bot : [det : −, num : 11x]]), {11t ↦ [det : +, num : 0x]})
(ε, {21x ↦ [num : sg]})
(le (N_A [top : [det : +, num : 0x], bot : [det : +, num : sg]]), {111t ↦ [det : +, num : 0x], 11x ↦ sg})
(ε, {1111x ↦ [det : +, num : sg], 0x ↦ sg})

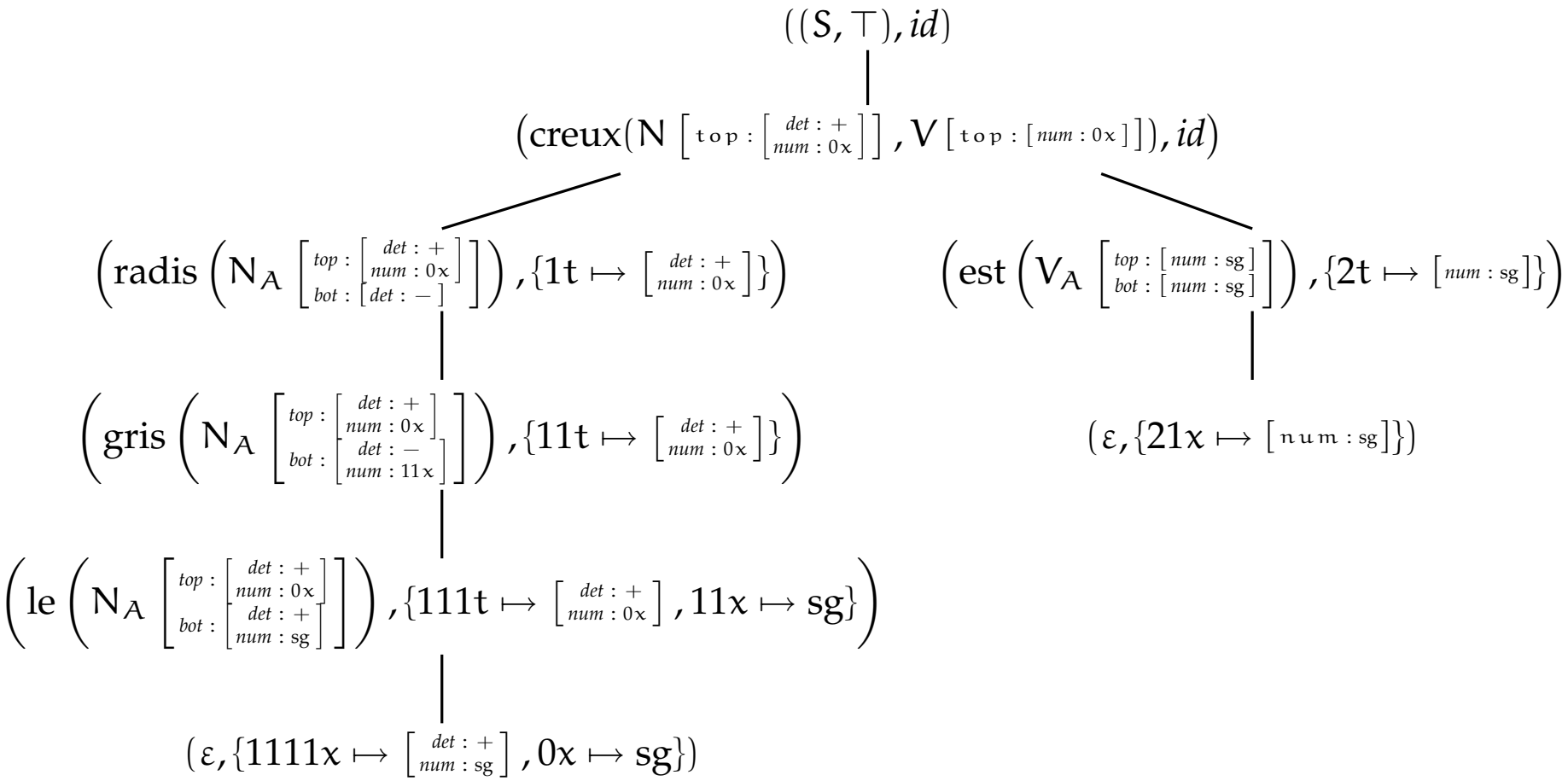
Example


((S, ⊤), id)
(creux(N [top : [det : +, num : 0x]], V [top : [num : 0x]]), id)
(radis (N_A [top : [det : +, num : 0x], bot : [det : −]]), {1t ↦ [det : +, num : 0x]})
(est (V_A [top : [num : sg], bot : [num : sg]]), {2t ↦ [num : sg]})
(gris (N_A [top : [det : +, num : 0x], bot : [det : −, num : 11x]]), {11t ↦ [det : +, num : 0x]})
(ε, {21x ↦ [num : sg]})
(le (N_A [top : [det : +, num : 0x], bot : [det : +, num : sg]]), {111t ↦ [det : +, num : 0x], 11x ↦ sg})
(ε, {1111x ↦ [det : +, num : sg], 0x ↦ sg})

Example


((S, ⊤), id)
(creux(N [top : [det : +, num : 0x]], V [top : [num : 0x]]), id)
(radis (N_A [top : [det : +, num : 0x], bot : [det : −]]), {1t ↦ [det : +, num : 0x]})
(est (V_A [top : [num : sg], bot : [num : sg]]), {2t ↦ [num : sg]})
(gris (N_A [top : [det : +, num : 0x], bot : [det : −, num : 11x]]), {11t ↦ [det : +, num : 0x]})
(ε, {21x ↦ [num : sg]})
(le (N_A [top : [det : +, num : 0x], bot : [det : +, num : sg]]), {111t ↦ [det : +, num : 0x], 11x ↦ sg})
(ε, {1111x ↦ [det : +, num : sg], 0x ↦ sg})

Example


((S, ⊤), id)
(creux(N [top : [det : + num : 0x]], V [top : [num : 0x]]), id)
(radis (N_A [top : [det : + num : 0x] bot : [det : −]]), {1t ↦ [det : + num : 0x]})
(est (V_A [top : [num : sg] bot : [num : sg]]), {2t ↦ [num : sg]})
(gris (N_A [top : [det : + num : 0x] bot : [det : − num : 11x]]), {11t ↦ [det : + num : 0x]})
(ε, {21x ↦ [num : sg]})
(le (N_A [top : [det : + num : 0x] bot : [det : + num : sg]]), {111t ↦ [det : + num : 0x], 11x ↦ sg})
(ε, {1111x ↦ [det : + num : sg], 0x ↦ sg})

Left Corner Transform

Rosenkrantz and Lewis II [1970]

- top feature structures of root nodes usually empty
- reverse order of root adjunctions on initial trees

Derivation Tree Language (3)

$$(S, \top) \longrightarrow \text{creux}\left(\mathrm{N}_{\mathrm{I}}\left[top : \begin{bmatrix} det : + \\ num : \mathtt{x} \end{bmatrix}\right], \mathrm{V}_{\mathrm{I}}\left[top : [\, num : \mathtt{x}\,]\right]\right)$$

$$\mathrm{N}\left[bot : [\, det : -\,]\right] \longrightarrow \text{radis}$$

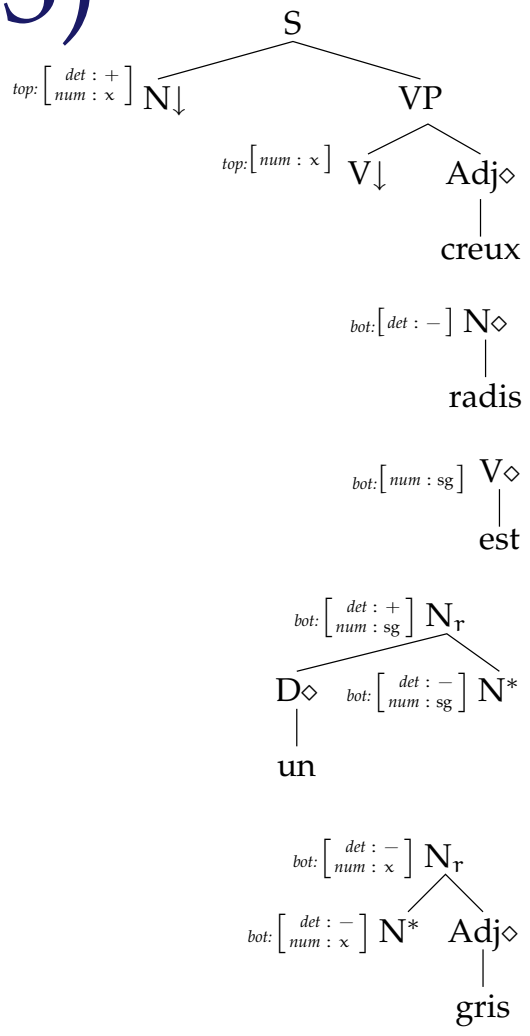
$$\mathrm{V}\left[bot : [\, num : \mathrm{sg}\,]\right] \longrightarrow \text{est}$$

$$\mathrm{N}\begin{bmatrix} top : \mathtt{t} \\ bot : \begin{bmatrix} det : + \\ num : \mathrm{sg} \end{bmatrix} \end{bmatrix} \longrightarrow \text{un}\left(\mathrm{N}\begin{bmatrix} top : \mathtt{t} \\ bot : \begin{bmatrix} det : - \\ num : \mathrm{sg} \end{bmatrix} \end{bmatrix}\right)$$

$$\mathrm{N}\begin{bmatrix} top : \mathtt{t} \\ bot : \begin{bmatrix} det : - \\ num : \mathtt{x} \end{bmatrix} \end{bmatrix} \longrightarrow \text{gris}\left(\mathrm{N}\begin{bmatrix} top : \mathtt{t} \\ bot : \begin{bmatrix} det : - \\ num : \mathtt{x} \end{bmatrix} \end{bmatrix}\right)$$

$$\mathrm{N}_{\mathrm{I}}\left[top : \mathtt{t}\right] \longrightarrow \varepsilon\left(\mathrm{N}\begin{bmatrix} top : \mathtt{t} \\ bot : \mathtt{t} \end{bmatrix}\right)$$

$$\mathrm{V}_{\mathrm{I}}\left[top : \mathtt{t}\right] \longrightarrow \varepsilon\left(\mathrm{V}\begin{bmatrix} top : \mathtt{t} \\ bot : \mathtt{t} \end{bmatrix}\right)$$



Test Suite Generation Algorithm

- greedy
- derivation-tree centric
- distances computed using the accessibility relation in the regular tree grammar
- elementary tree selection uses distances
  - to the remaining target classes
  - to the globally accumulated classes
  - to the classes accumulated in the current derivation

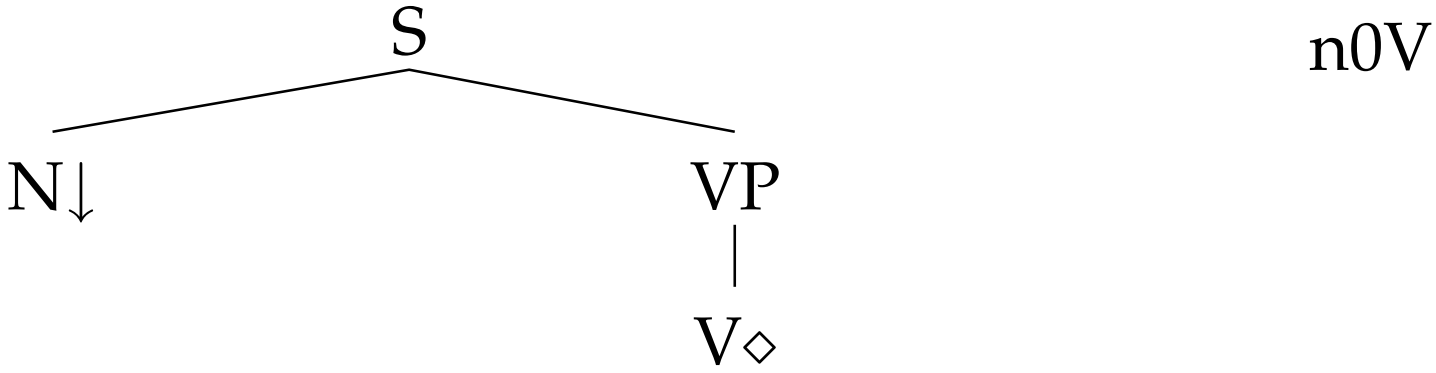
Algorithm

target: { *InvertedNominalSubject*, *RelativeObject* }

S

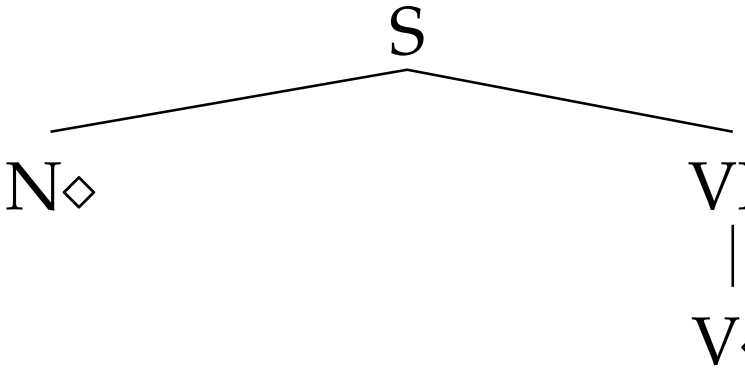
Algorithm

target: { *InvertedNominalSubject*, *RelativeObject* }



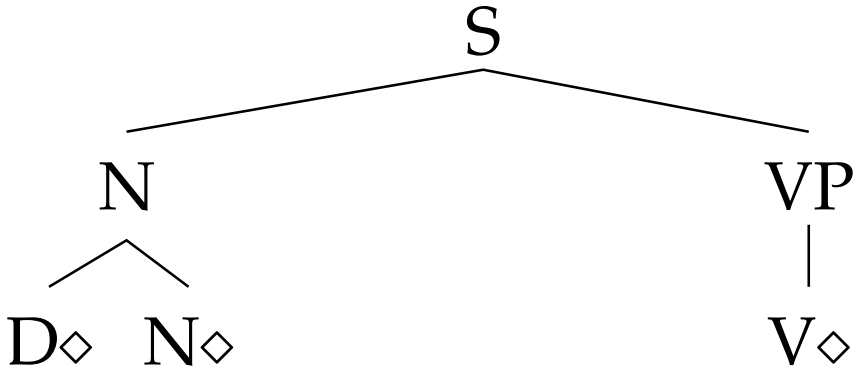
Algorithm

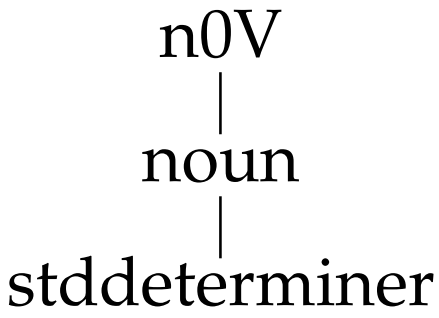
target: { *InvertedNominalSubject*, *RelativeObject* }



Algorithm

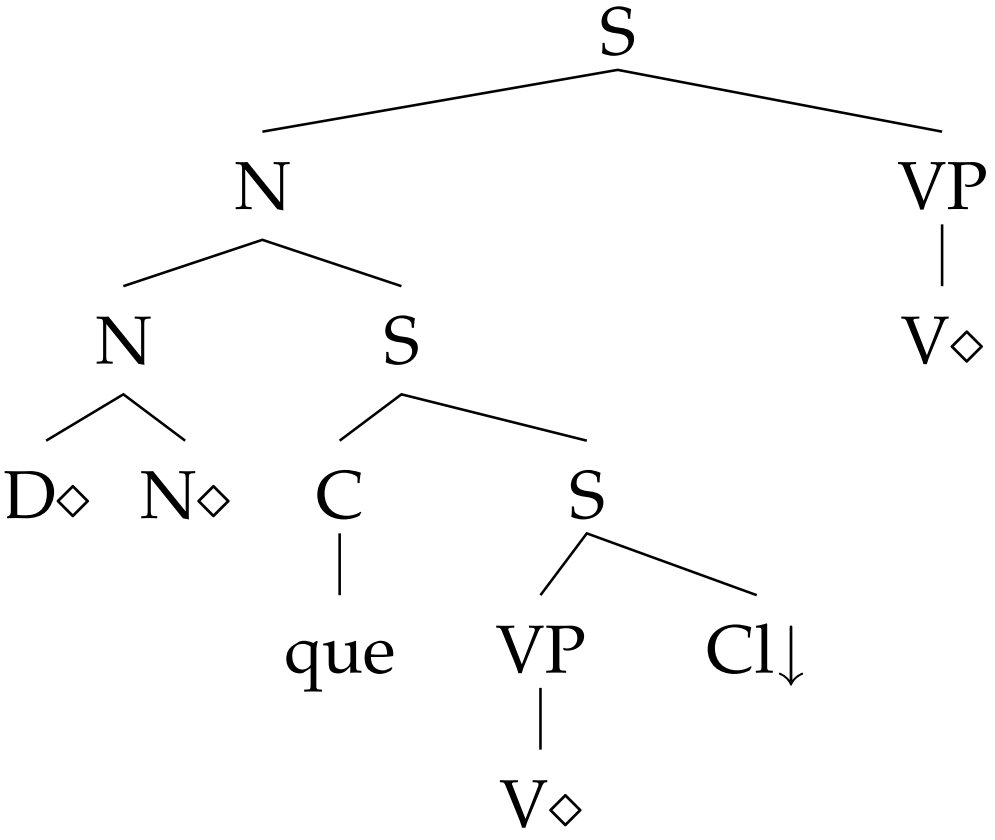
target: { *InvertedNominalSubject*, *RelativeObject* }

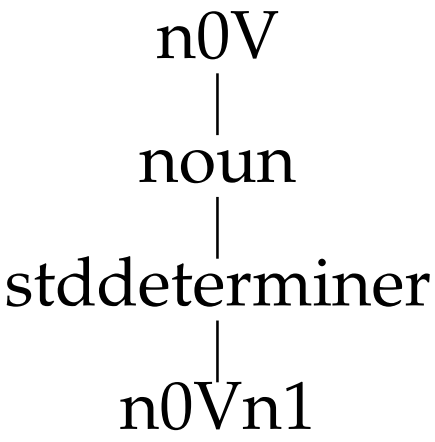




Algorithm

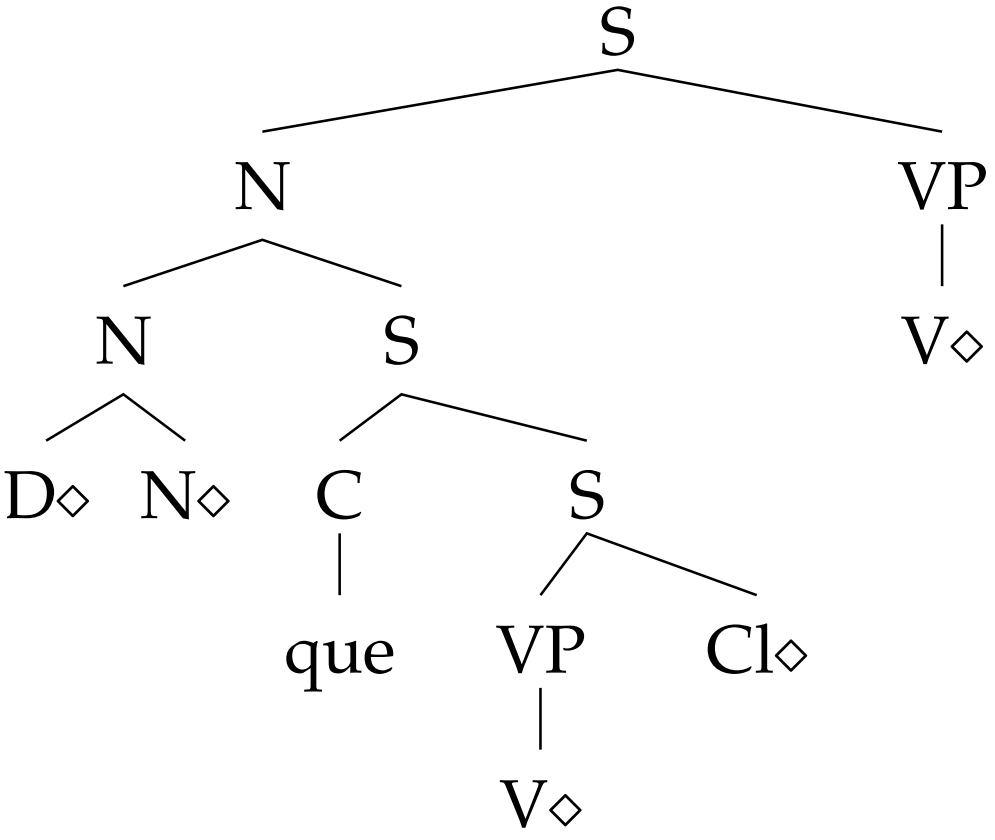
target: { }

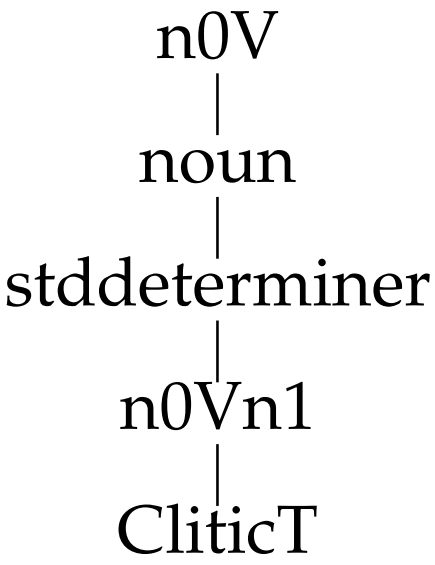




Algorithm

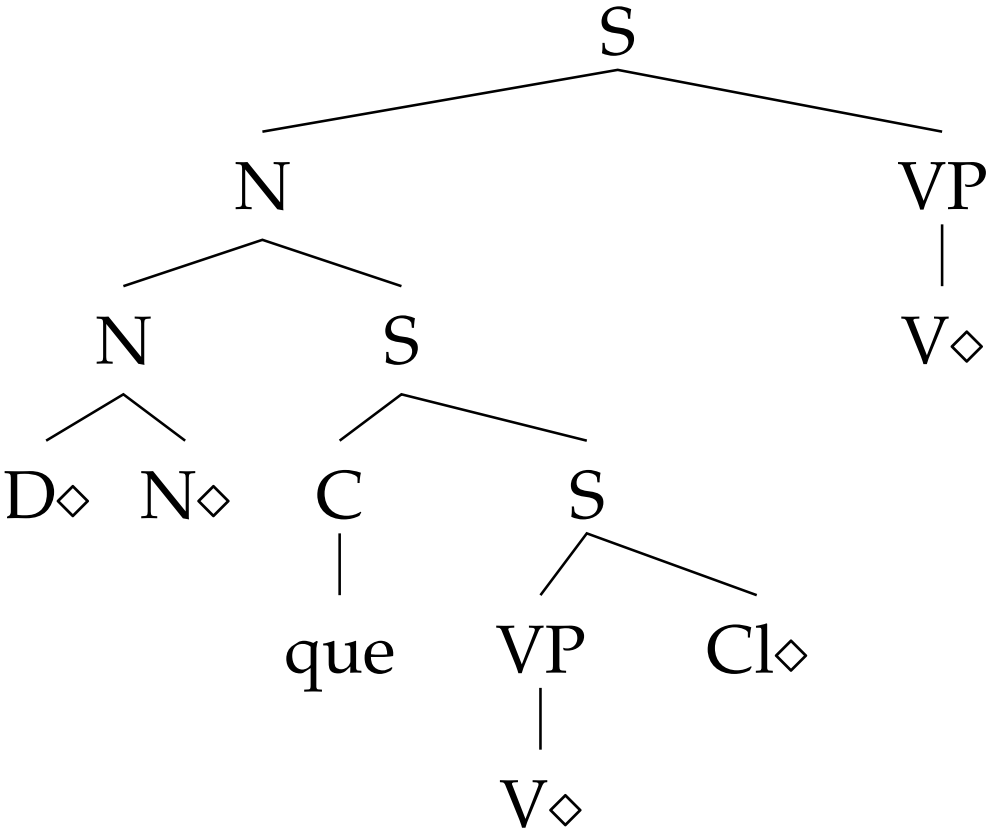
target: { }

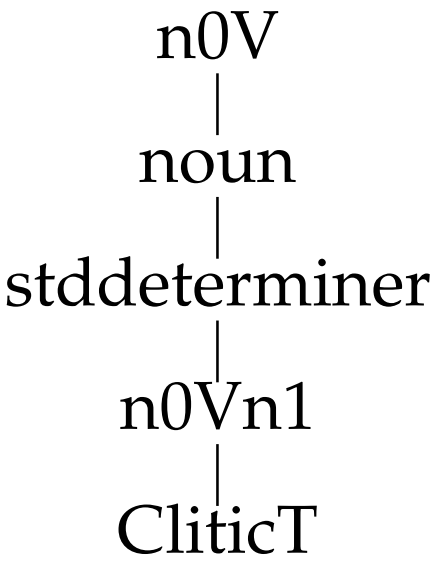




Algorithm

target: { }





Le repas que attendez vous arrive .

In Practice

- implementation (Prolog+XSLT+C)

  `svn co svn://scm.gforge.inria.fr/svn/paule/trunk/gtsg`
- grammar size
  - 5,800 trees
  - 44,000 nodes
  - allows up to 214.10^9 different feature structures
- many pre-computations
- approximation to 141 rhs variables

$$\begin{bmatrix} a : x \\ b : val \\ c : x \end{bmatrix} \equiv \begin{bmatrix} a : \_ \\ b : val \\ c : \_ \end{bmatrix}$$

More about the Algorithm

- full unification and backtracking
- no guarantee that a generated tree matches some target class:
  - drop the tree
  - change the computation environment
  - try again
- stop if no evolution

Influence of the Left-Corner Transform

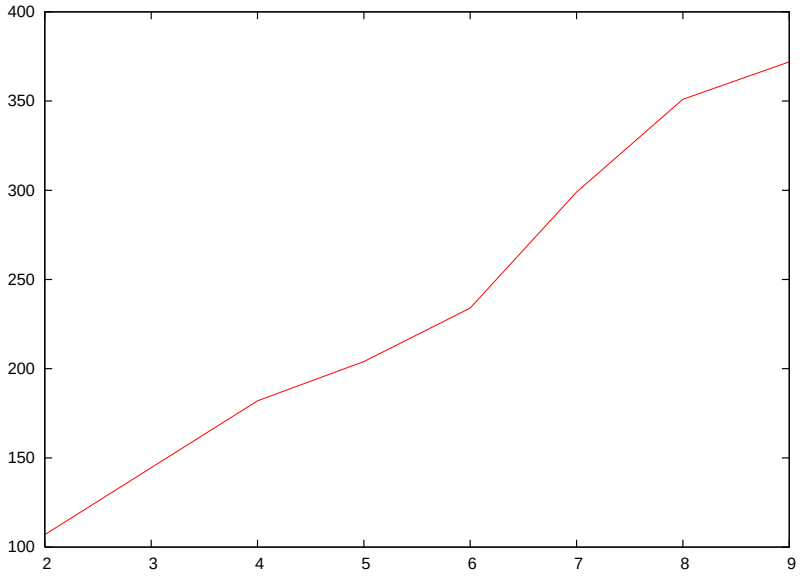
Asking for 5 times each class:

	original	left-corner
rules	5,821	6,413
classes	114	110
variables	137	138
timing	23h	1h12
out trees	190	204
coverage	84%	100%

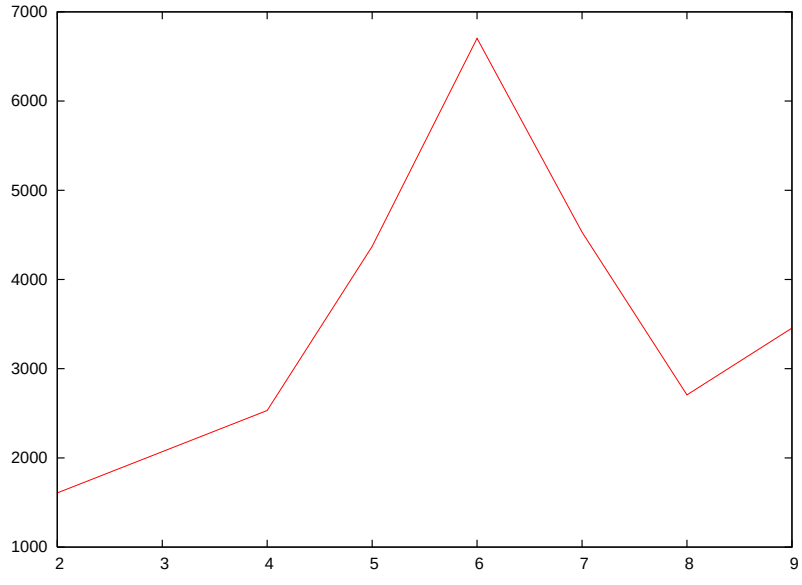
Influence of the Target Bag

Asking for n times each class, in left-corner mode:

Number of output trees



Running time (s)



Random Concluding Remarks

- application of two-level syntax
- opens new issues with error mining
- entropy measures?

J. Bos. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143, 1996. URL http://www.coli.uni-saarland.de/~bos/plu.ps.

W. S. Brainerd. Tree generating regular systems. *Information and Control*, 14(2):217–231, 1969. doi: 10.1016/S0019-9958(69)90065-5.

B. Crabbé. *Représentation informatique de grammaires fortement lexicalisées*. PhD thesis, Université Nancy 2, 2005. URL http://www.linguist.jussieu.fr/~bcrabbe/these.ps.

J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31(1):71–146, 1985. doi: 10.1016/0022-0000(85)90066-2.

C. Gardent. Integrating a unification-based semantics in a large scale lexicalised tree adjoining grammar for french. In *COLING'08*, pages 249–256, 2008. URL http://www.aclweb.org/anthology/C08-1032.

C. Gardent and E. Kow. Spotting overgeneration suspects. In *ENLG'07*, Schloss Dagstuhl, Germany, 2007. URL http://hal.inria.fr/inria-00149372/.

A. K. Joshi and Y. Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3: Beyond Words, chapter 2, pages 69–124. Springer, 1997. ISBN 3-540-60649-1. URL http://citeseer.ist.psu.edu/joshi97treeadjoining.html.

A. K. Joshi, L. S. Levy, and M. Takahashi. Tree adjunct grammars. *J. Comput. Syst. Sci.*, 10(1):136–163, 1975. doi: 10.1016/S0022-0000(75)80019-5.

S. Lehmann and S. Oepen. TSNLP—test suites for natural language processing. In *COLING'96*, pages 711–716, 1996. URL http://www.aclweb.org/anthology/C96-2120.

D. J. Rosenkrantz and P. M. Lewis II. Deterministic left corner parsing. In *11th Annual Symposium on Switching and Automata Theory*, pages 139–152. IEEE Computer Society, 1970.

B. Sagot and Éric de la Clergerie. Error mining in parsing results. In *ACL'06*, pages 329–336. ACL Press, 2006. doi: 10.3115/1220175.1220217. URL http://www.aclweb.org/anthology/P06-1042.

S. Schmitz and J. Le Roux. Feature unification in TAG derivation trees. In C. Gardent and A. Sarkar, editors, *TAG+9*, 2008. URL http://arxiv.org/abs/0804.4584.

S. M. Shieber. Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In *EACL'06*. ACL Press, 2006. ISBN 1-932432-59-0. URL http://www.aclweb.org/anthology/E06-1048.

G. van Noord. Error mining for wide-coverage grammar engineering. In *ACL'04*, pages 446–453. ACL Press, 2004. doi: 10.3115/1218955.1219012. URL http://www.aclweb.org/anthology/P04-1057.

K. Vijay-Shanker and A. K. Joshi. Feature structures based tree adjoining grammars. In *COLING'88*, pages 714–719. ACL Press, 1988. ISBN 963-8431-56-3. doi: 10.3115/991719.991783.

Experiments with Error Mining

test suite 500 trees generated using GenI [Gardent and Kow, 2007]

algorithm adaptation of Sagot and Éric de la Clergerie [2006]
expectation maximization

- bigrams instead of unigrams (capture interactions)
- derivation tree structure constraints
- more smoothing

Experiments with Error Mining

Two Rounds

Compute a suspicion rate for each bigram:

1. normalize at sentence level
2. normalize at test suite level

...and iterate.

Experiments with Error Mining

Issues: Ordering Suspects

```
Worst Form Number: 1    0.220409692766
complexAdvDeDeterminer  s0Pv1post
d7   0.220780944532
Worst Form Number: 2    0.185227187873
AdjectivalPredicativeform  s0Pv1post
d7   0.185539179236
d11  0.185187092097
```

Experiments with Error Mining

Issues: Bigrams

```
Worst Form Number: 5    0.083686136541 8
CanonicalSubject  InvertedNominalSubject
d10  0.083867096696 1
d1   0.083867096696 1
InvertedNominalSubject  RelativeObject
d10  0.083867096696 1
d1   0.083867096696 1
```