Strings to Trees to Strings*
An Introduction to Tree-adjoining Grammars

Anoop Sarkar
Simon Fraser University, Vancouver, Canada
http://bit.ly/anoop-wata2012
http://natlang.cs.sfu.ca
anoop@sfu.ca

WATA 2012, Dresden

*Title of talk “borrowed” from Prof Aravind Joshi

The Eleventh International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)

26-28 September 2012
University Paris-Diderot

http://alpage.inria.fr/tagplus11/doku.php

Abstract Submission Deadline: June 15, 2012

High Level Overview

- 1st session (90 minutes)
  1. Motivating Tree-Adjoining Grammars (TAGs) from a Computational Linguistics perspective
  2. Relation to Monadic Simple Context-free Tree Grammars
  3. Relation to algebraic notions of crossing dependencies in Dependency Grammar

High Level Overview

- 2nd session (90 minutes)
  1. TAGs as a formalism for natural language syntax
  2. Statistical Parsing with weighted TAG
  3. Synchronous Tree-adjoining grammar

Preliminaries

Sentences as Strings

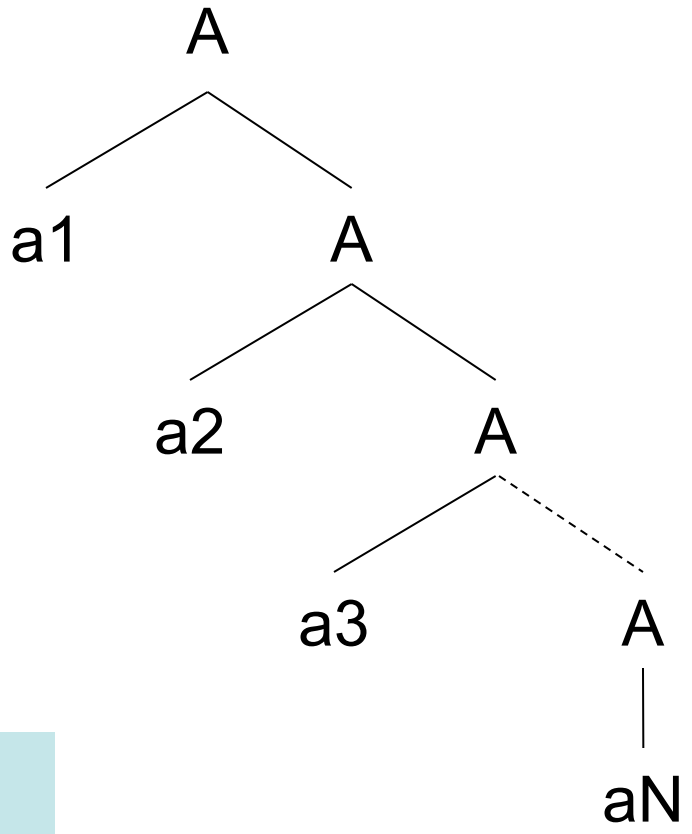
David	likes		peanuts	
Noun	Verb		Noun	
David	said	that	Mary	left
Noun	Verb	Comp	Noun	Verb

- Linear order: all important information is contained in the precedence information, e.g. useful "feature functions" are w-2, w-1, t-2, t-1, w0, w+1, w+2, t+2, t+1, etc.
- No hierarchical structure but every part-of-speech is lexicalized, e.g. Verb is lexicalized by likes
- Language (set of strings) generated by finite-state grammars

Finite State Grammars

$A \rightarrow a\ A$
$A \rightarrow a$

$A1 \Rightarrow a1\ A2$
$\Rightarrow a1\ a2\ A3$
$\Rightarrow a1\ a2\ a3\ A4$
$\Rightarrow a1\ a2\ ...\ aN$



Terminal symbol: a
Non-terminal symbol: A

Context-Free Grammars

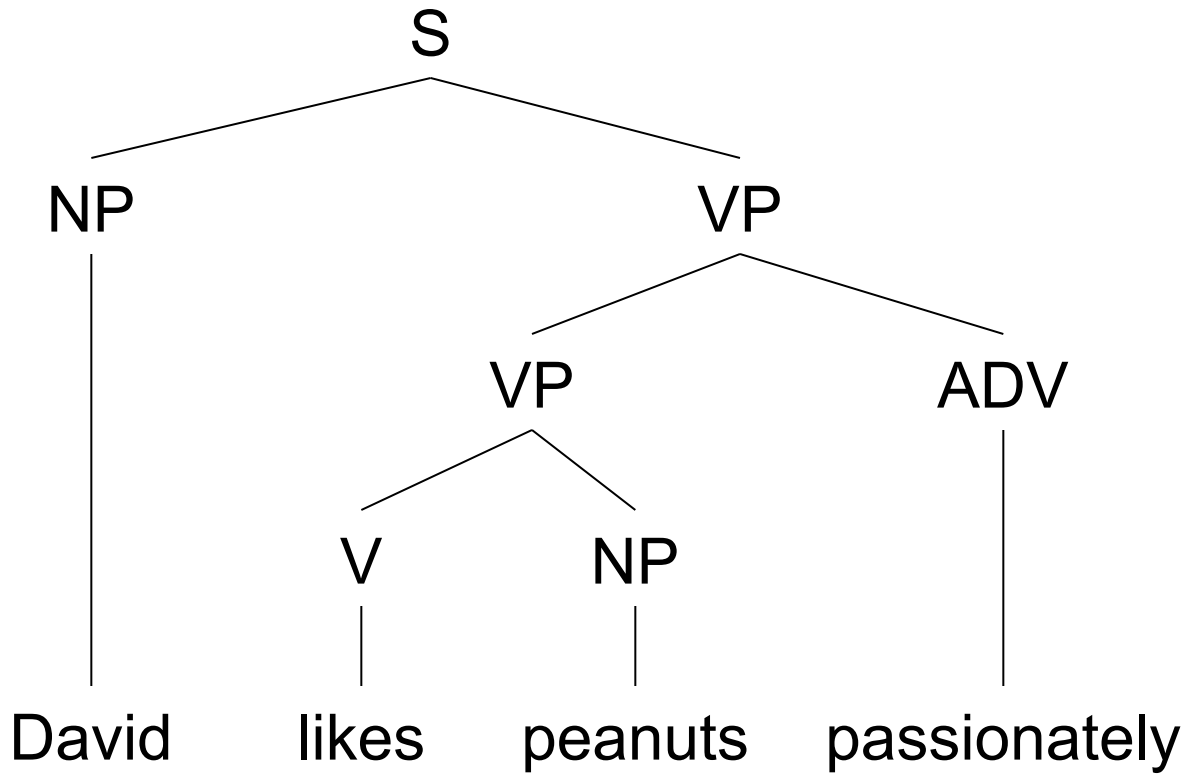
S → NP VP
VP → V NP | VP ADV
NP → David | peanuts
V → likes
ADV → passionately

- CFGs generate strings, e.g. language of G above is the set:

  { David likes peanuts,
  David likes peanuts passionately,
  ... }

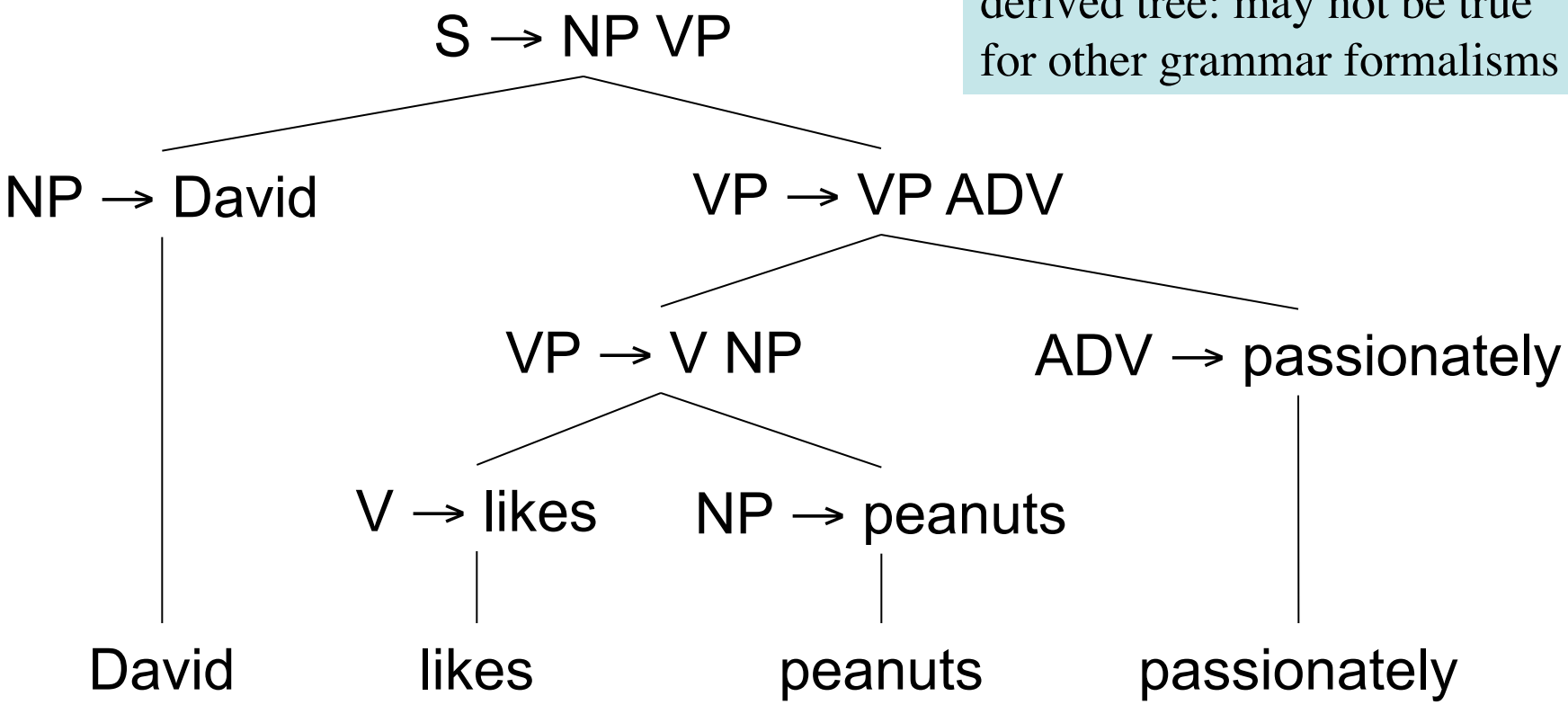
- Lexical sensitivity is lost
- CFGs also generate trees: hierarchical structure produced is non-trivial

CFG: Derived/Parse Tree



CFG: Derivation Tree

derivation tree is identical to derived tree: may not be true for other grammar formalisms



Preliminaries

- Rules of the kind $\alpha \rightarrow \beta$ where α, β are strings of terminals and non-terminals
- Chomsky hierarchy: regular, context-free, context-sensitive, recursively enumerable
- Automata: finite-state, pushdown, LBA, Turing machines (analysis of complexity of parsing)
- A rule $\alpha \rightarrow \beta$ in a grammar is lexicalized if β contains a terminal symbol
- Lexicalization is a useful property, e.g. a rule like NP $\rightarrow$ NP creates infinite valid derivations

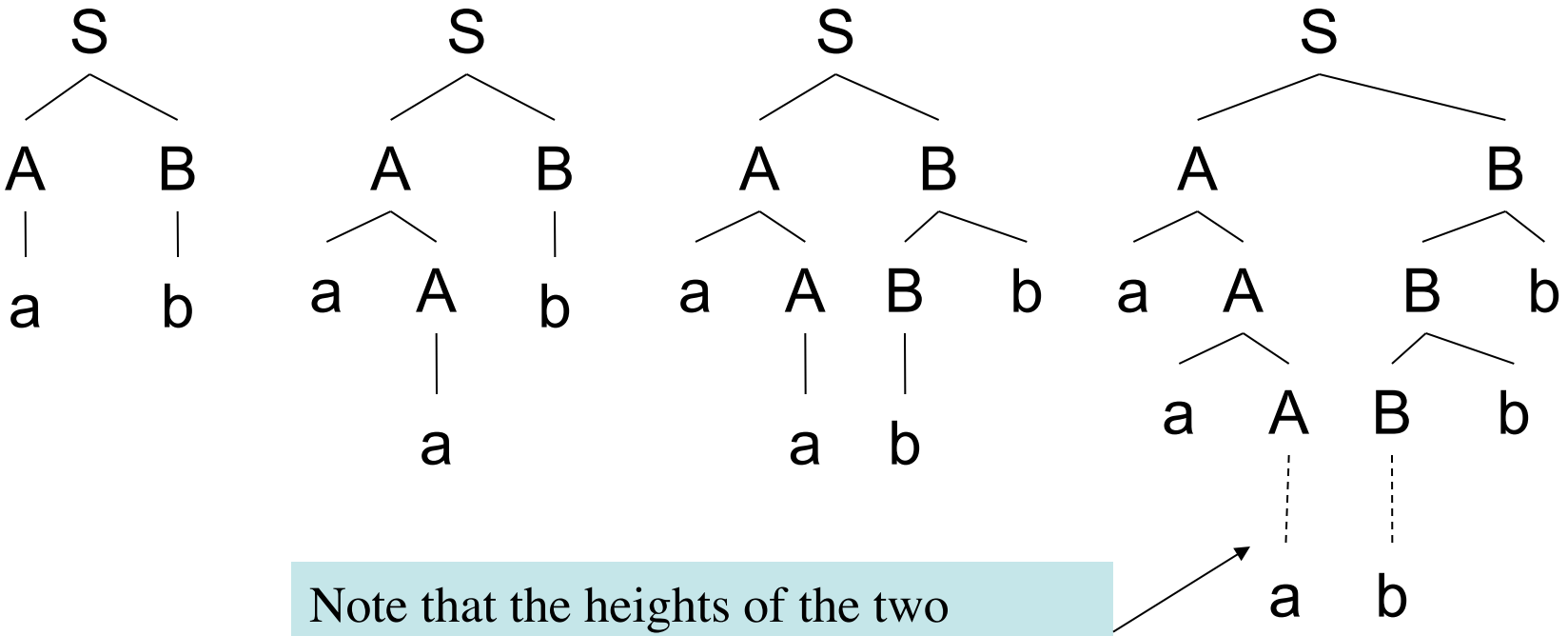
Strong vs. Weak Generative Capacity

- A property of a formal grammar, e.g. of a regular grammar or a CFG
- **Weak Generative Capacity** of a grammar is the set of strings or the *string language*
- **Strong Generative Capacity** of a grammar is the set of structures (usually the set of trees) produced by the grammar or the *tree language*

Tree Languages

$S \rightarrow A\ B$
$A \rightarrow a\ A \mid a$
$B \rightarrow B\ b \mid b$

This grammar generates the tree language informally shown below

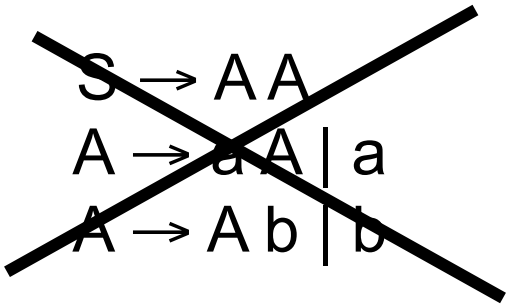


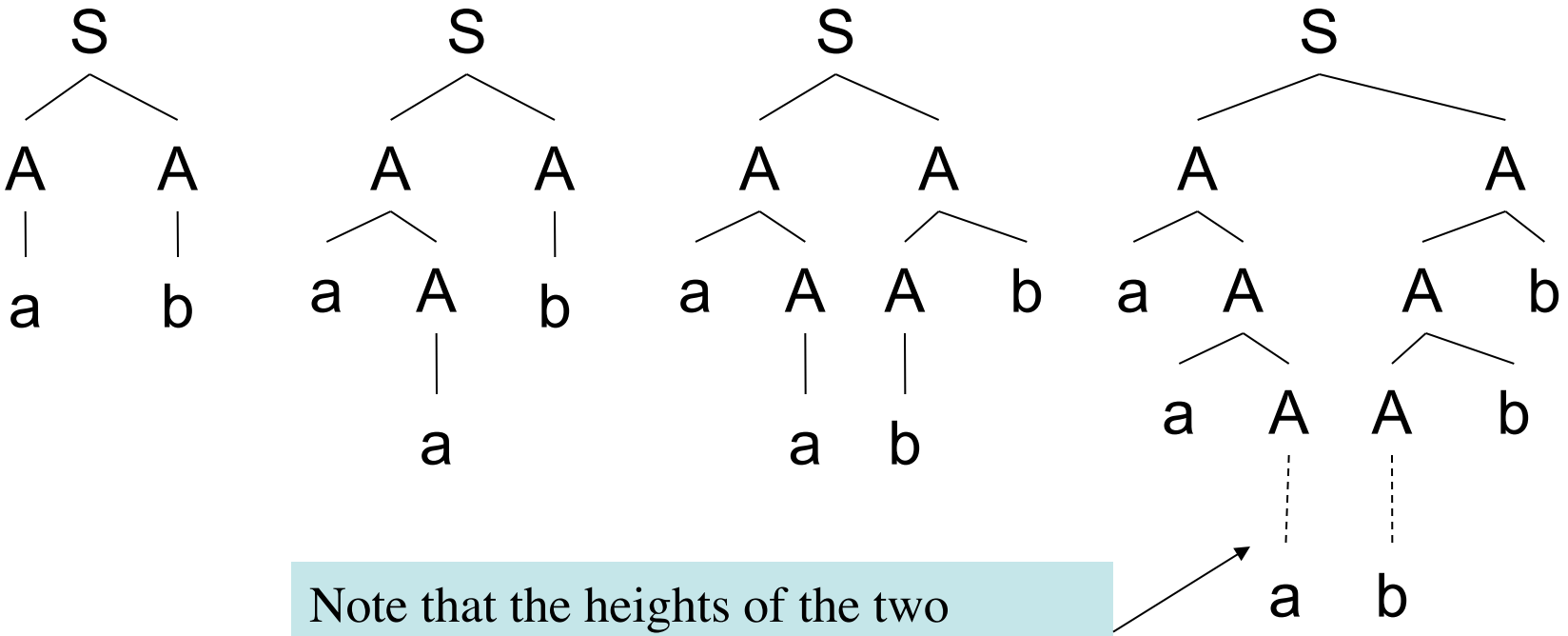
Note that the heights of the two branches do not have to be equal

Grammars that generate trees

A Tree Language with no CFG

Claim: There is no CFG that can produce the tree lang below:





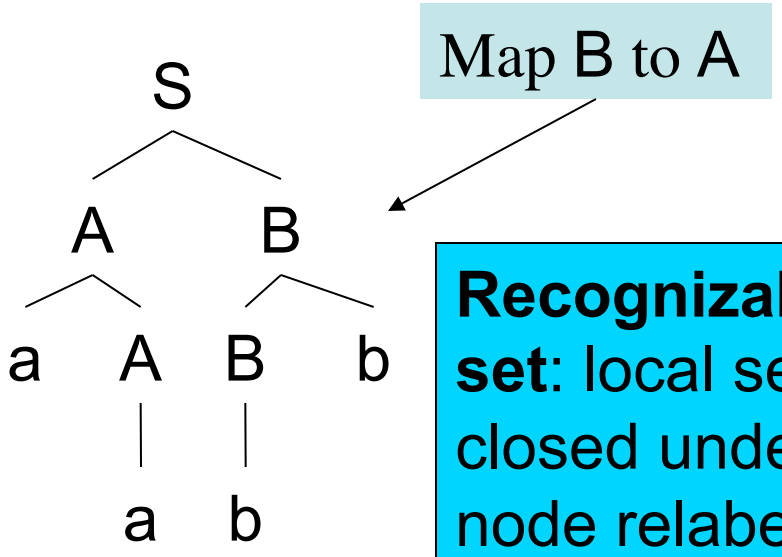
Note that the heights of the two branches do not have to be equal

Grammars for Tree Languages

- A simple trick: start with a CFG that almost works
- Then re-label the node labels, map B to A to get the desired tree set
- But how can we directly generate the tree sets?
- We need a **generative device** that generates *trees*, not *strings*
- (Thatcher, 1967) (Brainerd, 1969) and (Rounds, 1970) provided such a generative device

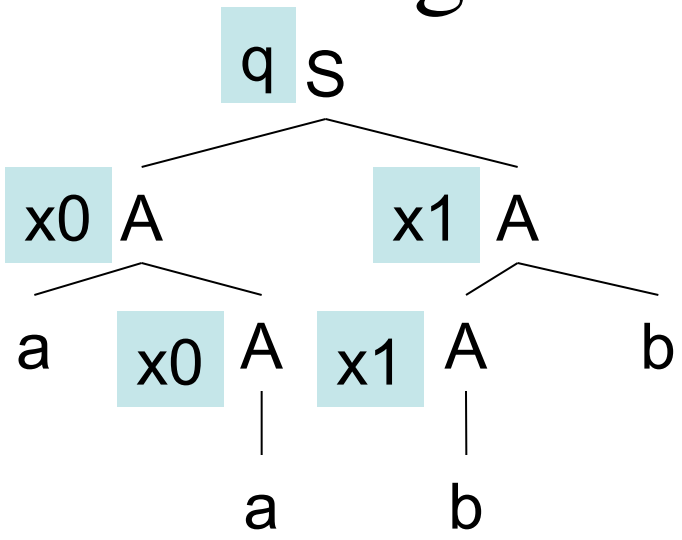
$S \rightarrow A\ B$
$A \rightarrow a\ A \mid a$
$B \rightarrow B\ b \mid b$

Local set: tree sets from CFGs



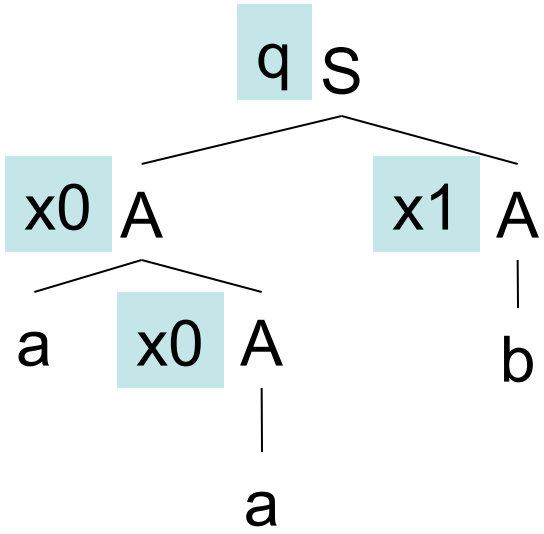
Recognizable set: local set closed under node relabeling

Regular Tree Grammars



start state: q
q → S(x0 x1)
x0 → A(a x0)
x1 → A(x1 b)
x0 → A(a)
x1 → A(b)

note: rhs can be a tree of any size!



- RTGs = Top-down tree automata
- **Can generate infinite tree sets**
- **Found useful in syntax-based statistical machine translation** (May & Knight, 2006)

Regular Tree Grammars

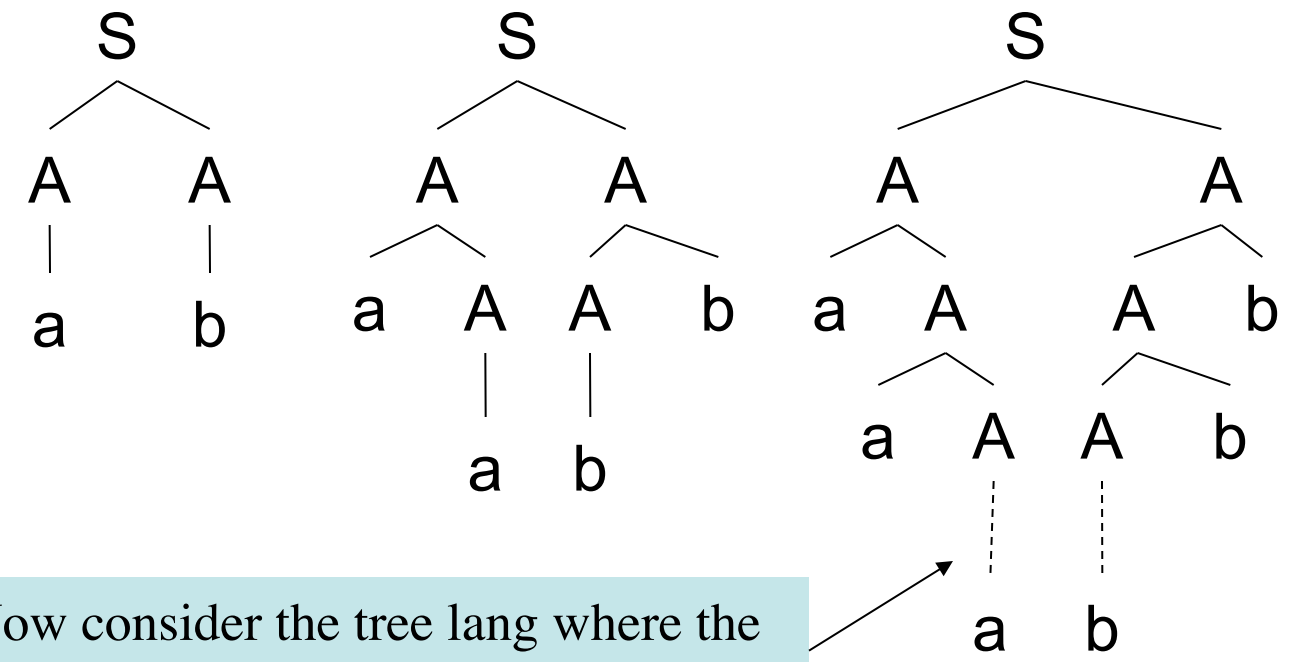
- RTGs generate tree languages
- The yield of each tree in this language produces a string
- *yield*(RTG) provides a string language
- For each RTG: *yield*(RTG) = CFL
- But the set of tree languages of CFGs is contained within that of RTGs

A Tree Language with no RTG

Claim: There is no RTG that can produce the tree language below:

~~q → S(x0 x1)~~
~~x0 → A(a x0) | A(a)~~
~~x1 → A(x1 b) | A(b)~~

RTG is like a finite-state machine, the state cannot count how many times it was reached

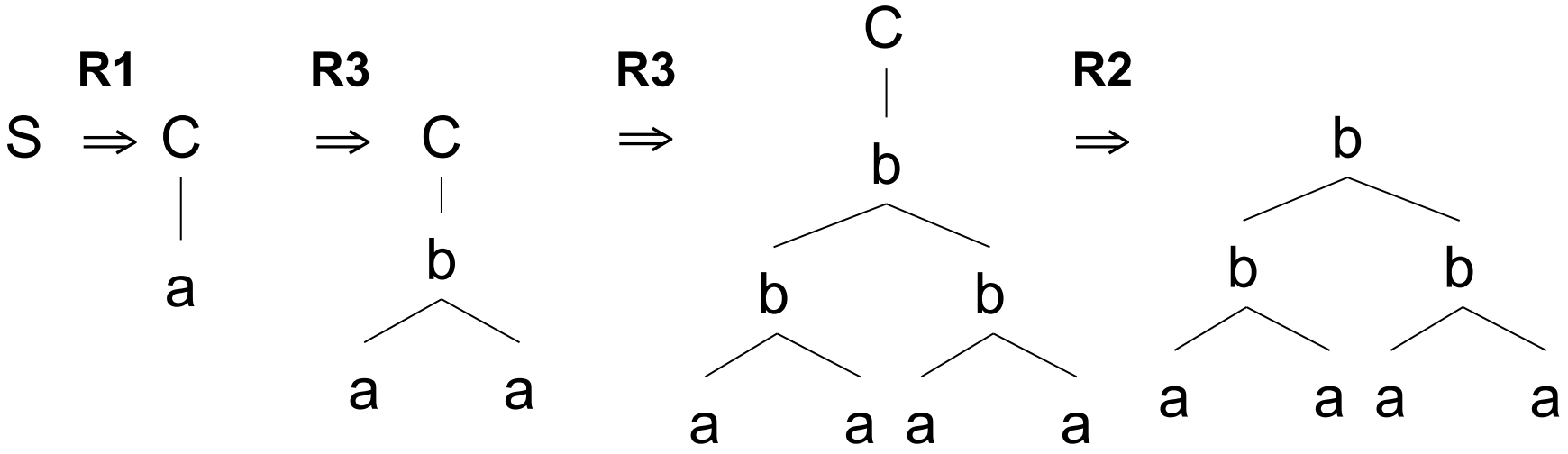


Now consider the tree lang where the depth of the two branches is **equal**

(Rounds 1970)

Context-free Tree Languages

R1: S → C(a)
R2: C(x1) → x1
R3: C(x1) → C(b(x1 x1))



String language $= \{a^{2^n} \mid n \geq 0\}$

Context-free Tree Languages

- *yield*(CFTLs) = Indexed Languages (Fischer, 1968)
- Indexed languages: does not have the constant growth property
- Also, recognition algorithm is NP-complete (Rounds, 1973)
- Perhaps there is a tree grammar formalism between RTG and CFTG?
- How much context-sensitivity over RTGs should this tree grammar have?

Motivation #1

Context-sensitive predicates on trees bear less fruit than you think*

* borrowed from a title of a paper by A. Joshi

Tree Languages: Another Example

A more practical example

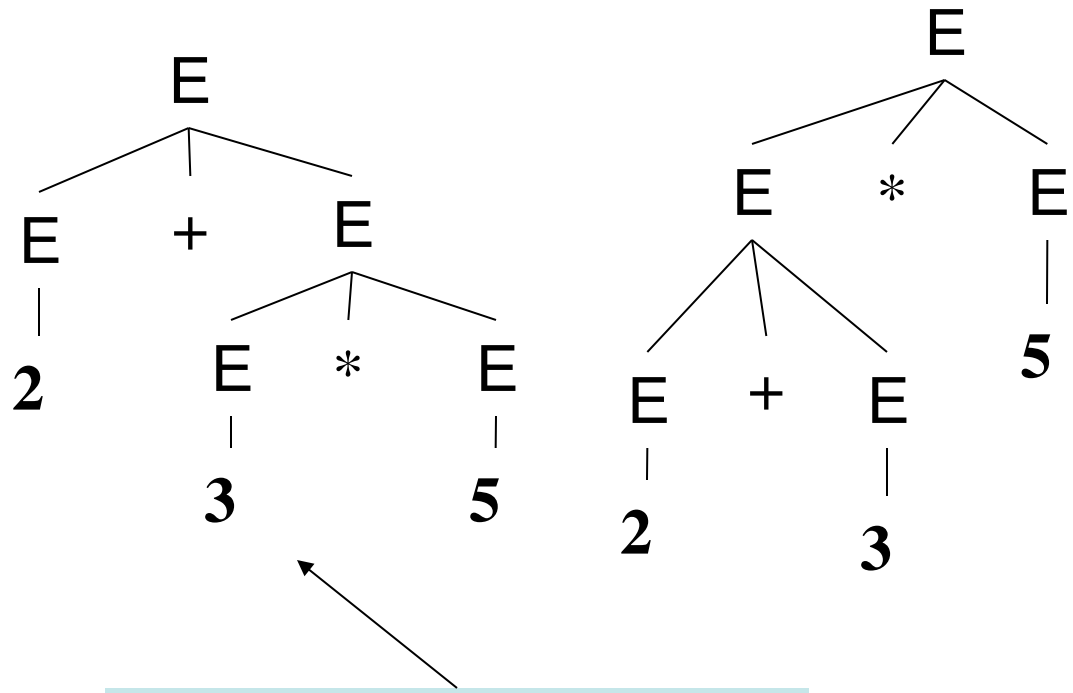
$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow \mathbf{N}$

2+3*5 is ambiguous either **17** or **25**



Ambiguity resolution: * has precedence over + cannot use RTGs!

Tree Languages: Context-sensitivity

Eliminating ambiguity

$E \rightarrow E + E$

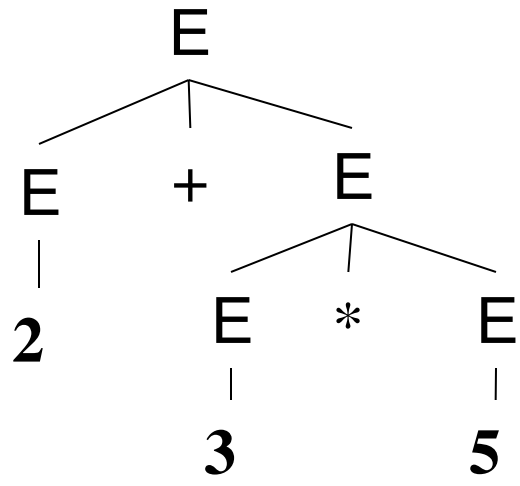
$\neg(+\_\_) \wedge \neg(*\_\_) \wedge \neg(\_\_*)$

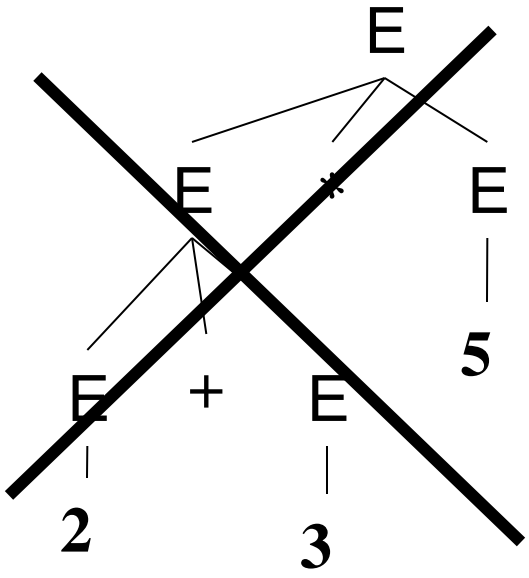
$E \rightarrow E * E$

$\neg(*\_\_)$

$E \rightarrow (E)$

$E \rightarrow \mathbf{N}$





similar to context-sensitive grammars!

Context-sensitive Grammars

- Rules of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$ where γ cannot be the empty string, also written as $A \rightarrow \gamma \; / \; \alpha \_\_ \beta$
- CSGs are very powerful: they can generate languages like $\{ 1^p : p \text{ is prime} \}$
- This kind of computational power is unlikely to be needed to describe natural languages
- Like other grammar formalisms in the Chomsky hierarchy CSGs generate string sets
- What if they are used to recognize tree sets?

Context-sensitive Grammars

1: S → S B C
2: S → a C
3: a B → a a
4: C B → B C
5: B a → a a
6: C → b

S ⇒ **S B C** (1)
⇒ **S B C** B C (1)
⇒ **a C** B C B C (2)
⇒ a **B C** C B C (4)
⇒ **a a** C C B C (3)
⇒ a a C **B C** C (4)
⇒ a a **B C** C C (4)
⇒ a **a a** C C C (3)
⇒ a a a **b** C C (6)
⇒ a a a b **b** C (6)
⇒ a a a b b **b** (6)

Context-sensitive predicates

- Consider each CSG rule $A \rightarrow \gamma / \alpha\_\_\beta$ to be a predicate (i.e. either true or false)
- Apply all the rules in a CSG as predicates on an input tree
- If all predicates are true then *accept* the tree, else *reject* the tree
- Can be easily extended to a set of trees and used to accept a tree set
- Can we precisely describe this set of tree languages?

Peters-Ritchie Theorem

- The Peters-Ritchie Theorem (Peters & Ritchie, 1967) states a surprising result about the generative power of CSG predicates
- Consider each tree set accepted by CSG predicates
- Theorem: The string language of this tree set is a context-free language
- Each CSG when applied as a set of predicates can be converted into a weakly equivalent CFG
- See also: (McCawley, 1967) (Joshi, Levy & Yueh, 1972) (Rogers, 1997)

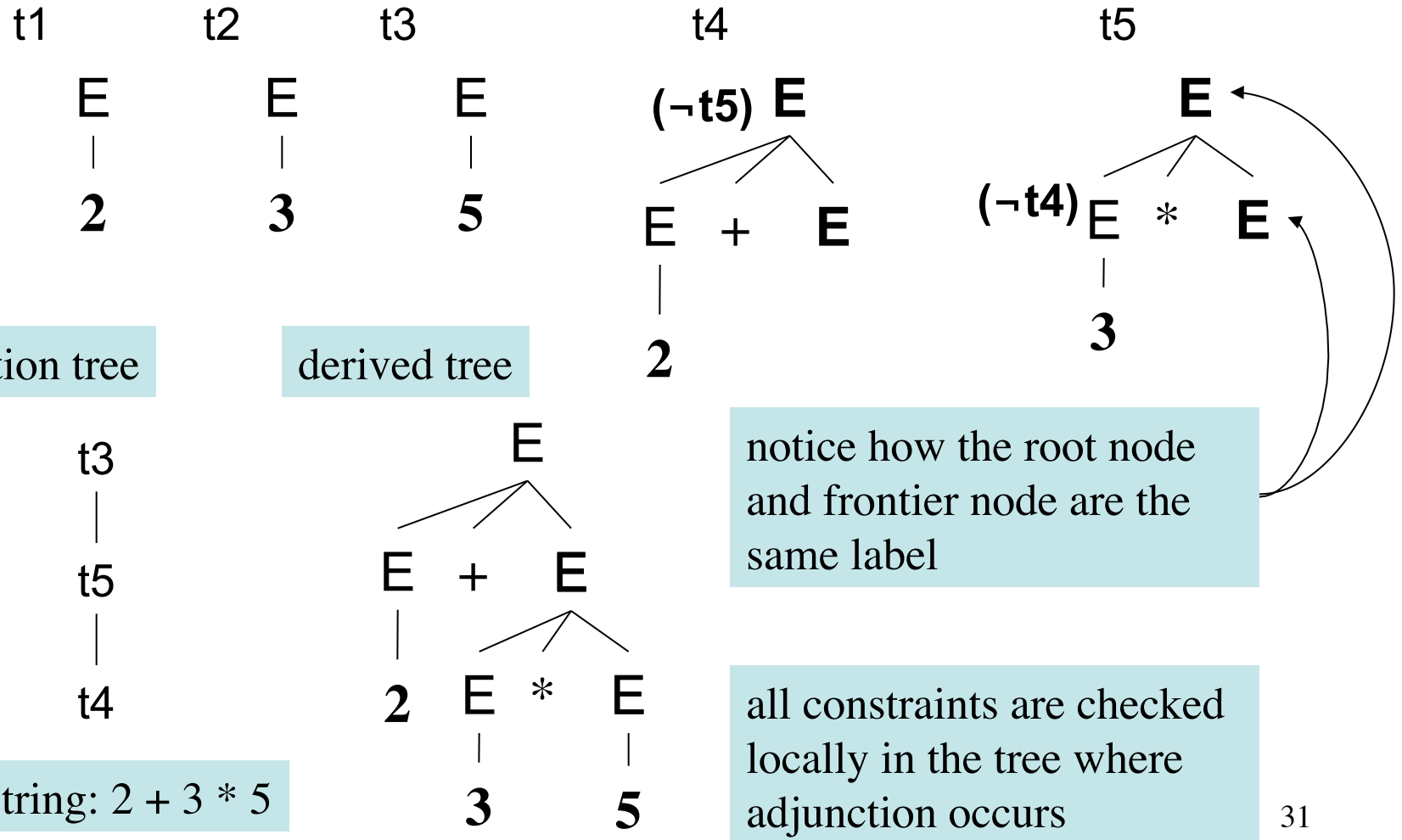
Local Transformations

- This theorem was extended by (Joshi & Levy, 1977) to handle arbitrary boolean combinations and sub-tree / domination predicates
- Proof involves conversion of all CSG predicates into top-down tree automata that accept tree sets
- (Joshi & Levy, 1977) showed transformations used in transformational grammar can be written in this way
- Important caveat: we assume some source GEN generating trees which are then validated. (connection to Optimality Theory)

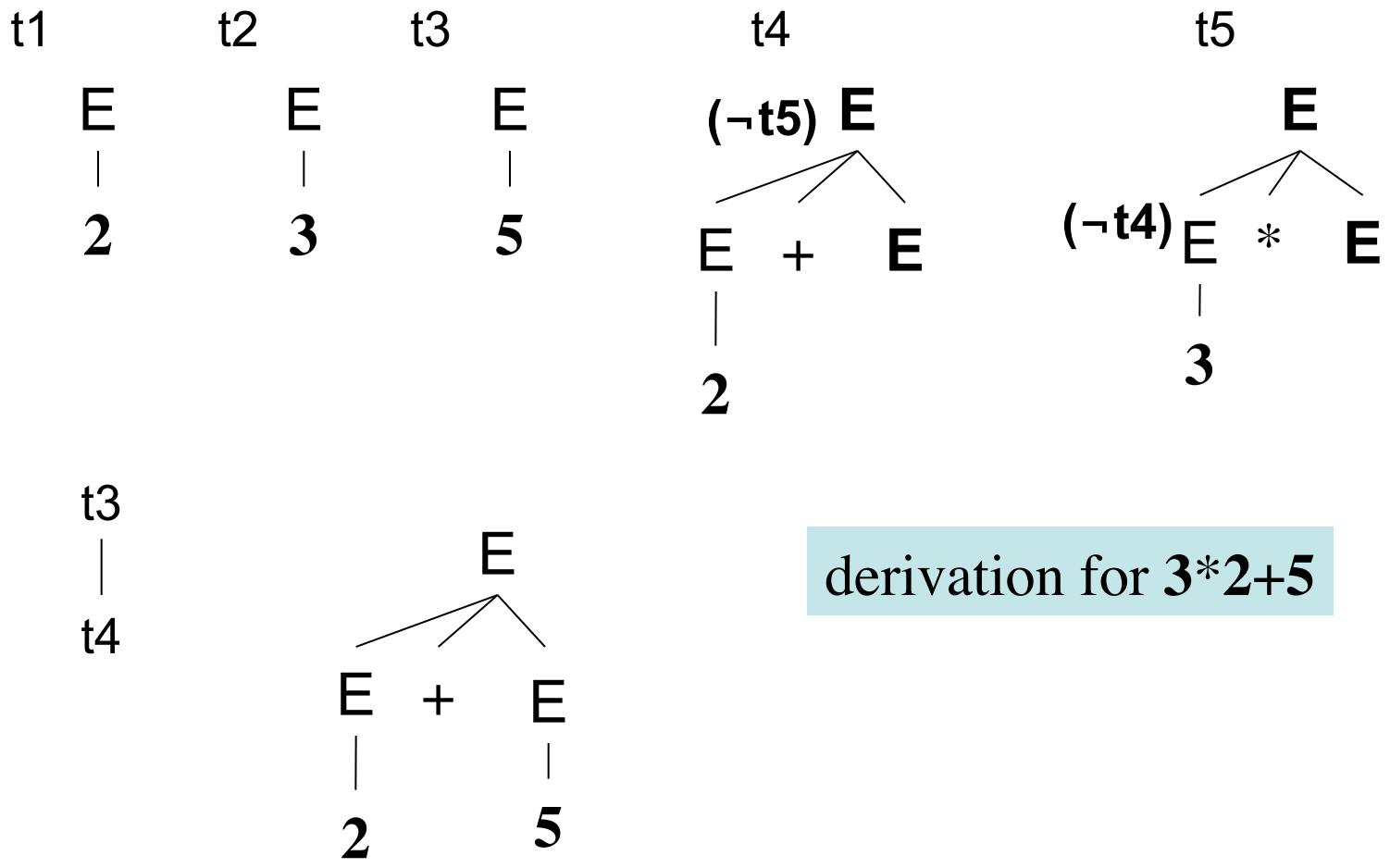
Tree-Adjoining Grammars

- Construct a tree set out of tree fragments
- Each fragment contains only the structure needed to express the locality of various CSG predicates
- Each tree fragment is called an elementary tree
- In general we need to expand even those non-terminals that are not leaf nodes: leads to the notion of adjunction

Tree-Adjoining Grammars

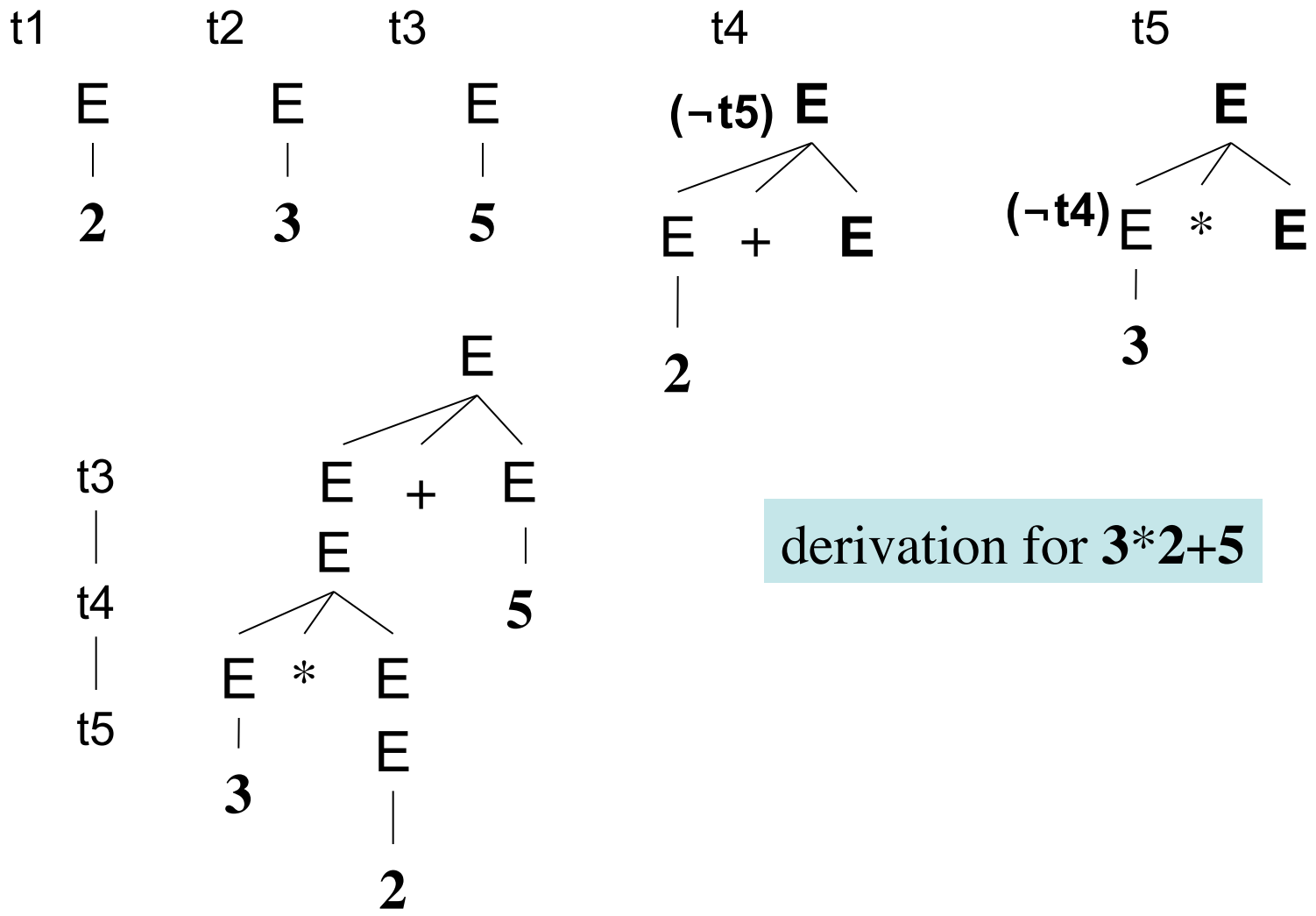


Tree-Adjoining Grammars



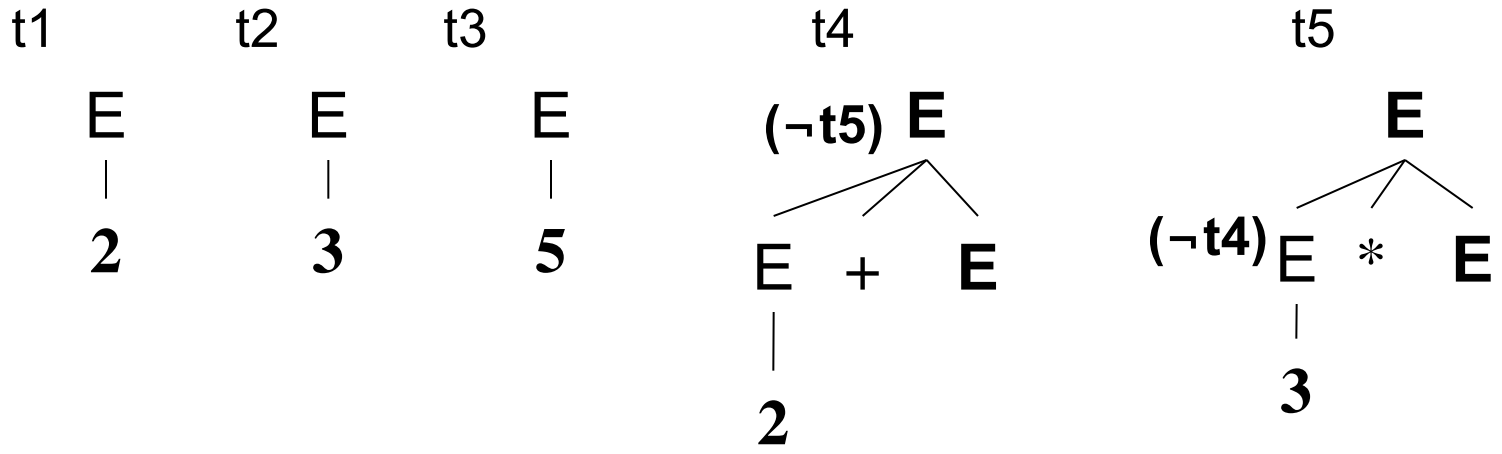
derivation for **3*2+5**

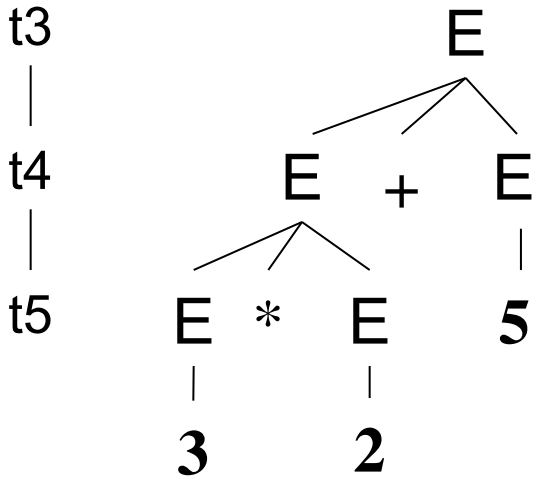
Tree-Adjoining Grammars



derivation for **3*2+5**

Tree-Adjoining Grammars





derivation for **3*2+5**

Tree-Adjoining Grammars

- A TAG G = (N, T, I, A, S) where
  - N is the set of non-terminal symbols
  - T is the set of terminal symbols
  - I is the set of initial or non-recursive trees built from N, T and domination predicates
  - A is the set of recursive trees: one leaf node is a non-terminal with same label as the root node
  - S is set of start trees (has to be initial)
  - I and A together are called *elementary trees*

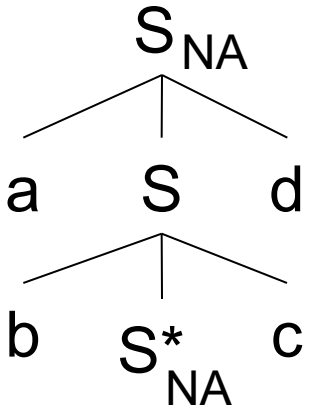
Adjunction Constraints

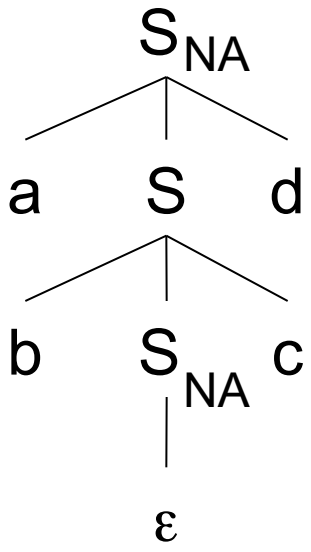
- Adjunction is the rewriting of a non-terminal in a tree with an auxiliary tree
- We can think of this operation as being "context-free"
- Constraints are essential to control adjunction: both in practice for NL syntax and for formal closure properties

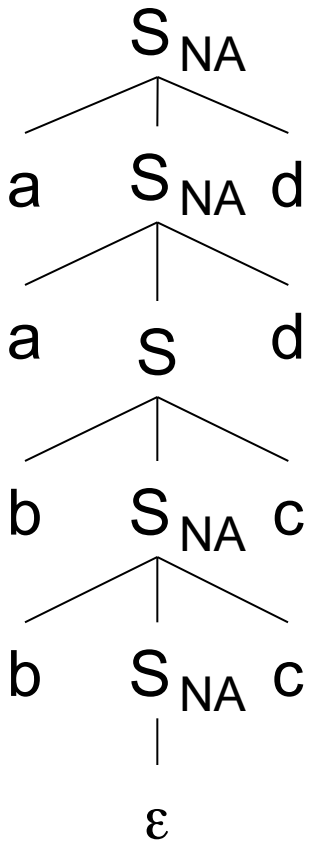
Adjunction Constraints

- Three types of constraints:
  - null adjunction (NA): no adjunction allowed at a node
  - obligatory adjunction (OA): adjunction must occur at a node
  - selective adjunction (SA): adjunction of a pre-specified set of trees can occur at a node

Adjunction Constraints





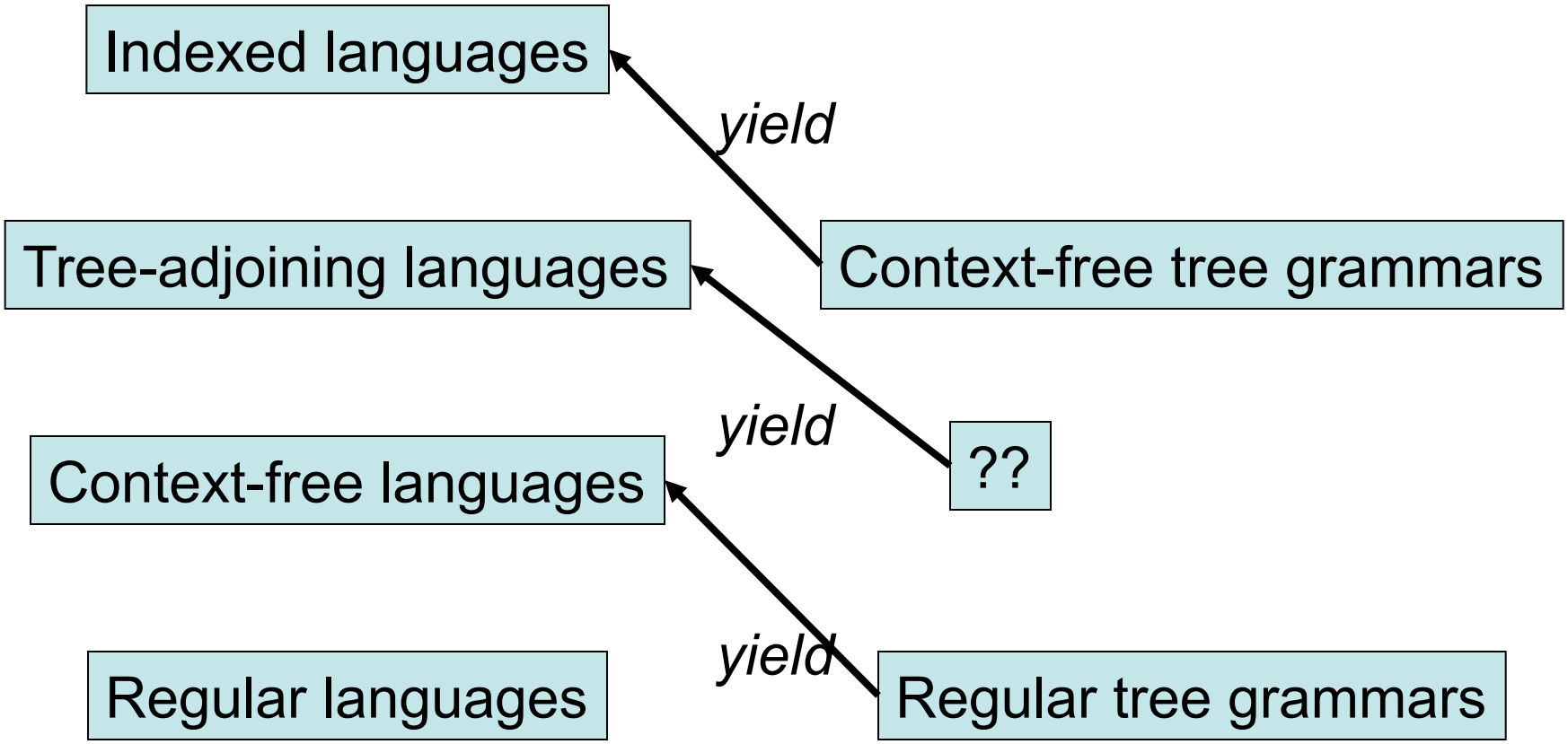


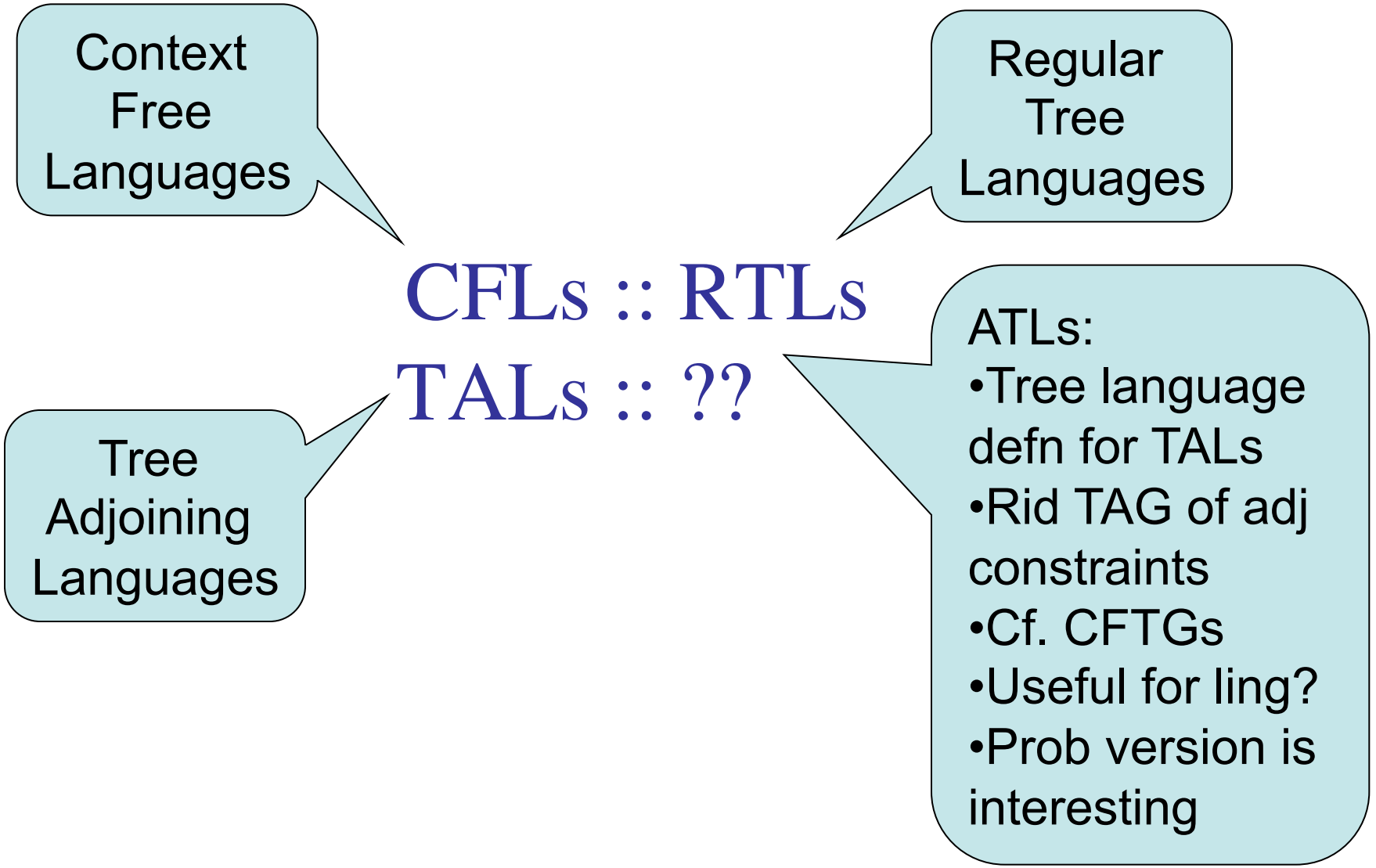
This TAG can generate the language
$L = \{ a^n b^n c^n d^n : n \geq 1 \}$
Note that the OA & NA constraints are crucial to obtain the correct language

Tractable Descriptions

- Why not use context-sensitive grammars?
- For G, given a string *x* what is the complexity of an algorithm for the question: is *x* in L(G)?
  - Unrestricted Grammars/Turing machines: undecidable
  - Context-sensitive: NSPACE[n] linear non-deterministic space
  - Indexed Grammars: NP-complete
  - Tree-Adjoining Grammars: $O(n^6)$
  - Context-free: $O(n^3)$
  - Regular: O(n)

Connections with Context-free tree grammars

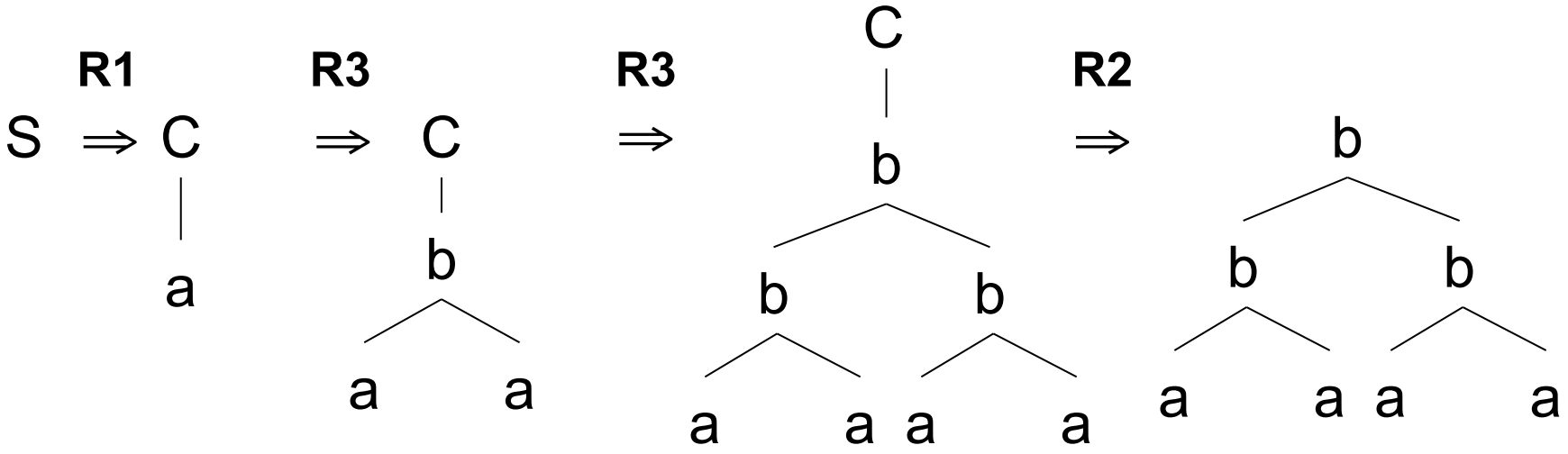

Indexed languages
yield
Context-free tree grammars
Tree-adjoining languages
yield
??
Context-free languages
yield
Regular languages
Regular tree grammars
Strings
Trees


Context Free Languages
Regular Tree Languages
CFLs :: RTLs
TALs :: ??
Tree Adjoining Languages
ATLs:
•Tree language defn for TALs
•Rid TAG of adj constraints
•Cf. CFTGs
•Useful for ling?
•Prob version is interesting

(Rounds 1970)

Context-free Tree Languages

R1: S → C(a)
R2: C(x1) → x1
R3: C(x1) → C(b(x1 x1))



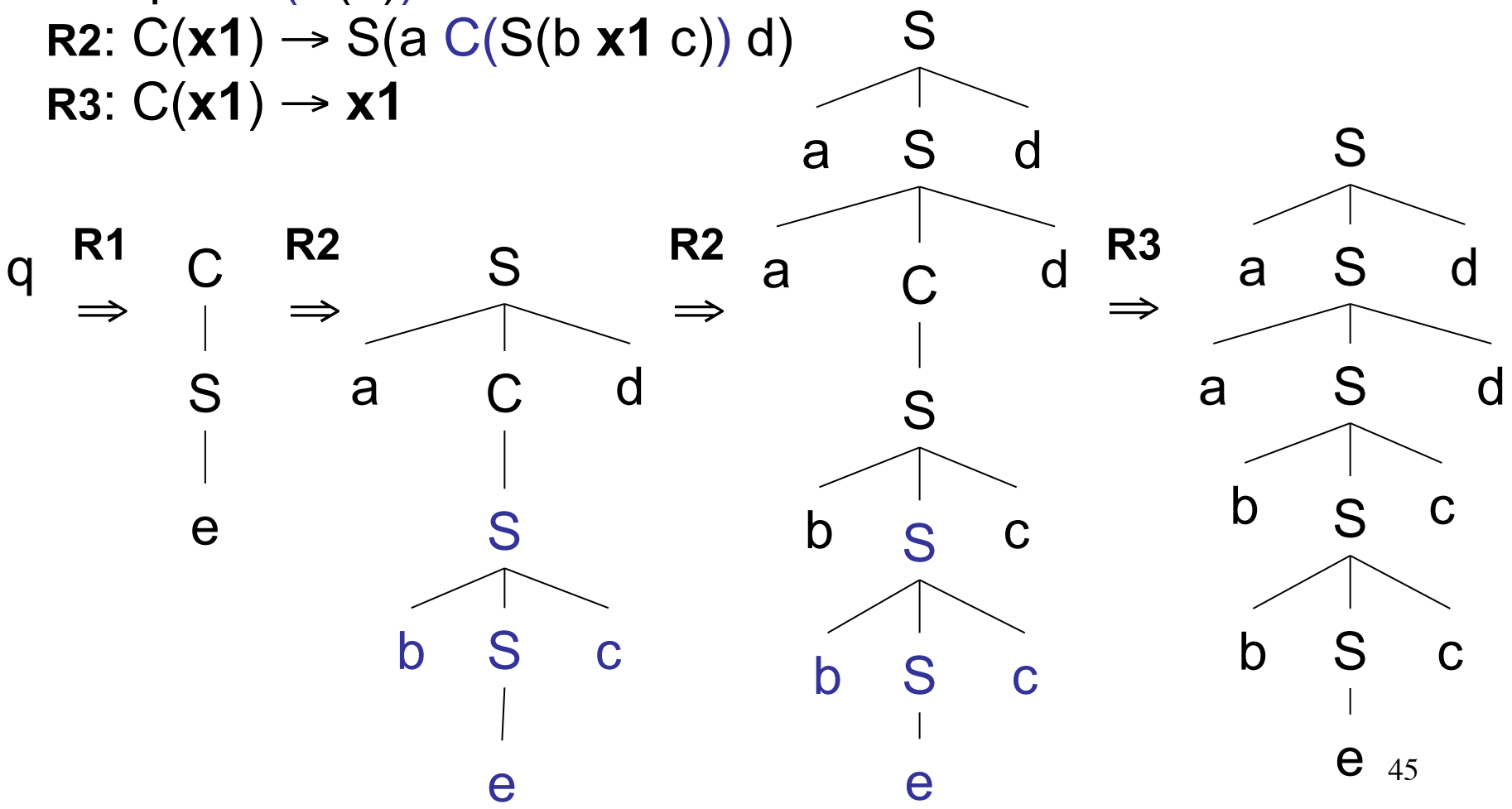
String language $= \{a^{2^n} \mid n \geq 0\}$

Modifying Context-free Tree Grammars

- Simple CFTG = linear and non-deleting
- Linear = tree variables shalt not multiply
- Non-deleting = tree variables shalt not be matched on the lhs and dropped in the rhs
- The non-deleting condition can be dropped (Fujiyoshi, 2005)
- Monadic CFTG = only one subtree can be matched on the lhs of any rule, $A(x) \rightarrow T$

Monadic Simple Context-free Tree Languages

R1: q → C(S(e))
R2: C(**x1**) → S(a C(S(b **x1** c)) d)
R3: C(**x1**) → **x1**



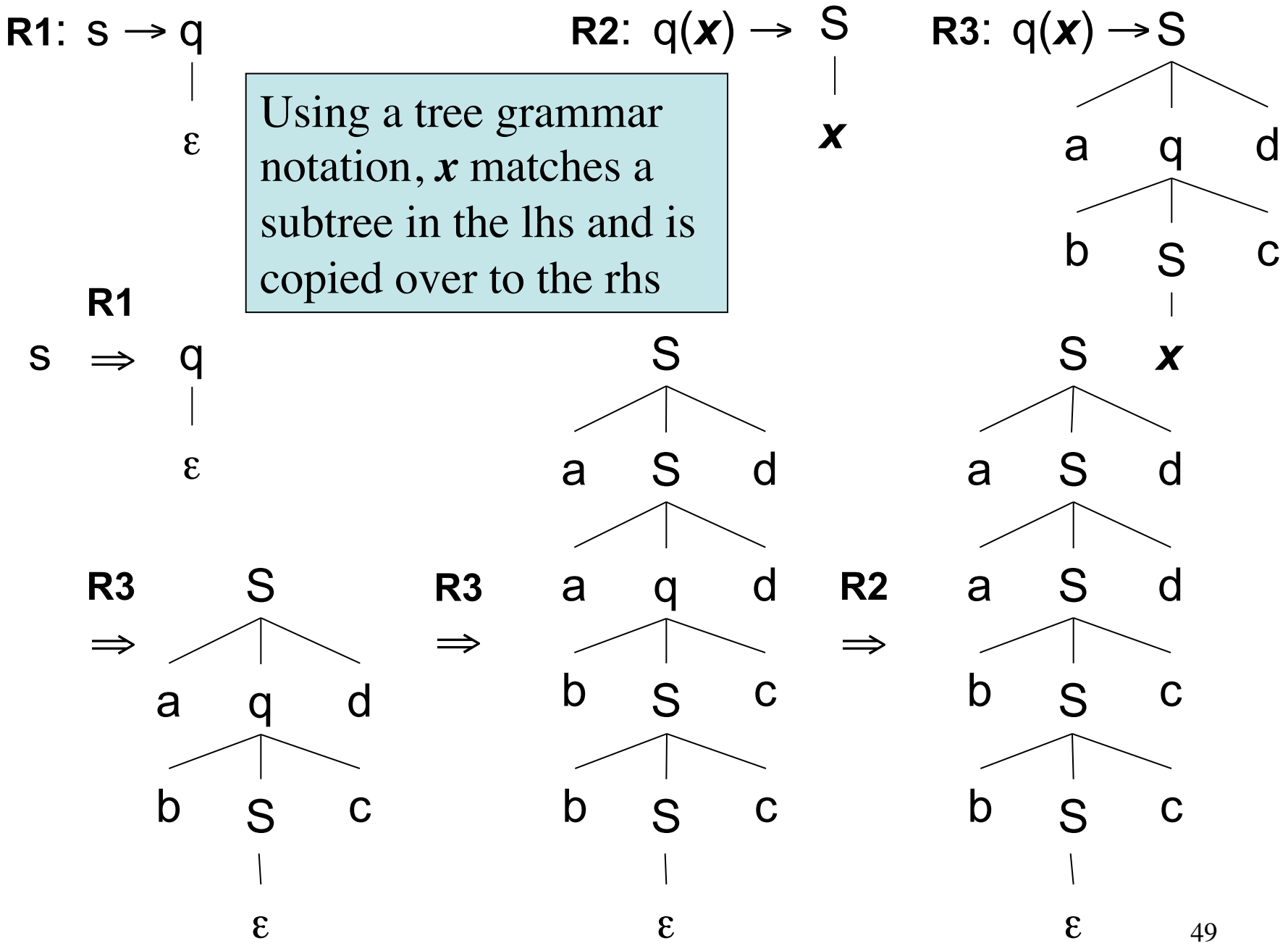
Monadic Simple CFTGs

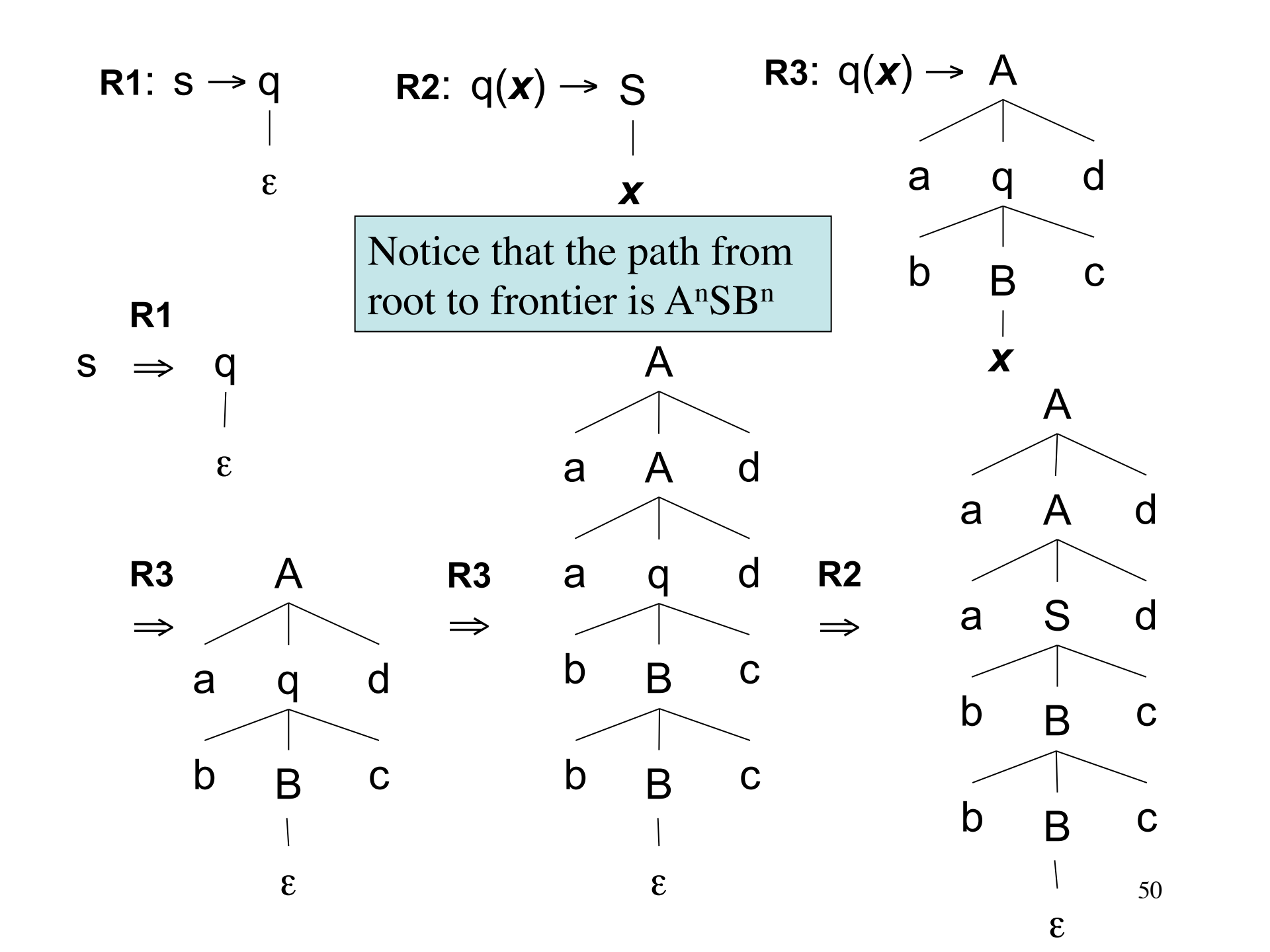
- Tree language of TAGs is contained within *monadic simple CFTGs*
- TAGs are weakly equivalent to CFTGs (Fujiyoshi & Kasai, 2000; Mönnich 1997)
- Focus of this talk: how about extending RTGs instead? (Lang, 1994)
- Another way to limit CFTGs is the so-called *spinal form CFTG* (Fujiyoshi & Kasai, 2000)

From Trees to Strings

Adjoining Tree Grammars

- ATG is a tree grammar formalism over strings
- Rules are of the form $q(x) \rightarrow T$; q is a state, x is tree variable, T is a tree
- The rhs tree T is built with terminals a, b, … and non-terminals A, B, …
- Tree T can also contain tree variables which can be internal nodes dominating a single subtree (unlike RTGs where they occur on the frontier)
- Finally, ATGs have a start tree variable
- An ATG is well-formed if for every sentential form w ($q \Rightarrow^* w$) is a well-formed tree.


R1: s → q
ε
R2: q(x) → S
x
R3: q(x) → S
a q d
b S c
x
Using a tree grammar notation, x matches a subtree in the lhs and is copied over to the rhs
R1
s ⇒ q
ε
S
a S d
R3
a q d
b S c
R2
ε


R1: s → q
ε
R2: q(x) → S
x
R3: q(x) → A
a q d
b B c
x
Notice that the path from root to frontier is AnSBn
R1
s ⇒ q
ε
R3
⇒ A
a q d
b B c
ε
R3
⇒ A
a q d
b B c
ε
R2
⇒ A
a A d
a S d
b B c
b B c
ε

Adjoining Tree Grammars

- Similar to defn by (Lang, 1994)
- No adjoining constraints required
- Weakly equivalent to TAGs
- Set of tree languages for TAGs contained within that for ATGs
- Is ATG attractive for simplifying some TAG-based linguistic analysis?
  - Analyses that use adjoining constraints (feature structures)
  - Analyses that require different labels on rootnode and footnode

Adjoining Tree Grammars

- Closure properties for TALs (union, concat, homomorphism, substitution) can be shown using ATGs instead of TAGs.
  - By taking yield of the tree language
  - Without using adjunction constraints
- Intersection with regular languages (Lang, 1994)
- What about pumping lemma? cf. (Kanazawa, 2006)
- Polynomial time parsing algorithm provided by (Lang, 1994) = takes a string as input **not** a tree.

ATGs and monadic simple CFTGs

- Are ATGs strongly equivalent to monadic simple CFTGs?
- First step: what is strong equivalence?
- For each m.s. CFTG construct an ATG that produces the same tree set, and vice versa
- Shown by (Kepser & Rogers, 2011): TAGs closed under node relabeling are equal to monadic simple CFTGs.

Motivation #2
Lexicalization of Context-Free Grammars

Lexicalization of Grammars

- We know that a CFG can be ambiguous: provide more than one parse tree for an input string
- A CFG can be infinitely ambiguous
- Structure can be introduced without influence from input string, e.g. the chain rule NP → NP has this effect
- Lexicalization of a grammar means that each rule or elementary object in the grammar is associated with some terminal symbol

Lexicalization of Grammars

- Lexicalization is an interesting idea for syntax, semantics (in linguistics) and sentence processing (in psycho-linguistics)
- What if each word brings with it the syntactic and semantic context that it requires?
- Let us consider lexicalization of Context-free Grammars (CFGs)

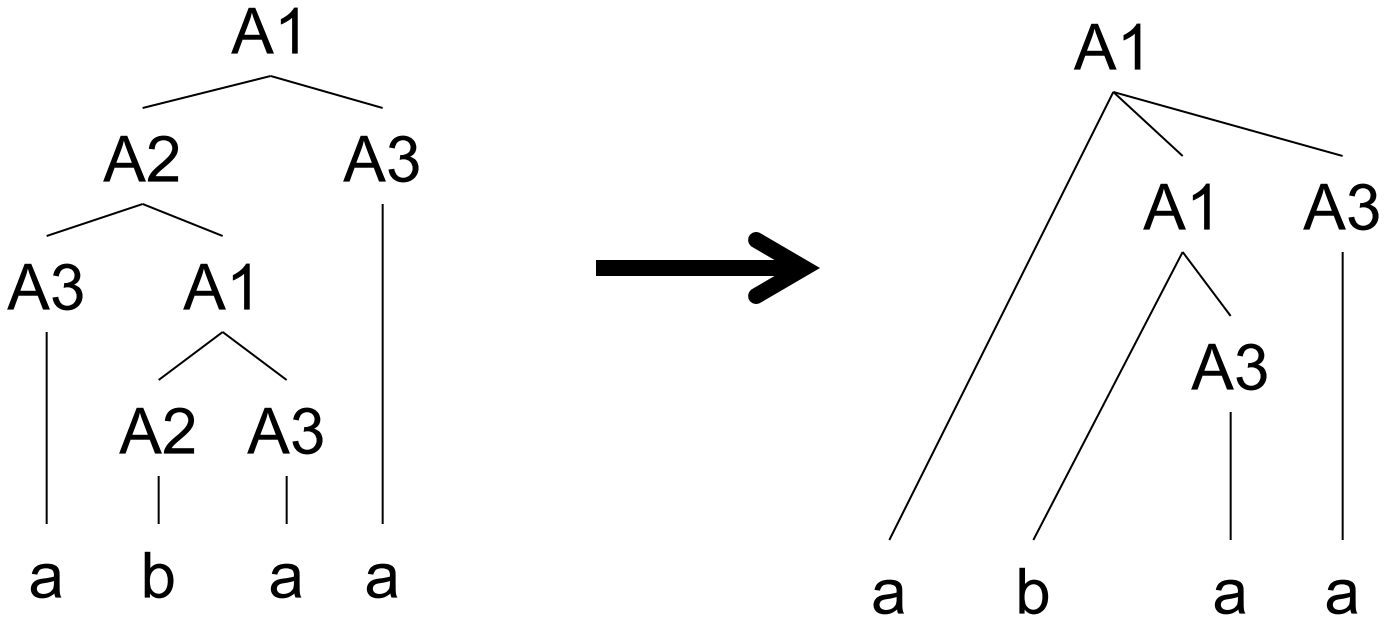
Lexicalization of CFGs

- A **normal form** is a grammar transformation that does not change the language of the grammar
- Can we transform every CFG to a normal form where there is guaranteed to be a terminal symbol on the right hand side of each rule
- Answer: yes - using Greibach Normal Form (GNF)
- GNF: every CFG can be transformed into the form $A \rightarrow a\alpha$ where A is a non-terminal, a is a terminal and α is a string of terminals and non-terminals

T(G) ≠ T(GNF(G))

A1 → A2 A3
A2 → A3 A1 | b
A3 → A1 A2 | a

Greibach Normal Form does not provide a strongly equivalent lexicalized grammar: the original tree set is not preserved

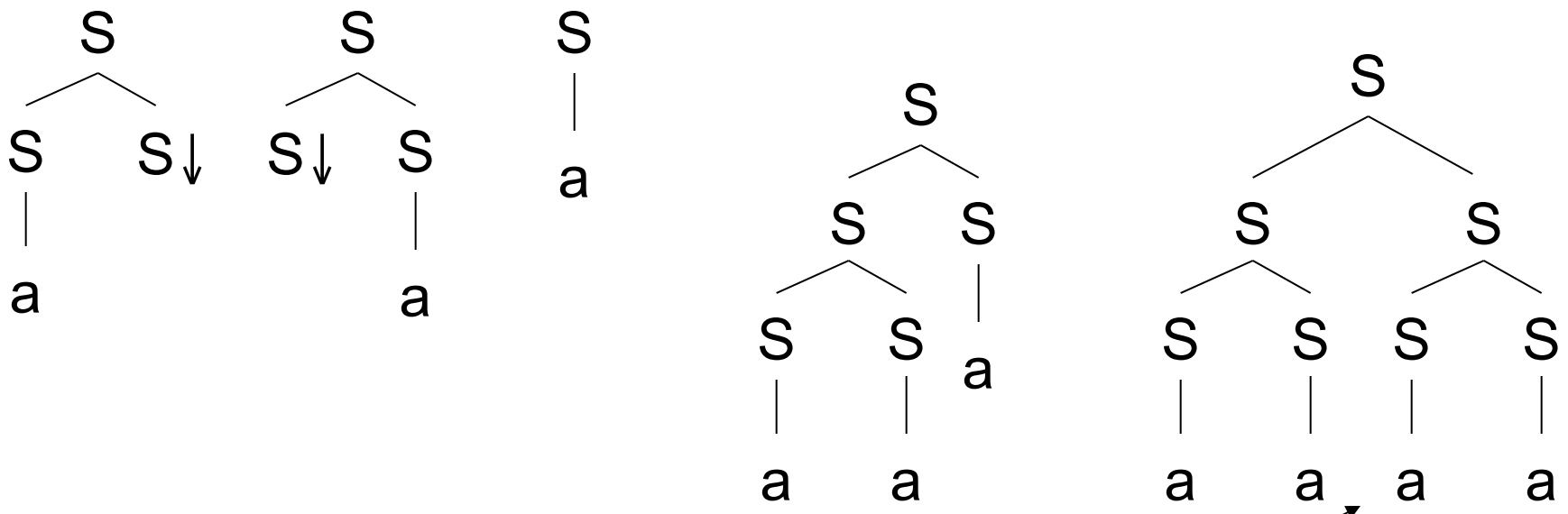


Tree Substitution Grammar

$S \rightarrow S\ S$

$S \rightarrow a$

Consider a simple expansion of each context-free rule into a tree fragment where each fragment is lexicalized

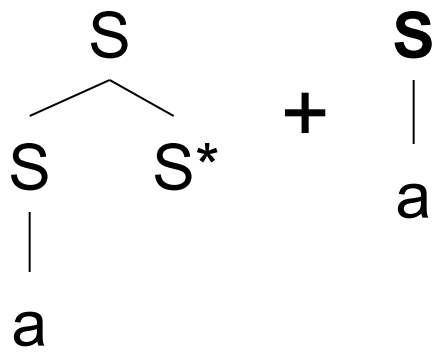


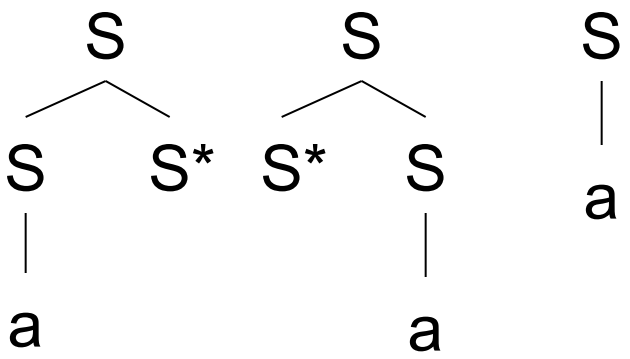
this tree cannot be derived

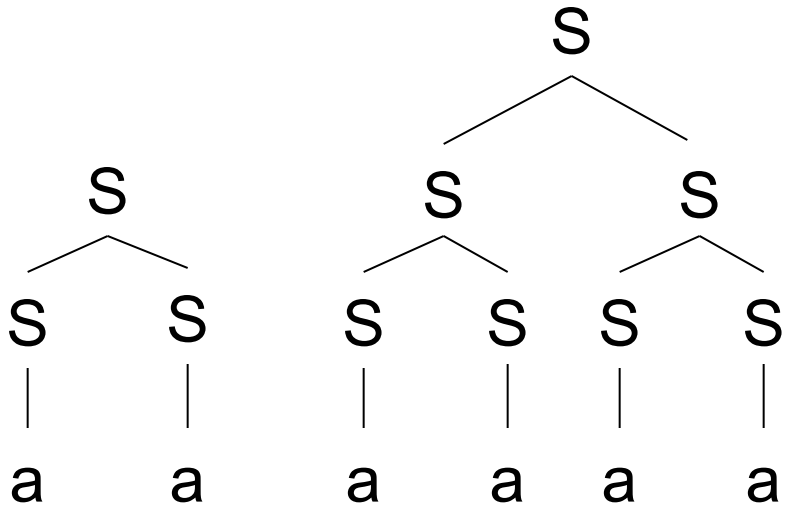
Tree Adjoining Grammar

$S \rightarrow S\ S$

$S \rightarrow a$



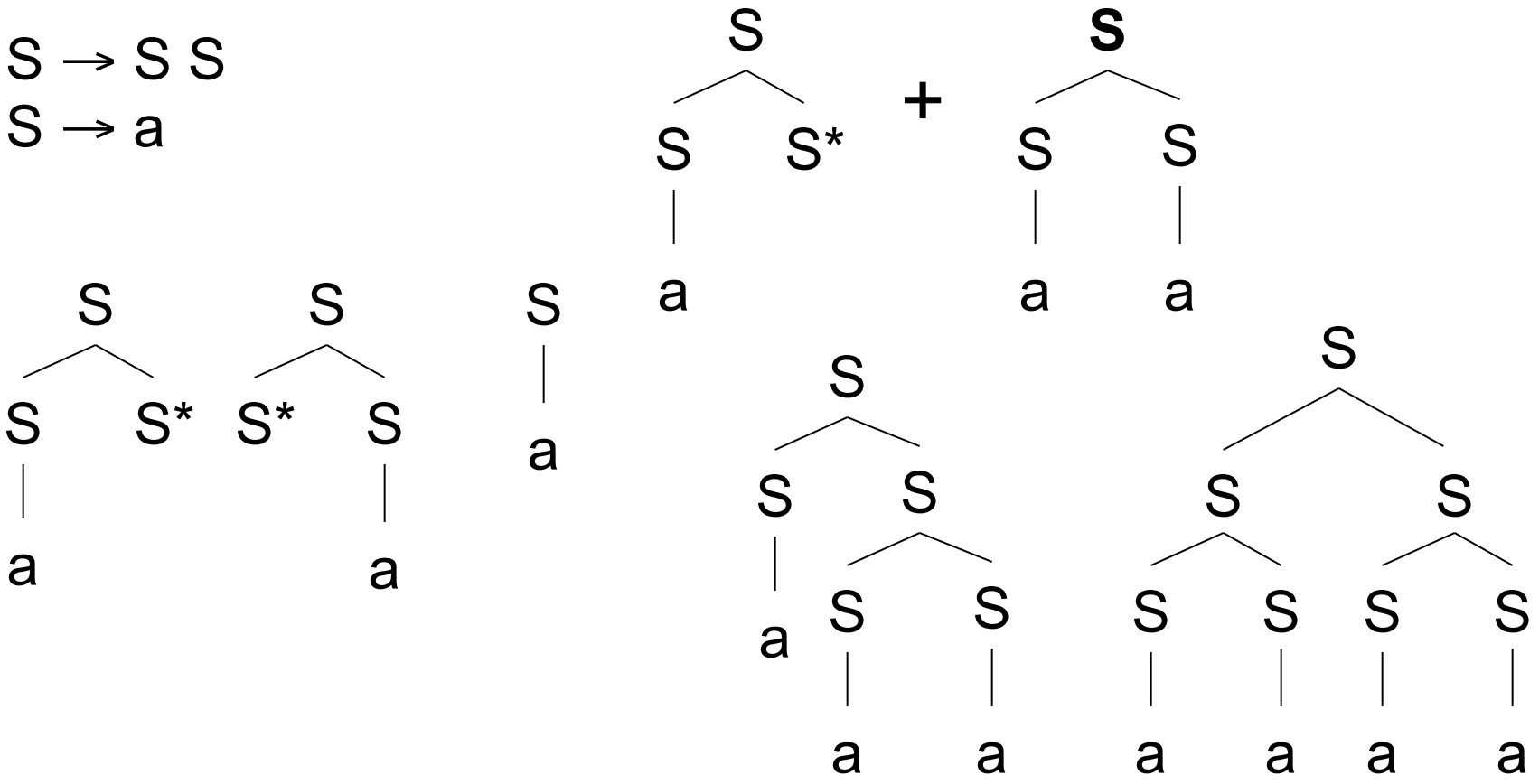




Tree Adjoining Grammar

$S \rightarrow S\ S$

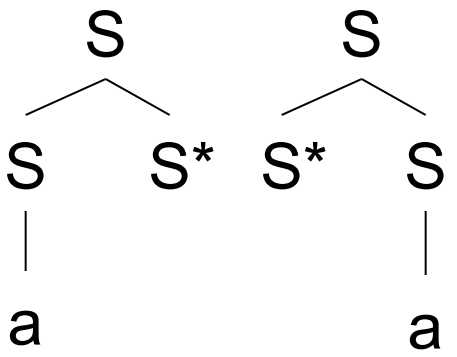
$S \rightarrow a$

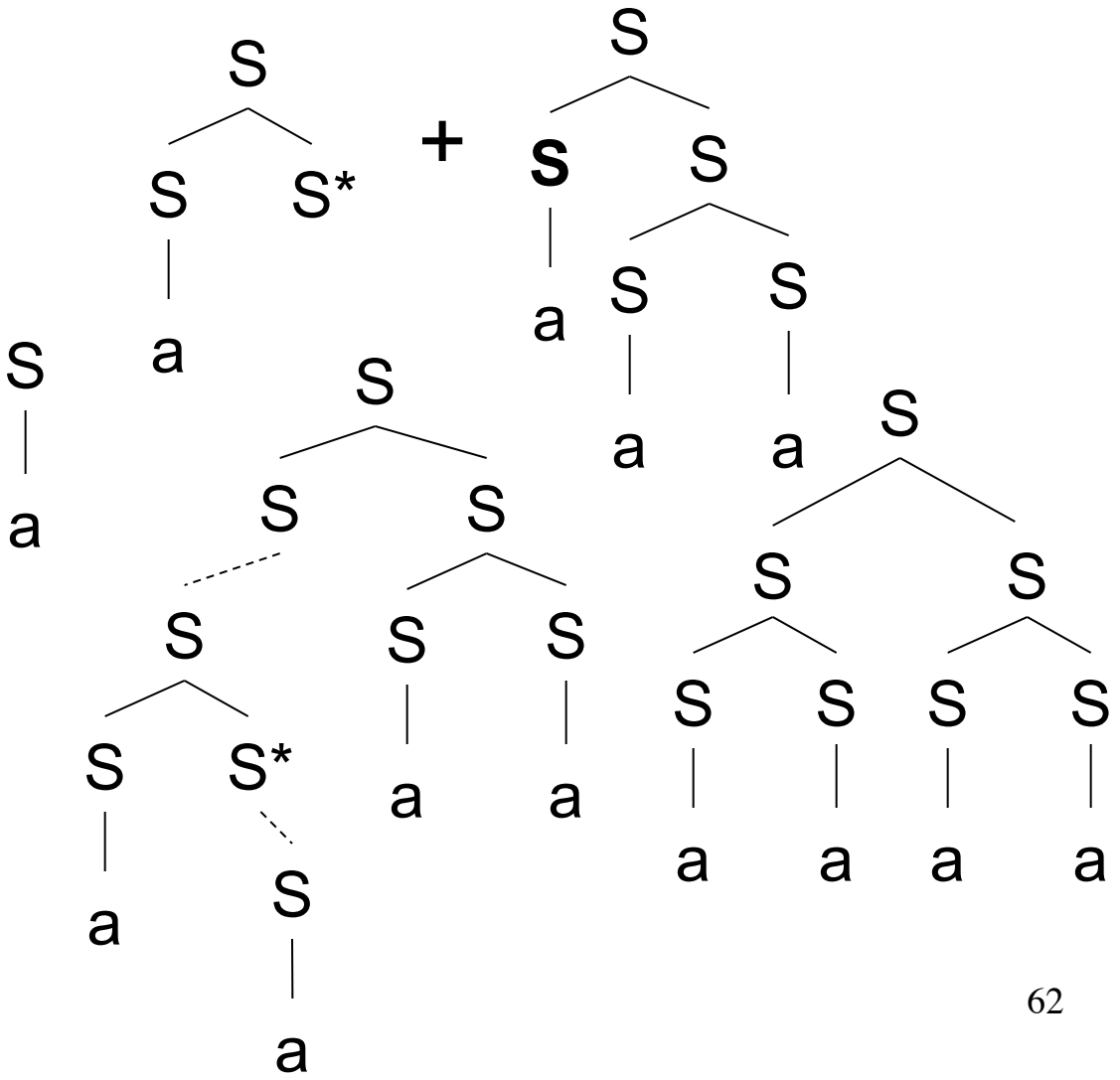


Tree Adjoining Grammar

$S \rightarrow S\ S$

$S \rightarrow a$

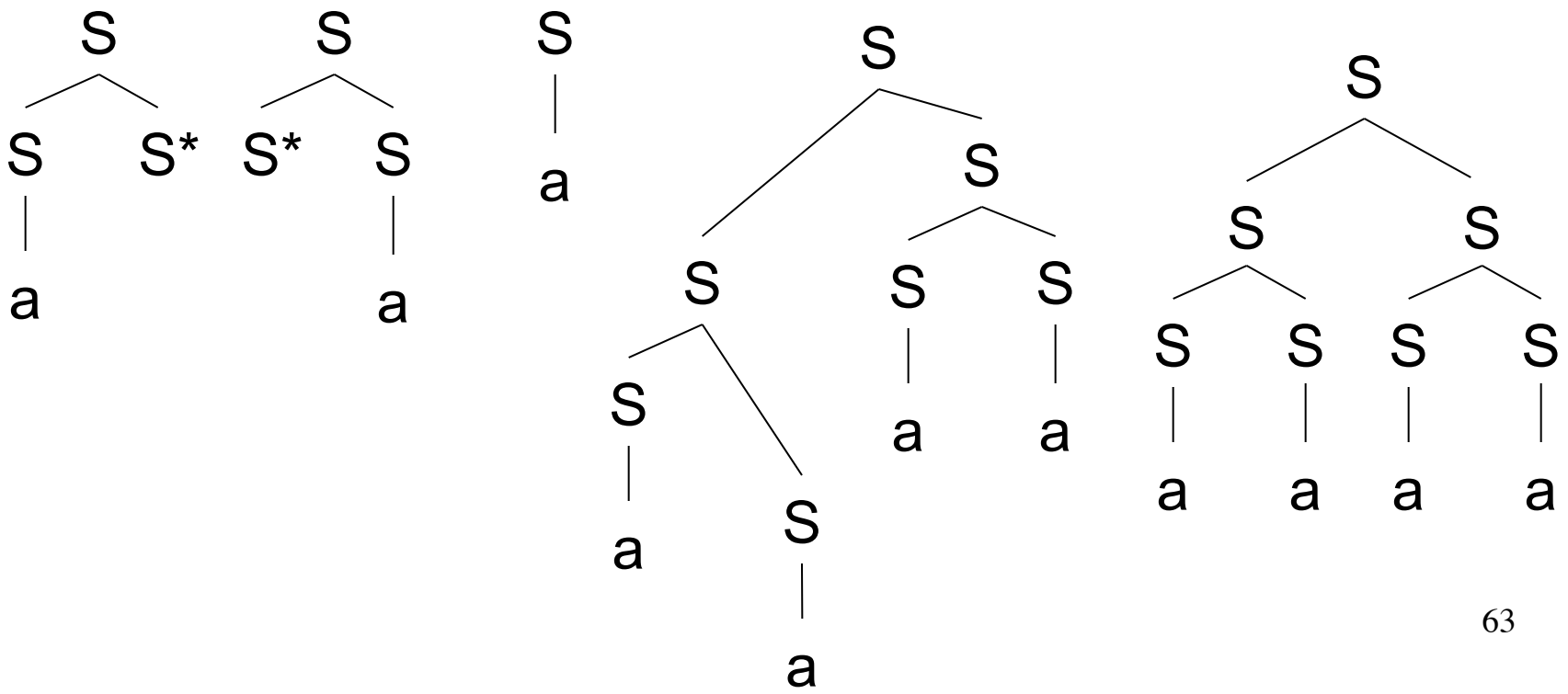




Tree Adjoining Grammar

$S \rightarrow S\ S$

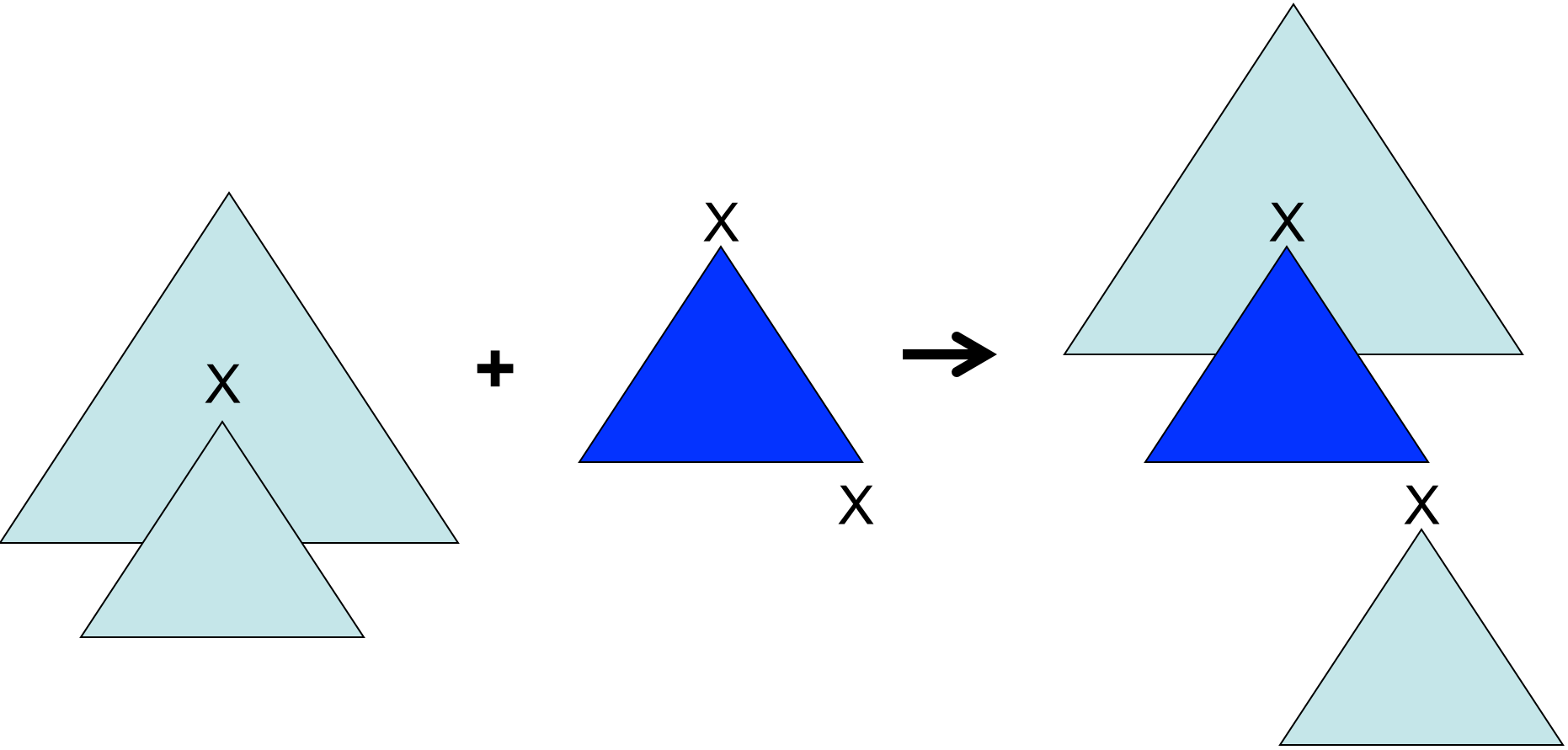
$S \rightarrow a$



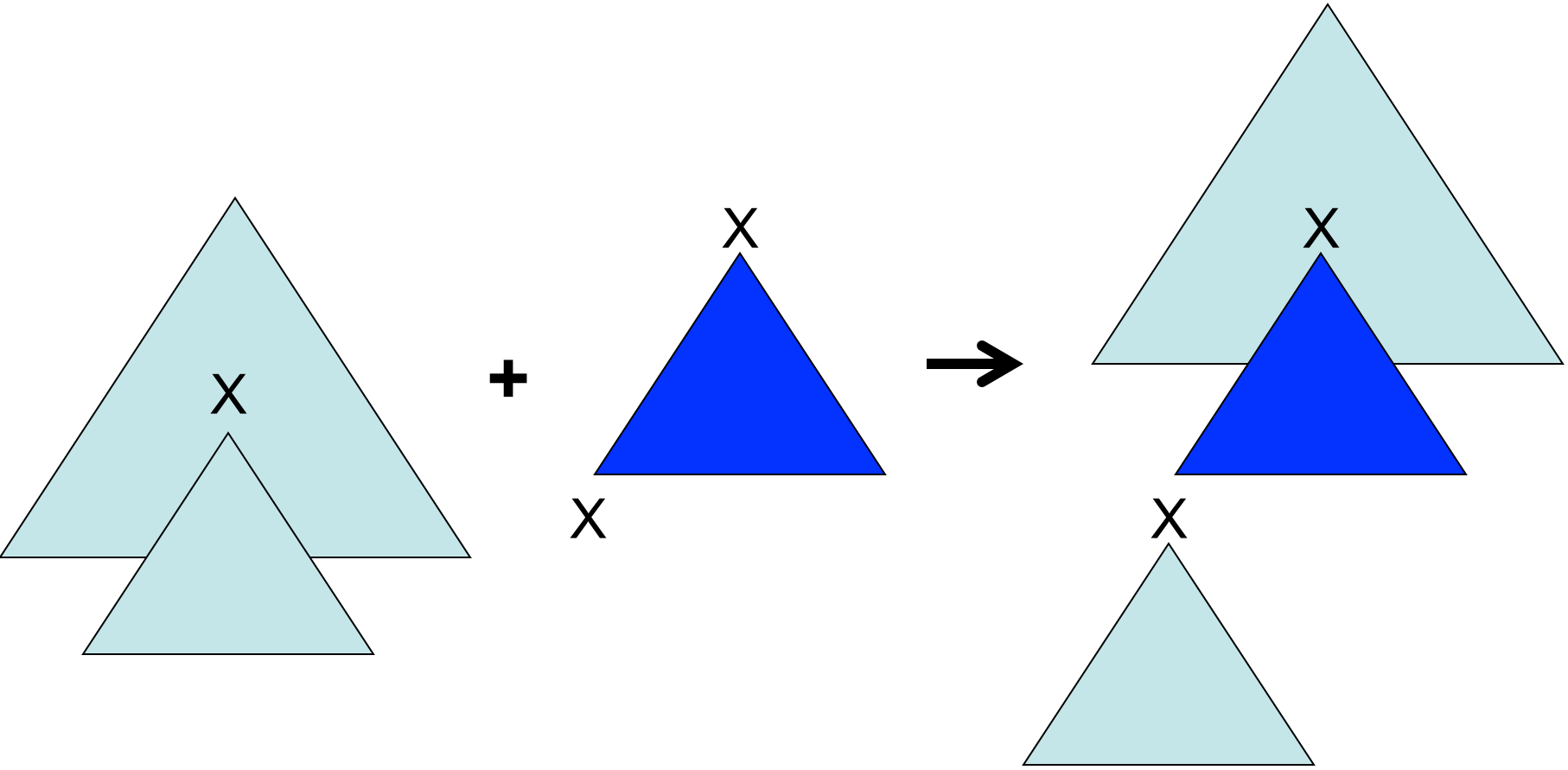
Lexicalization Through TAG

- This was an instructive example of how adjoining can be used to lexicalize CFGs while preserving the tree sets (strong generative capacity)
- (Joshi & Schabes, 1997) show that every CFG can be strongly lexicalized by TAG
- Later work by Schabes et al shows that CFGs can be lexicalized by **Tree-insertion grammars** which are weakly equivalent to CFGs

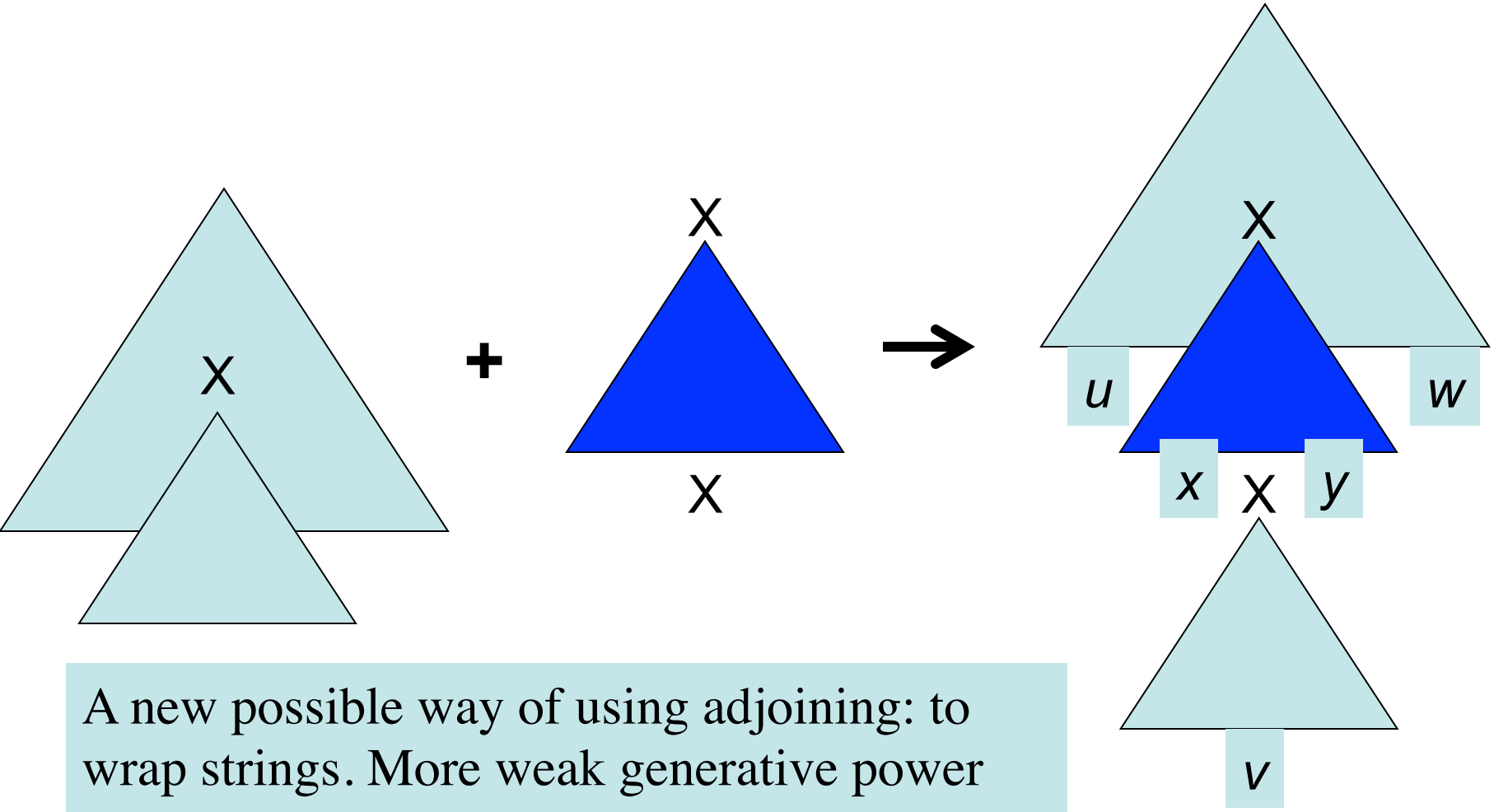
Tree-Insertion with adjoining



Tree-Insertion with adjoining



Wrapping with adjoining



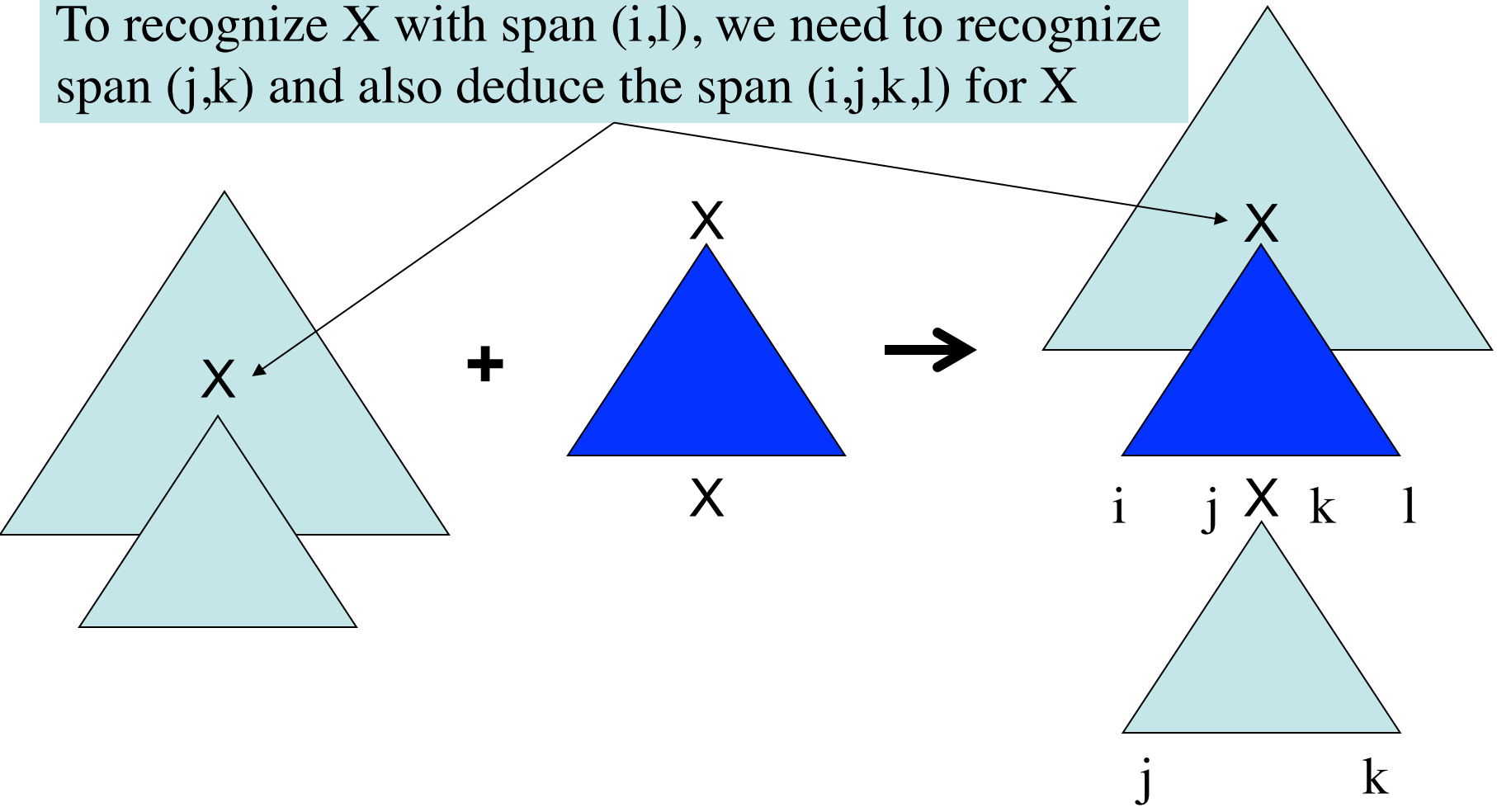
A new possible way of using adjoining: to wrap strings. More weak generative power than concatenation possible in CFGs.

Lexicalization and TAG

- (Kuhlmann & Satta, CL 2012) show that Tree-Adjoining Languages are **not** closed under lexicalization
- Every TAL does not have a lexicalized TAG grammar
- (Maletti and Engelfriet, ACL 2012) show that Context-free Tree Grammars of rank 2 **can** lexicalize TAGs

Parsing Complexity: CKY for TAG

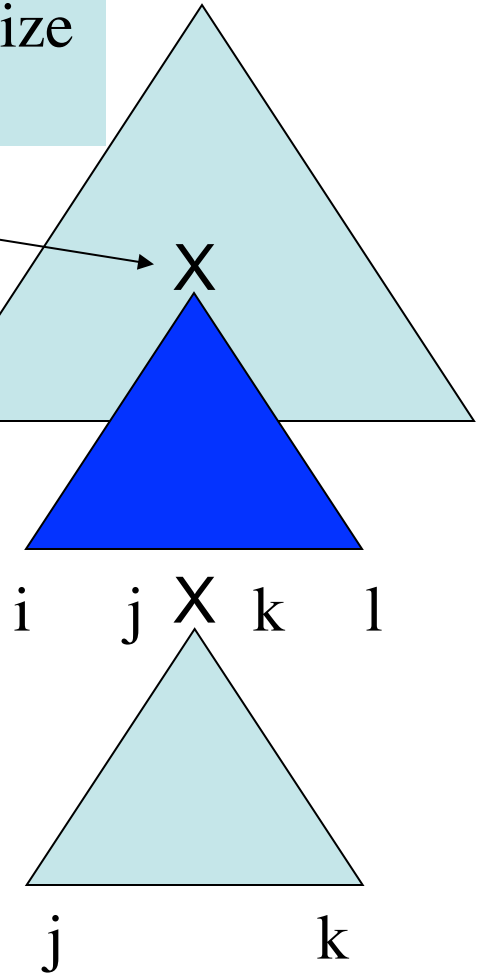
To recognize X with span (i,l), we need to recognize span (j,k) and also deduce the span (i,j,k,l) for X



Parsing Complexity: CKY for TAG

To recognize X with span (i,l), we need to recognize span (j,k) and also deduce the span (i,j,k,l) for X

- Each substring (i,l) can be a constituent, there are $O(n^2)$ substrings,
- For each of them we need to check for each non-terminal if it dominates an adjunction span (i,j,k,l)
- There are $O(n^4)$ such spans
- Hence we have complexity of recognizing membership of a string in a TAG to be $O(n^6)$



TAG Formal Properties
(Vijay-Shanker, 1987)

- Membership is in P: $O(n^6)$
- Tree-Adjoining Languages (TALs) are closed under *union*, *concatenation*, *Kleene closure* (*), h, h^{-1}, *intersection with regular languages*, and *regular substitution*
- There is also a pumping lemma for TALs
- TALs are a full abstract family of languages (AFL)
- TALs are not closed under intersection, intersection with CFLs, and complementation

Motivation #3

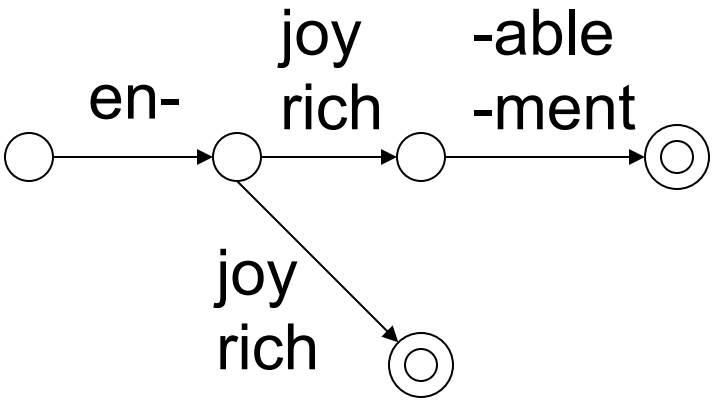
Is Human Language Regular, Context-free or Beyond?

Natural Language & Complexity

- One notion of computational complexity: the complexity of various recognition and generation algorithms
- Another notion: the complexity of the description of human languages
- What is the lowest upper bound on the description of all human languages? regular, context-free or beyond?
- Describes a class of languages, including closure properties such as union, intersection, etc.
- Automata theory provides recognition algorithms, determinization, and other algorithms

Grammar Size

- Consider the set of strings that includes *enjoy*, *enrich*, *enjoyable*, *enrichment* but not **joyable*, **richment*
- The CFG is clearly more compact
- Argument from learning: if you already know *enjoyment* then learning *rich* means you can generate *enrichment* as well



V → X
A → X -able | X -ment
X → en- NA
NA → joy | rich

Regular grammars can be exponentially larger than equivalent CFGs

Sufficient Generative Capacity

- Does a formal grammar have sufficient generative capacity?
- Two cases: **weak** and **strong** generative capacity
- For strong GC: does the grammar permit the right kind of dependencies, e.g. nested dependencies
- For weak GC: usually requires some kind of homomorphism into a formal language whose weak GC can be determined (the formal language class should be closed under homomorphisms)

Is NL regular: strong GC

- Regular grammars cannot derive nested dependencies
- Nested dependencies in English:
  - the shares that the broker recommended were bought

    N1 N2 V2 V1
  - the moment when the shares that the broker recommended were bought has passed

    N1 N2 N3 V3 V2 V1
- Can you provide an example with 4 verbs?
- Set of strings has to be infinite: competence vs. performance

Is NL regular: strong GC

- Assume that in principle we could process infinitely nested dependencies: **competence** assumption
- The reason we cannot is because of lack of memory in pushdown automata: **performance** can be explained
- CFGs can easily obtain nested dependencies

$$S \rightarrow a\ S\ b$$
$$S \rightarrow \varepsilon$$

$$S1 \Rightarrow a1\ S2\ b1$$
$$\Rightarrow a1\ a2\ S3\ b2\ b1$$
$$\Rightarrow a1\ a2\ \ldots\ aN\ S\ bN\ \ldots\ b2\ b1$$
$$\Rightarrow a1\ a2\ \ldots\ aN\ bN\ \ldots\ b2\ b1$$

Is NL regular: Weak GC

- Consider the following set of strings (sentences):
  - S = *if* S *then* S
  - S = *either* S *or* S
  - S = *the man who said* S *is arriving today*
- Map *if, then* to *a* and *either, or* to *b*
- Map everything else to the empty string
- This results in strings like *abba*, *abaaba*, or *abbaabba*

Is NL regular: Weak GC

- The language is the set of strings

  L = { ww’ : w from (a|b)* and w’ is reversal of w }
- L can be shown to be non-regular using the pumping lemma for regular languages
- L is context-free

Is NL context-free: Strong GC

- CFGs cannot handle crossing dependencies
- Dependencies like aN... a2 a1 bN... b2 b1 are not possible using CFGs
- But some widely spoken languages have clear cases of crossing dependencies
  - Dutch (Bresnan et al., 1982)
  - Swiss German (Shieber, 1984)
  - Tagalog (Rambow & MacLachlan, 2002)
- Therefore, in terms of strong GC, NL is not context-free

Is NL context-free: Weak GC

- Weak GC of NL being greater than context-free was harder to show, cf. (Pullum, 1982)
- (Huybregts, 1984) and (Shieber, 1985) showed that weak GC of NL was beyond context-free using examples with explicit case-marking from Swiss-German

mer	d'chind	em Hans	es huus	lönd	hälfed	aastriiche
we	children-**acc**	Hans-**dat**	house-**acc**	let-**acc**	help-**dat**	**paint-acc**
	[	(	{	]	)	}

this language is not context-free

Generating Crossing Dependencies

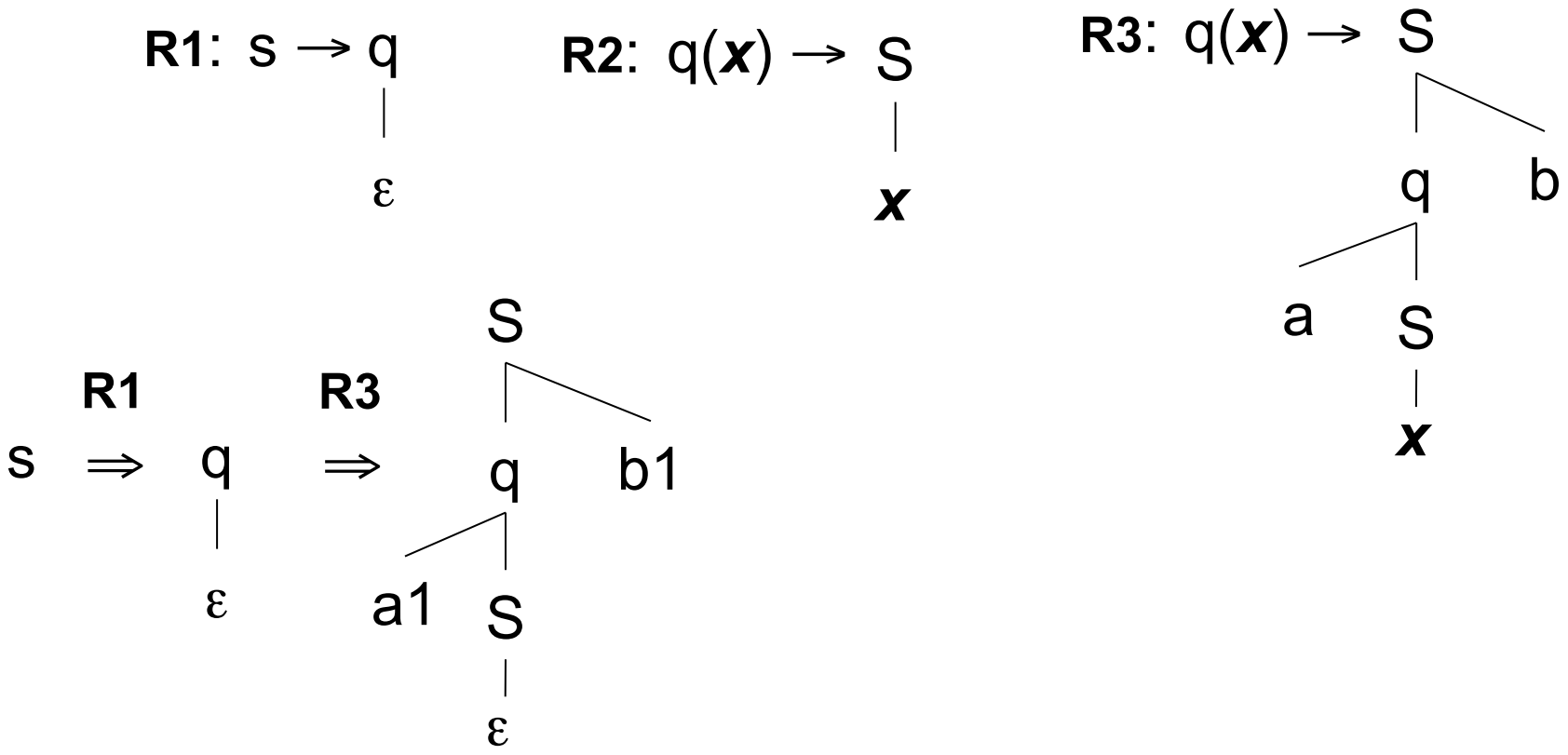
1: S → S B C
2: S → a C
3: a B → a a
4: C B → B C
5: B a → a a
6: C → b

S1 ⇒ **S2 B1 C1** (1)
⇒ **S3 B2 C2** B1 C1 (1)
⇒ **a3 C3** B2 C2 B1 C1 (2)
⇒ a3 **B2 C3** C2 B1 C1 (4)
⇒ **a3 a2** C3 C2 B1 C1 (3)
⇒ a3 a2 C3 **B1 C2** C1 (4)
⇒ a3 a2 **B1 C3** C2 C1 (4)
⇒ a3 **a2 a1** C3 C2 C1 (3)
⇒ a3 a2 a1 **b3** C2 C1 (6)
⇒ a3 a2 a1 b3 **b2** C1 (6)
⇒ a3 a2 a1 b3 b2 **b1** (6)

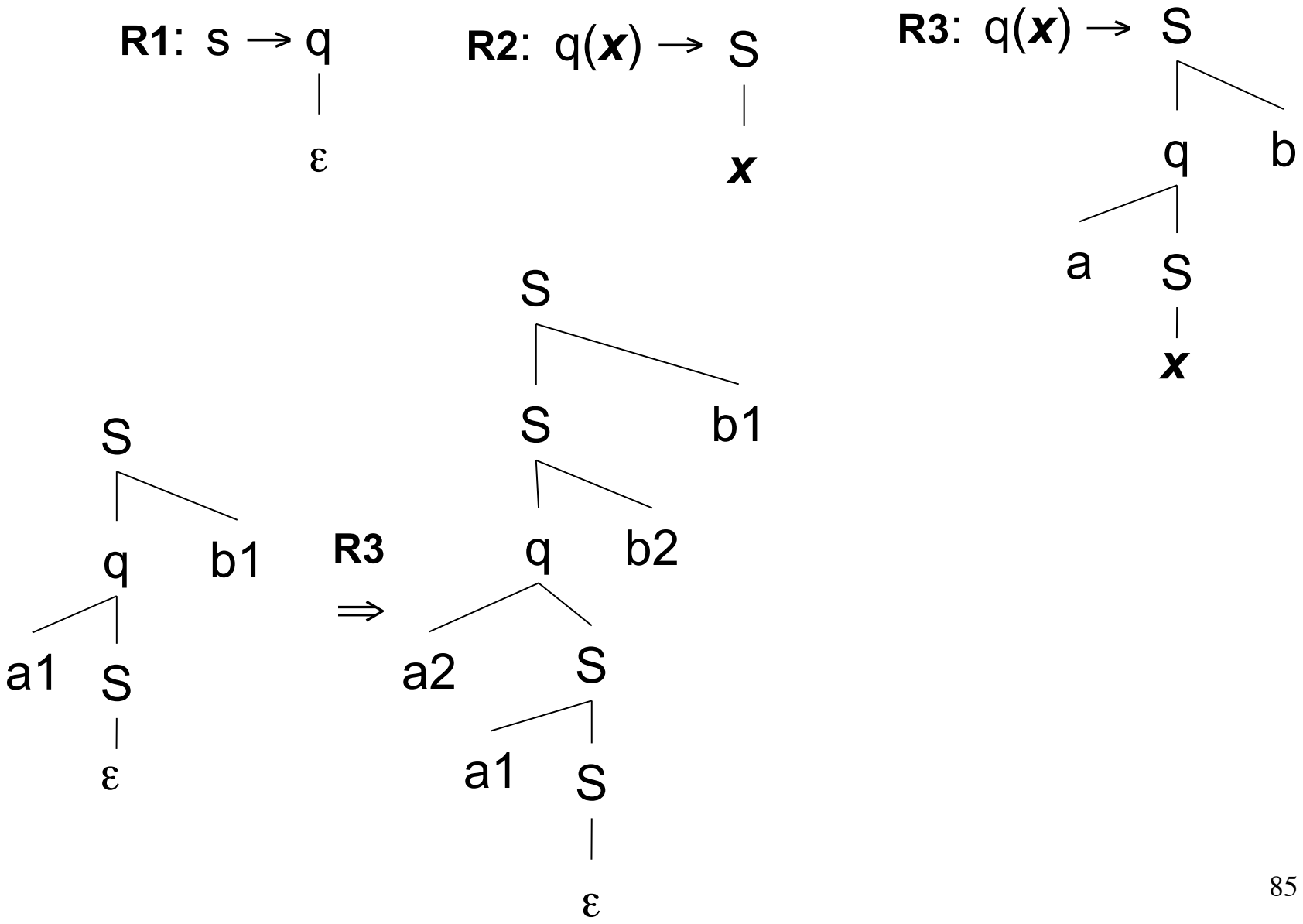
Simple Generation of Crossing Dependencies

- Instead of using powerful swapping operations (corresponding to more powerful automata)
- We instead build local dependencies into elementary trees
- Strong GC: Crossing dependencies arise by simple composition of elementary trees
- The context-sensitive part is built into each elementary tree: the remaining composition is "context-free"
- Weak GC: Crossing dependencies = string wrapping

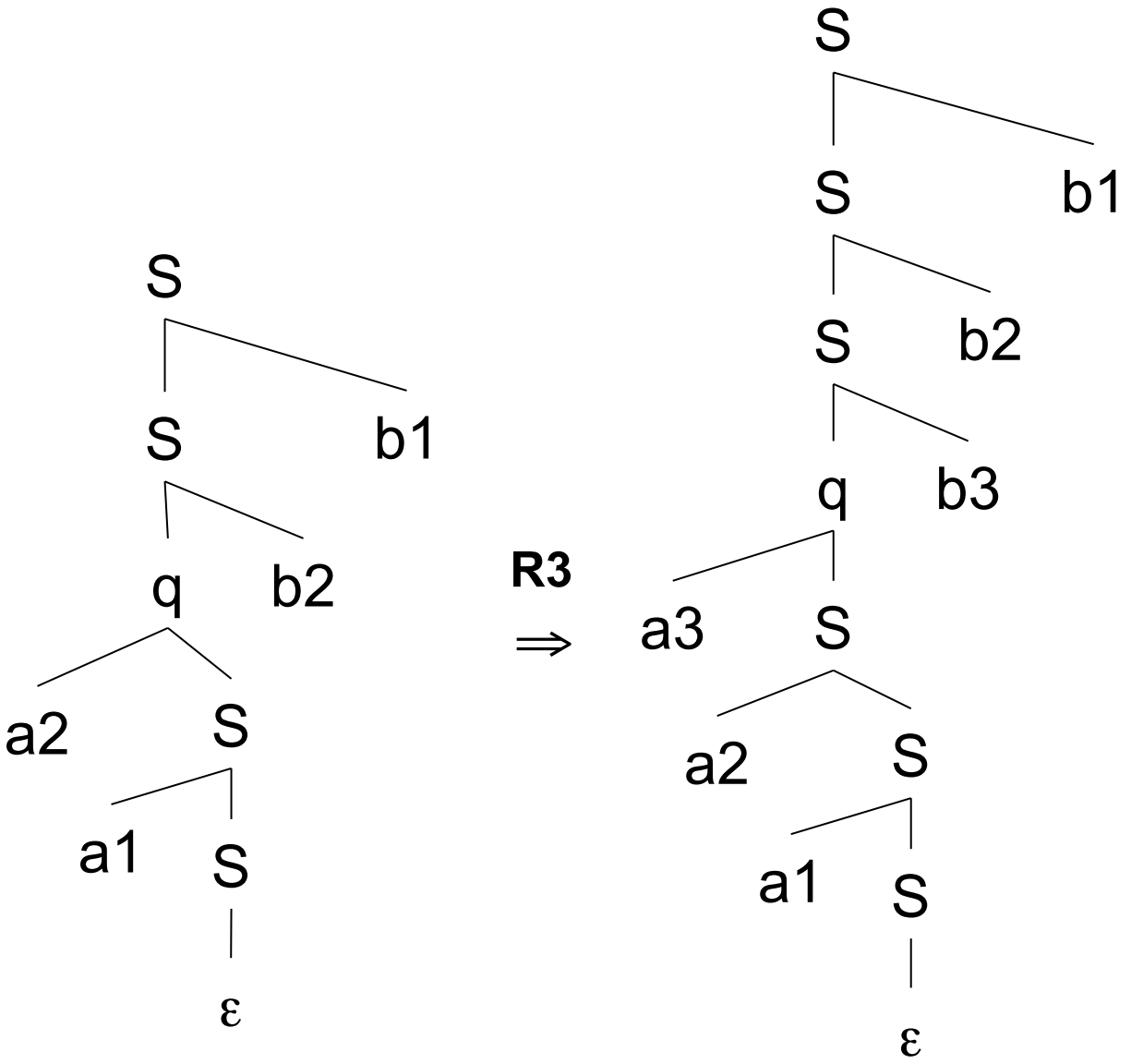
Crossing Dependencies with Adjoining



Crossing Dependencies with Adjoining



Crossing Dependencies with Adjoining

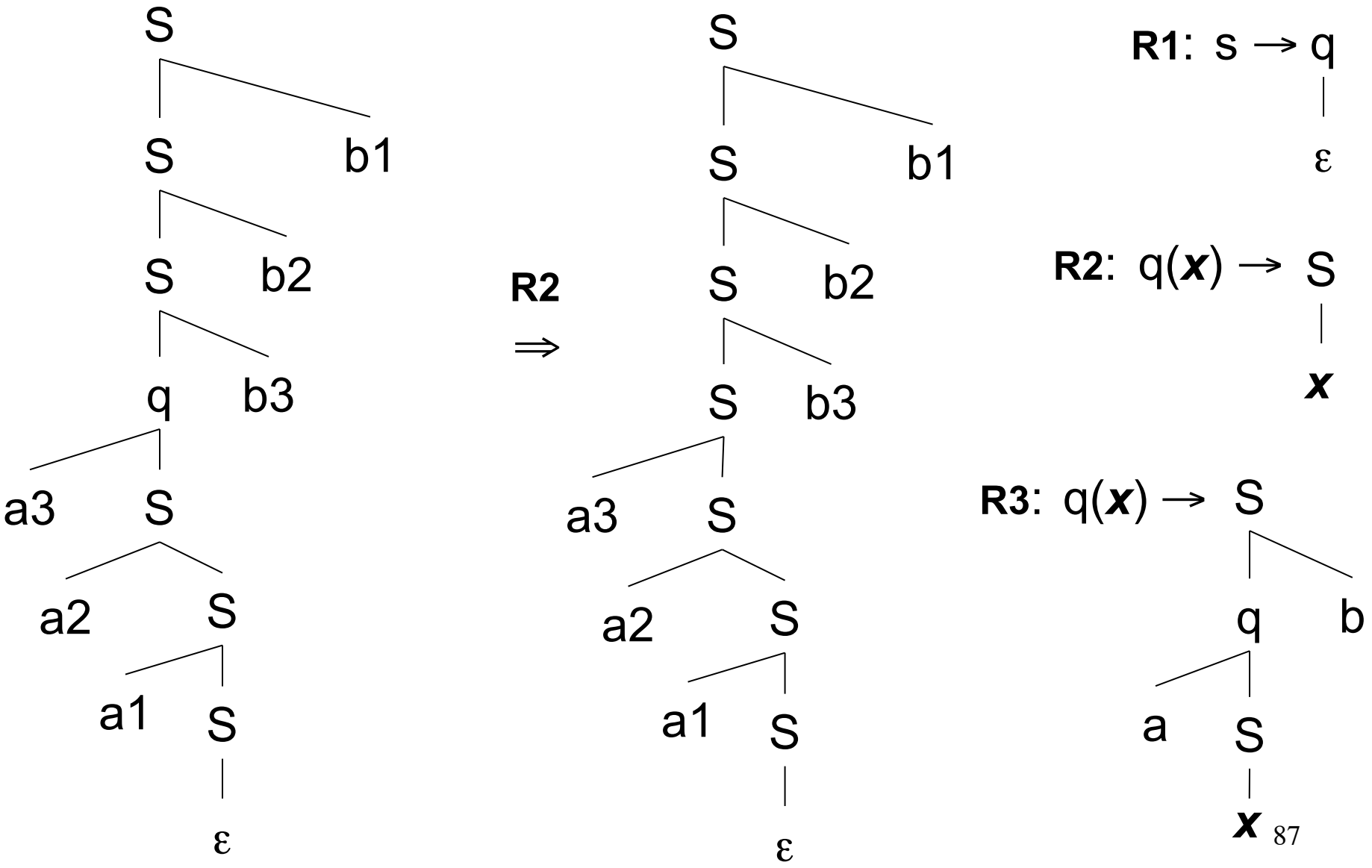


R1: s → q
ε

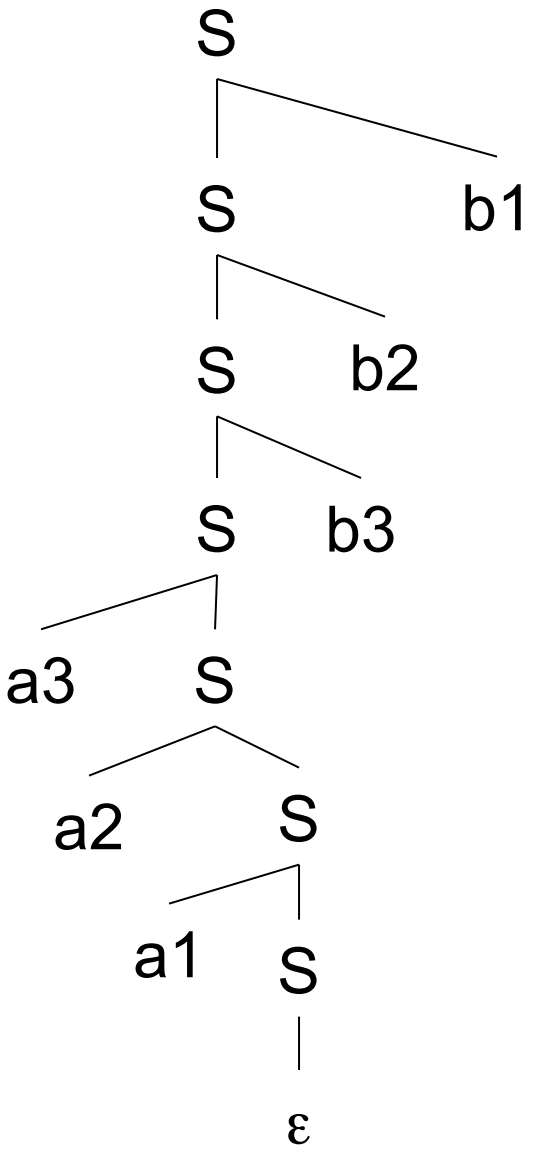
R2: q(***x***) → S
x

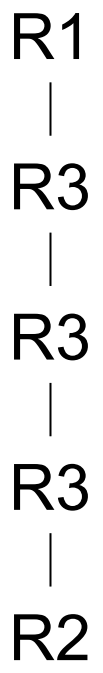
R3: q(***x***) → S
q b
a S
x

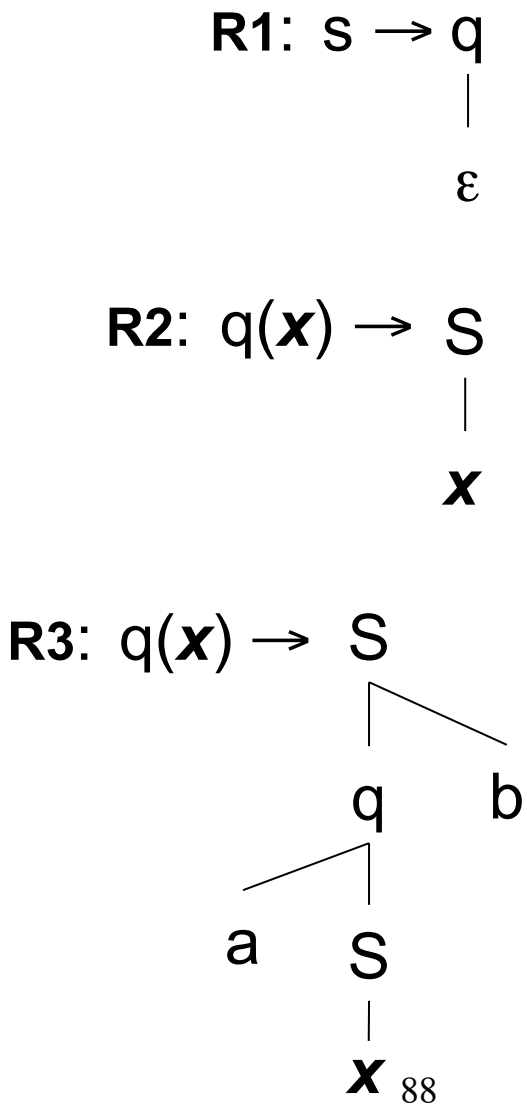
Crossing Dependencies with Adjoining



Crossing Dependencies with Adjoining





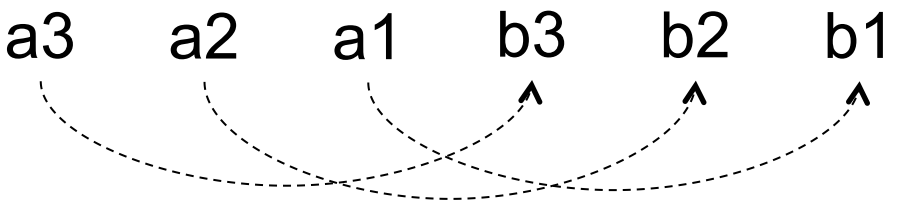


Crossing Dependencies with Adjoining

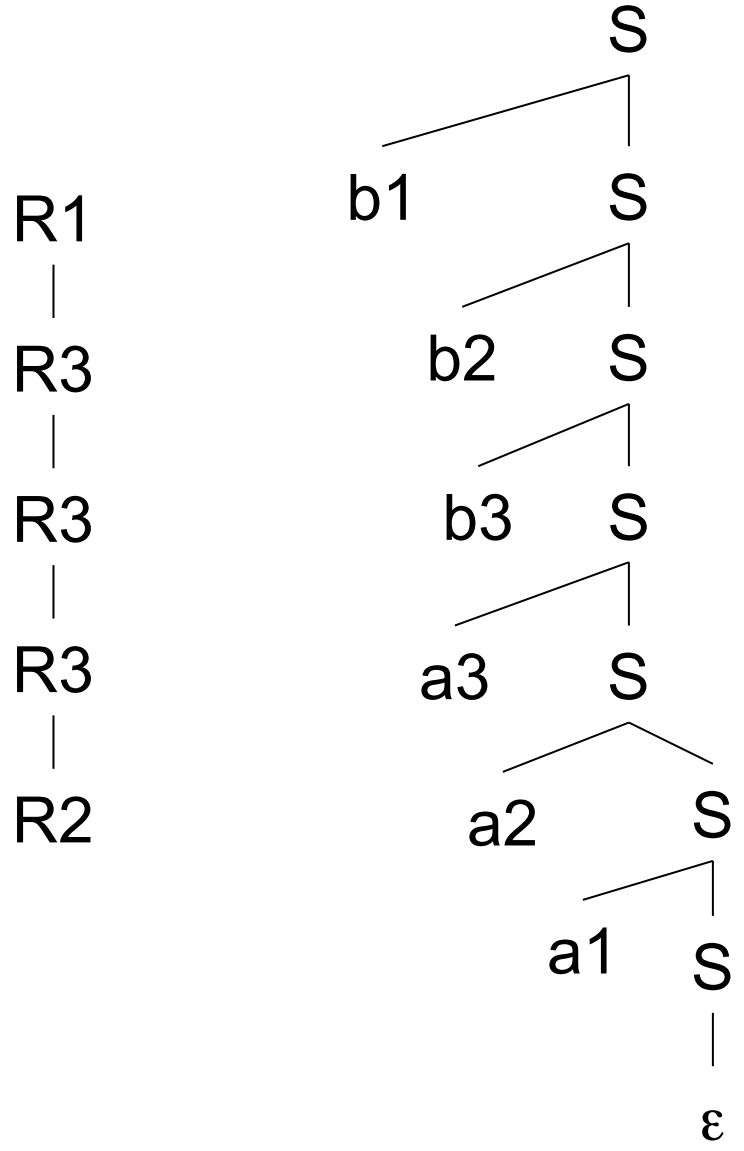
R1: s → q
ε

R2: q(***x***) → S
x

R3: q(***x***) → S
q b
a S
x



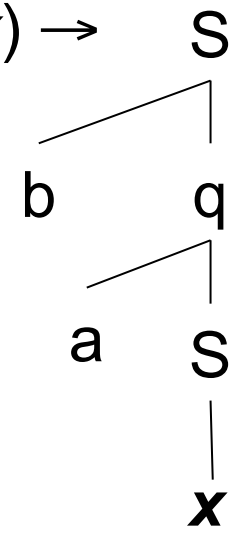
Nested Dependencies with Adjoining



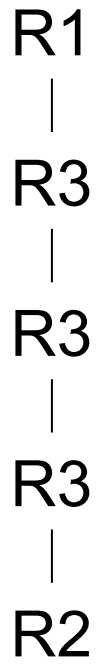
R1: s → q
ε

R2: q(***x***) → S
x

R3: q(***x***) →



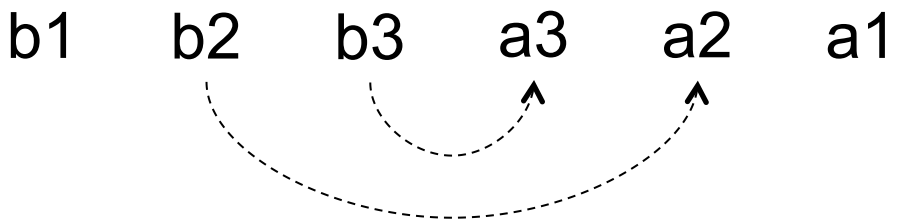
Nested Dependencies with Adjoining



R1: s → q
ε

R2: q(***x***) → S
x

R3: q(***x***) → S
b q
a S
x



Related Formalisms

- Another route to lexicalization of CFGs: categorial grammars (CG is not strongly equivalent to CFGs but can weakly lexicalize them)
- Several different mathematically precise formal grammars were proposed to deal with the motivations presented here
- Some examples:
  - Multiple context-free grammars (MCFG)
  - head grammars (HG does string wrapping);

Related Formalisms

- Some examples:
  - combinatory categorial grammars (CCG; extends CG);
  - linear indexed grammars (LIG; less powerful than indexed grammars)
- (Vijay-Shanker, 1987) and (Weir, 1988) showed that HG, CCG, LIG and TAG are all *weakly equivalent*!
- They also provide a powerful formalism called Linear Context-Free Rewriting System (LCFRS)

Mildly Context-Sensitive Formalism

(Joshi, 1985)

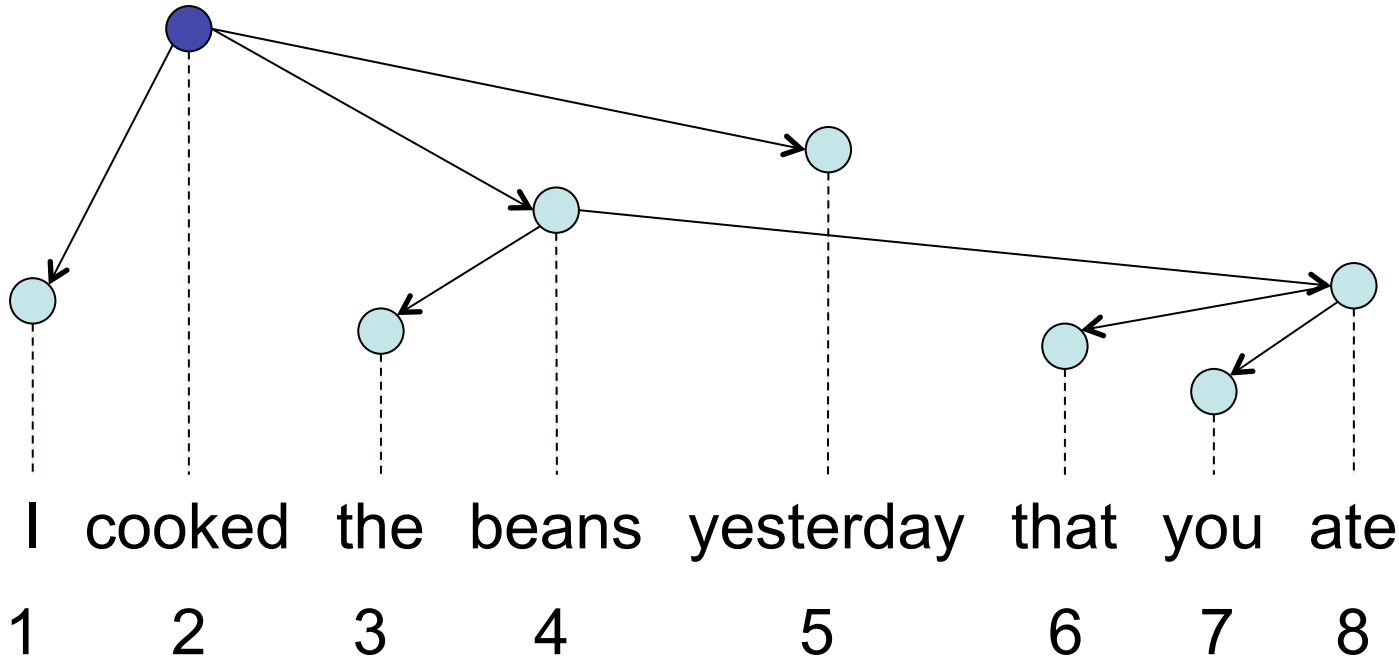
- Proper extension of Context-free Languages
  - Should contain CFLs
- Limited crossing dependencies
  - Less powerful than full context-sensitive languages
- Constant growth property
  - Relation of derivation to string is linear
- Tractable: polynomial time parsing

Tree-adjoining grammars and crossing dependencies

Material adapted from Marco Kuhlmann's slides

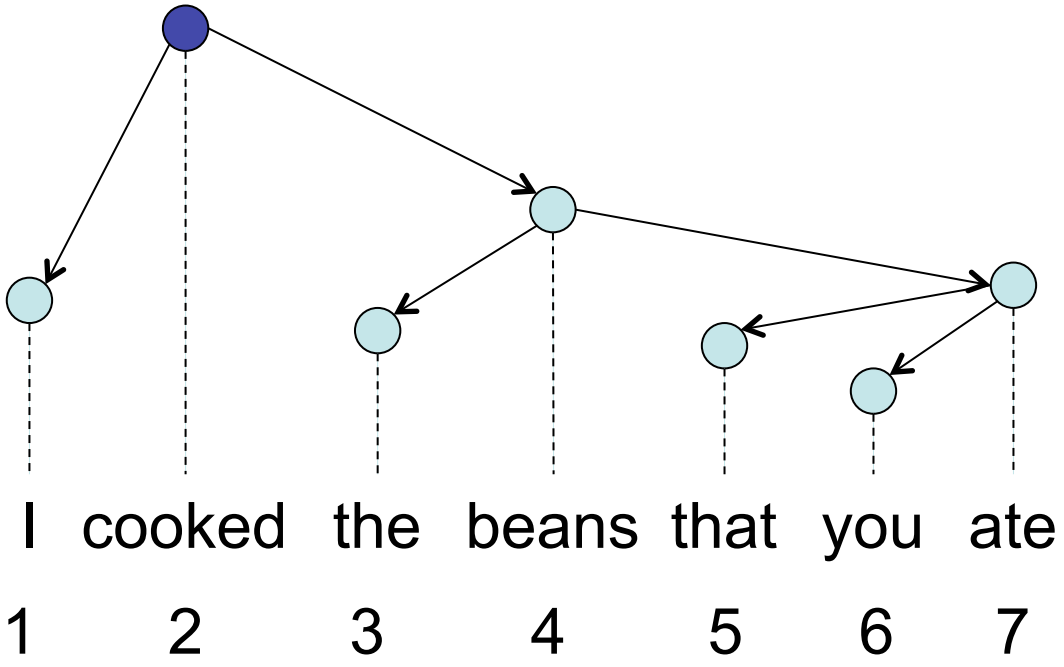
Dependency structures

Lexicalized Grammars produce dependencies



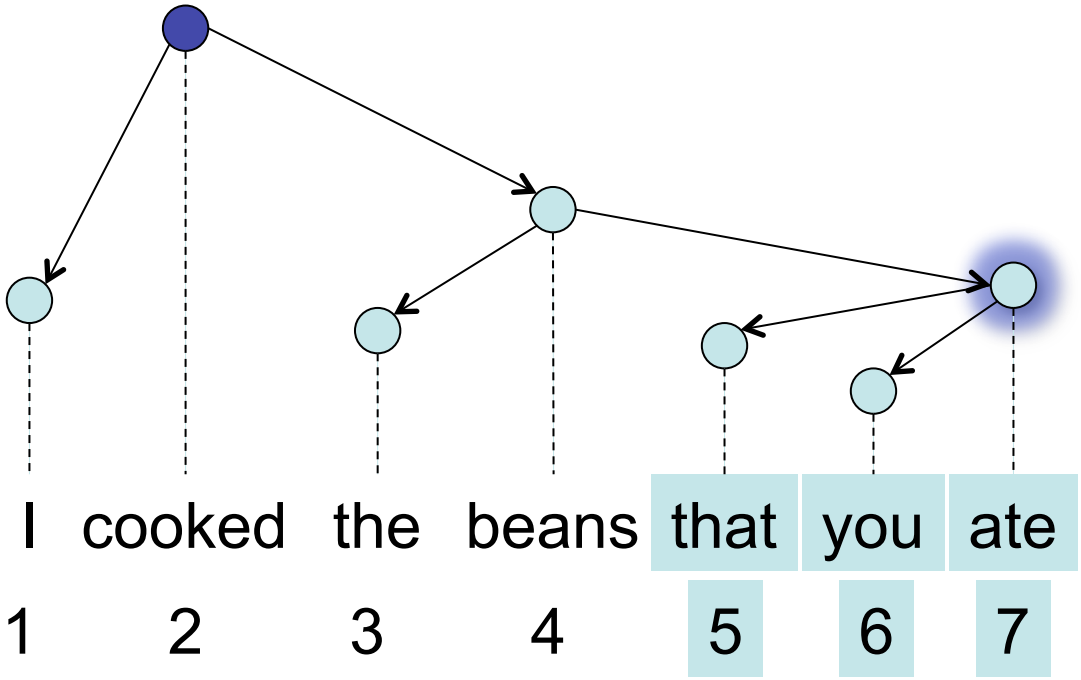
Projective Dependencies

- Intervals define contiguous sub-strings
- Projective dependencies: all subtrees form intervals



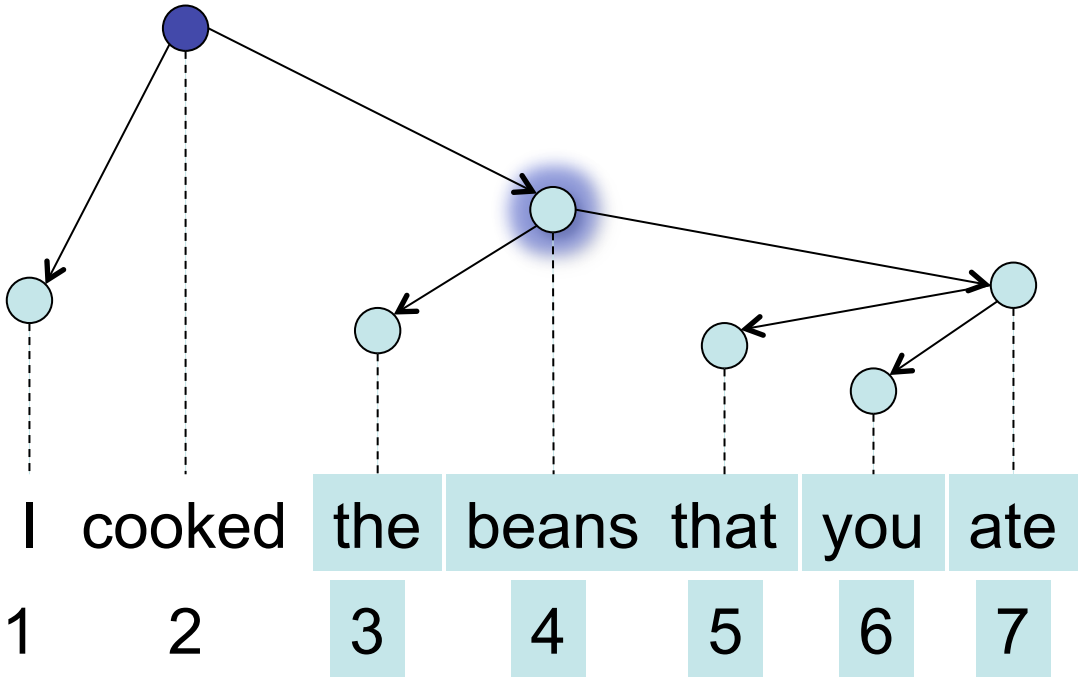
Projective Dependencies

- Intervals define contiguous sub-strings
- Projective dependencies: all subtrees form intervals, e.g. [5,7]



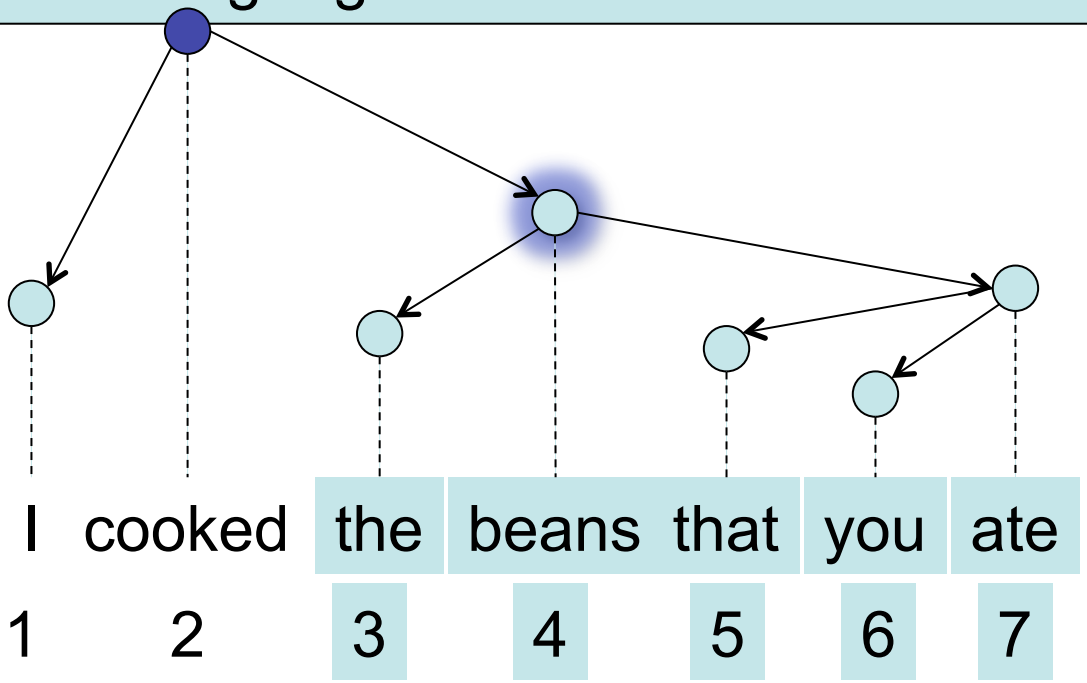
Projective Dependencies

- Intervals define contiguous sub-strings
- Projective dependencies: all subtrees form intervals, e.g. [3,7]



Projective Dependencies

- Context-free grammars can model projective dependencies
- Are projective dependencies sufficient for natural language?

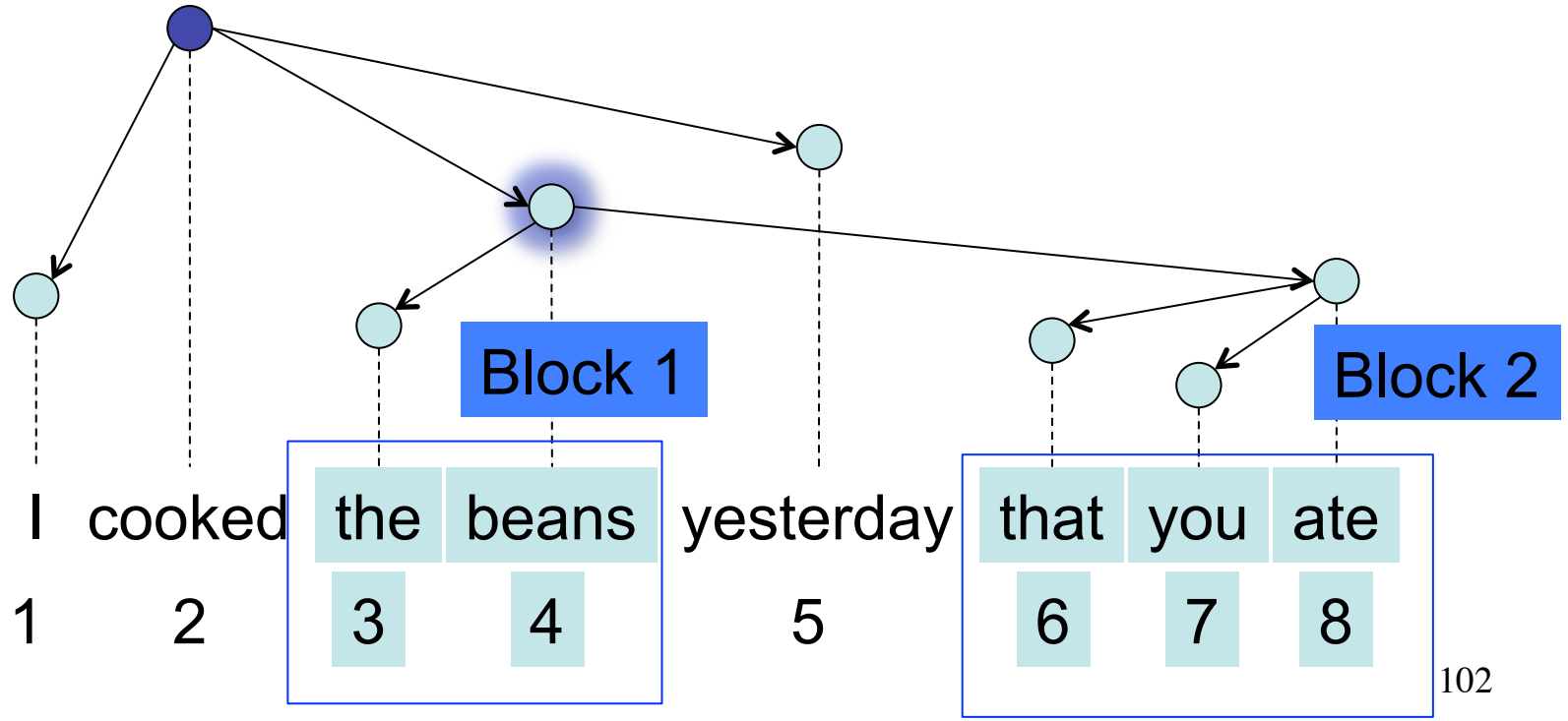


Projective Dependencies

Language	Crossing Dependencies %	Sentences with crossing deps %
Arabic	0.4	10.1
Basque	**2.9**	**26.2**
Catalan	0.1	2.9
Chinese	0.0	0.0
Czech	**1.9**	**23.2**
English	0.3	6.7
Greek	**1.1**	**20.3**
Hungarian	**2.9**	**26.4**
Italian	0.5	7.4
Turkish	**5.5**	**33.3**

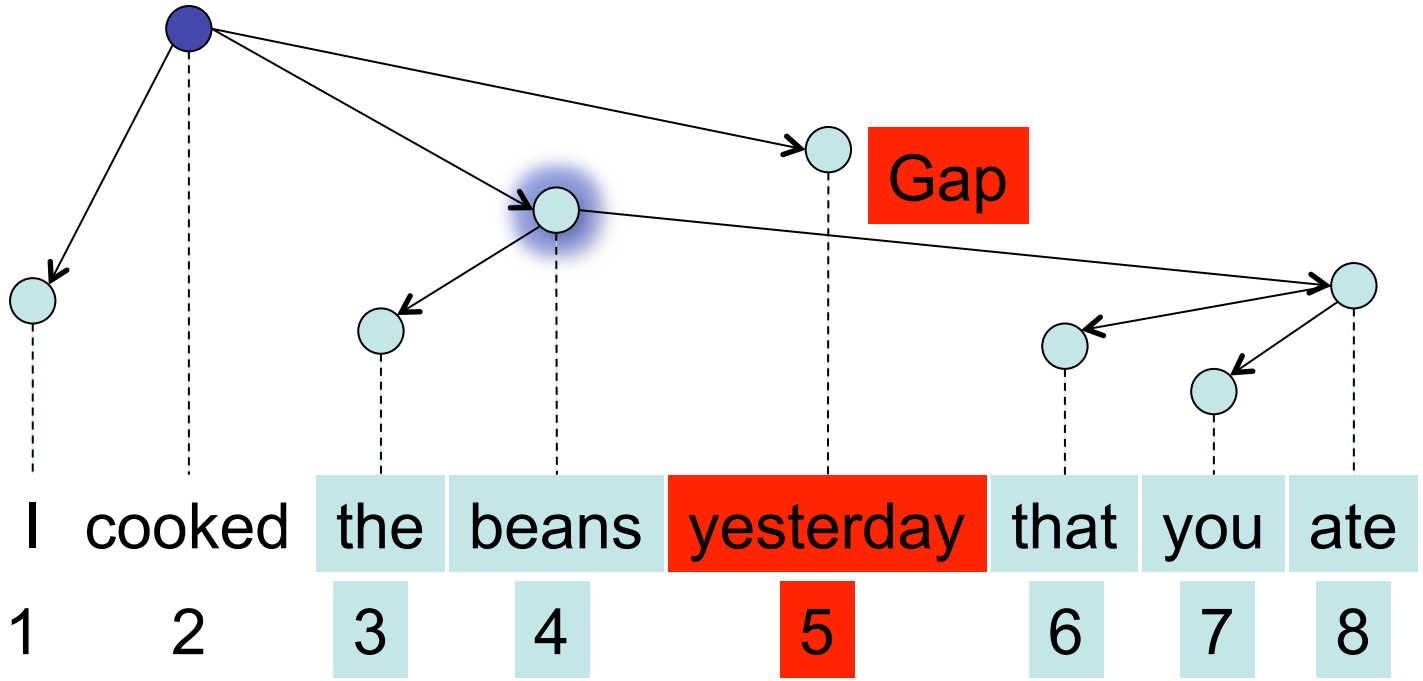
Block Degree

- Block degree measures non-projectivity
- Block degree = number of intervals per sub-tree, e.g. { [3,4], [6,8] } is block degree 2

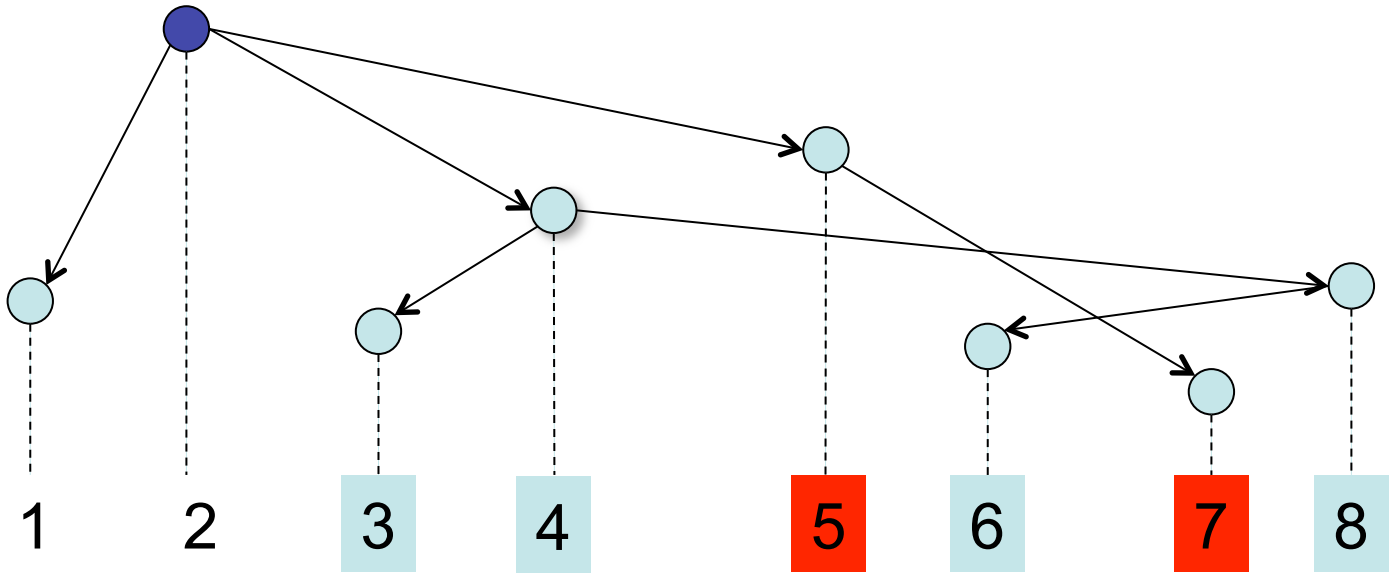


Block Degree

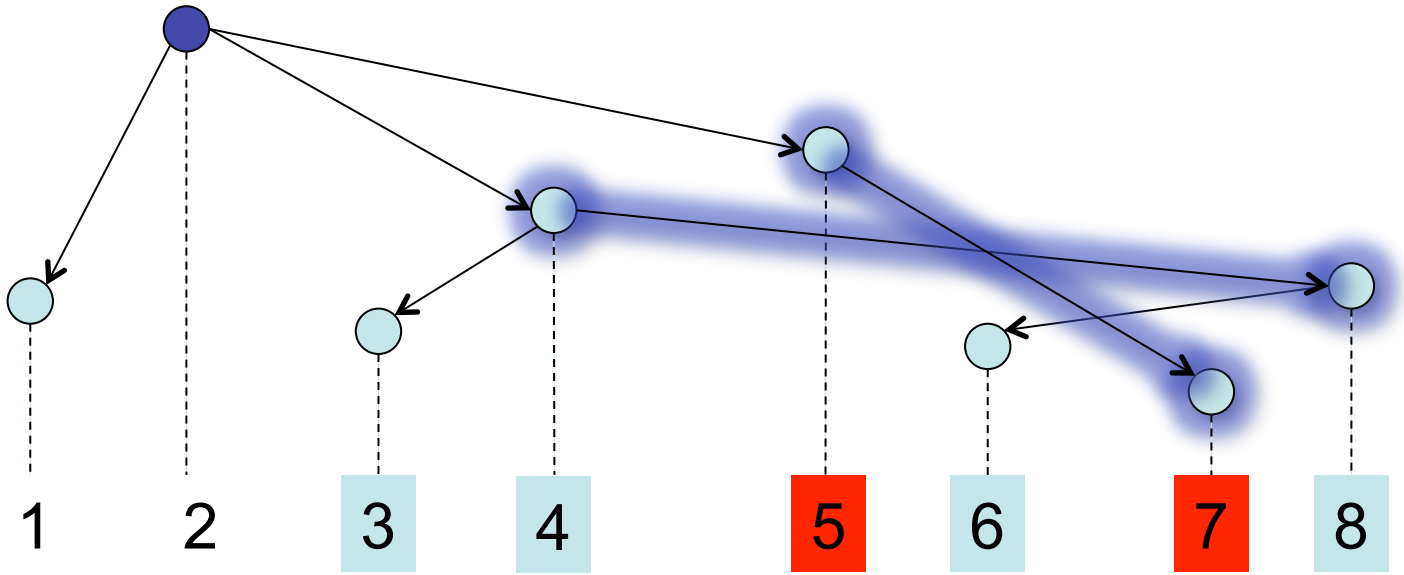
- Block degree measures non-projectivity
- Block degree = number of intervals per sub-tree, e.g. { [3,4], [6,8] } is block degree 2



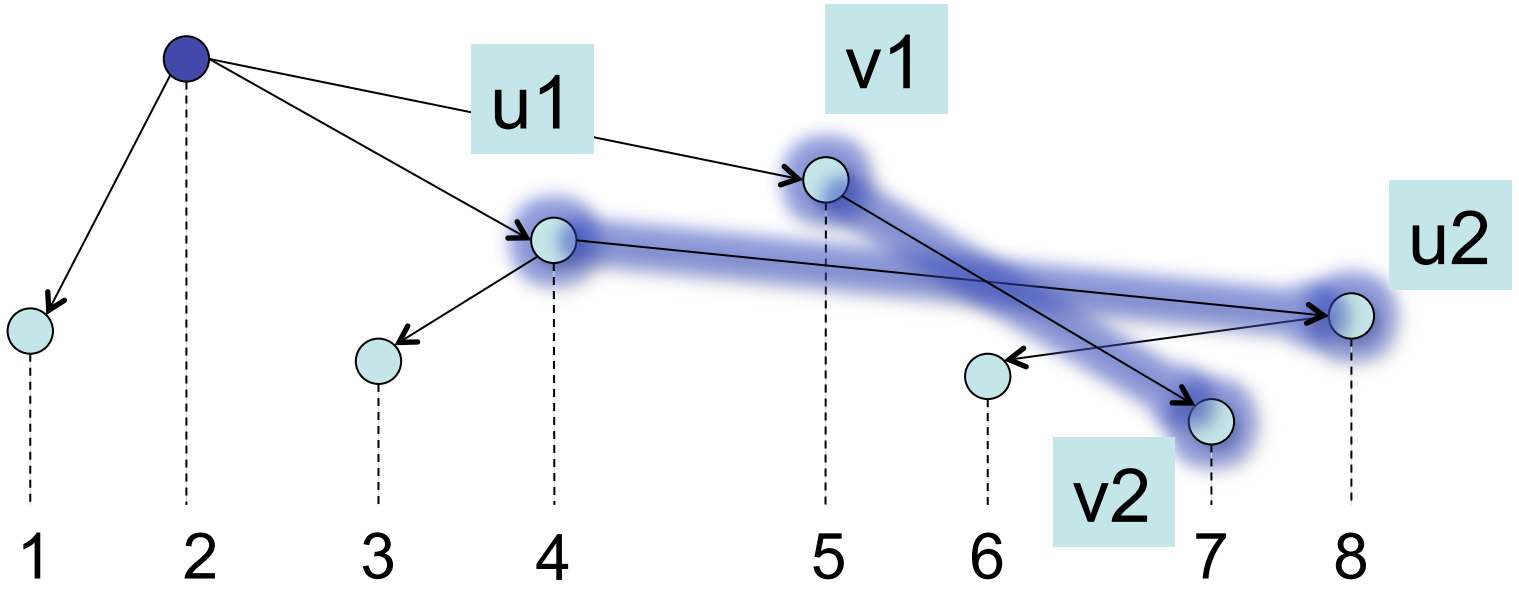
Ill-nested Dependencies



Ill-nested Dependencies

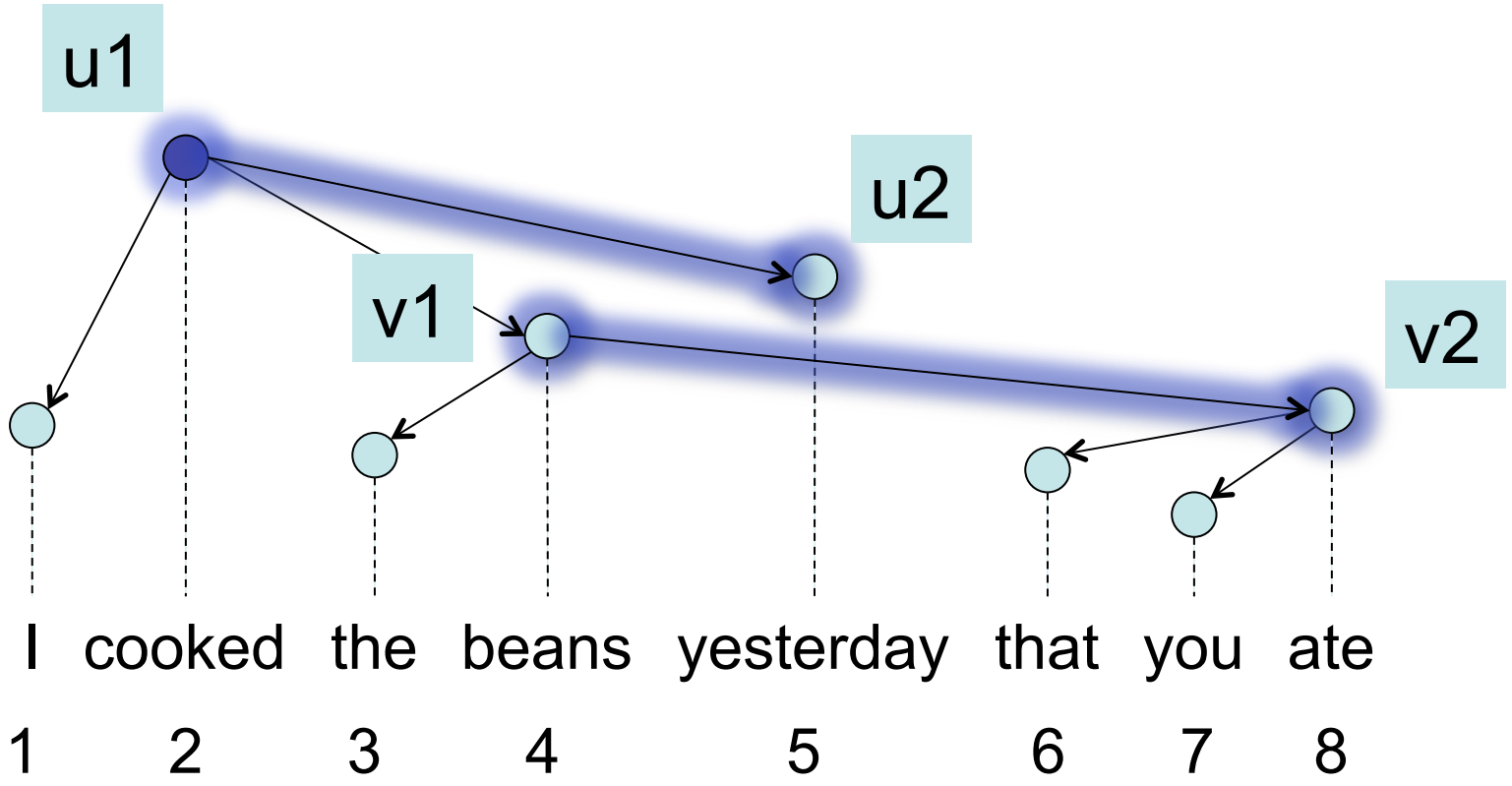


Ill-nested Dependencies



- Well-nested dependencies: constraint on pairs of edges
- If u1 < u2 and v1 < v2 and v1 < u2 then u1 must dominate v1 or vice versa

Well-nested Dependencies



- In this example, u1 < v1 < u2 < v2
- And u1 dominates v1 and u2

Block Degree and Well-nestedness

- Prague Dependency Treebank 1.0 with 73,088 sentences.
- 99.48% of sentences have a block degree of 2 and are well-nested.
- 0.52% are either ill-nested or block degree is greater than 2.
- Similar analysis for other languages.
- Results from Marco Kuhlmann's thesis

Generative Capacity
(Crossing Dependencies)

Formalism	Dependencies
CFG	projective
TAG	Block degree ≤ 2 and Well-nested
Coupled CFG	Block degree ≤ k and Well-nested
LCFRS	Block degree ≤ k

Motivations for TAG

- NL is probably not weakly context-free, but also possibly not fully context-sensitive and so NL could be **mildly context-sensitive**
- Strong **lexicalization** of CFGs leads to the adjunction operation
- Some NLs clearly show **crossing dependencies**, but maybe of limited variety
- Many issues of **locality** in linguistics, e.g. constraints on long-distance dependencies, can be encoded within the notion of elementary tree

Tree-Adjoining Grammars: Application to Natural Language

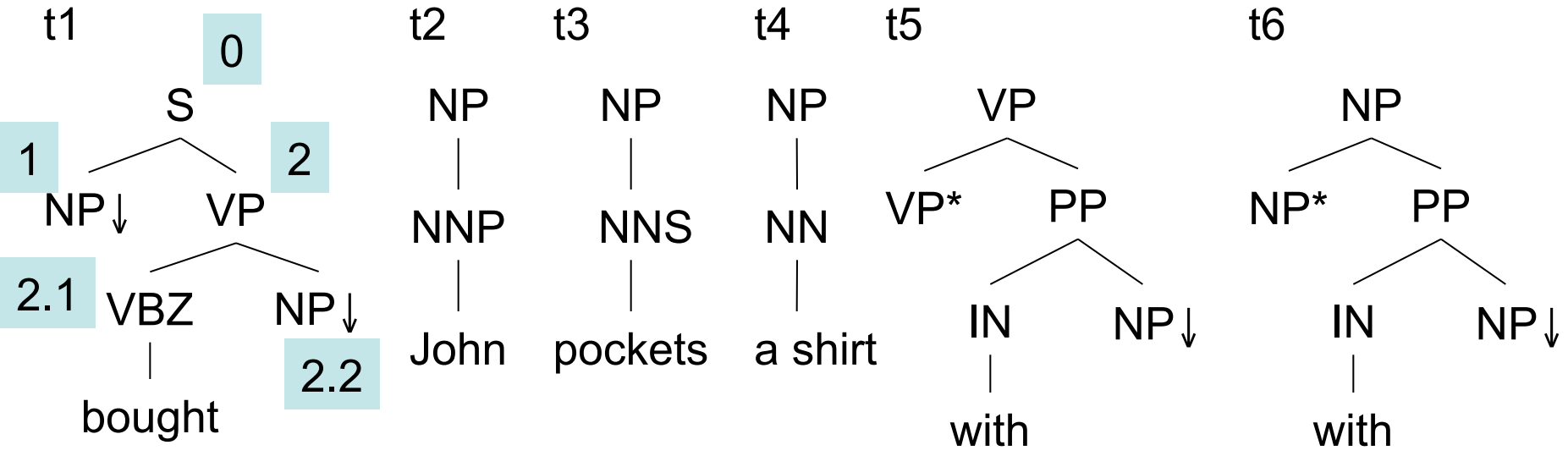
Lexicalized TAG

- A Lexicalized TAG (LTAG) is a TAG where each elementary tree has at least one terminal symbol as a leaf node
- Lexicalization has some useful effects:
  - finite ambiguity: corresponds to our intuition about NL ambiguities,
  - statistical dependencies between words can be captured which can improve parsing accuracy

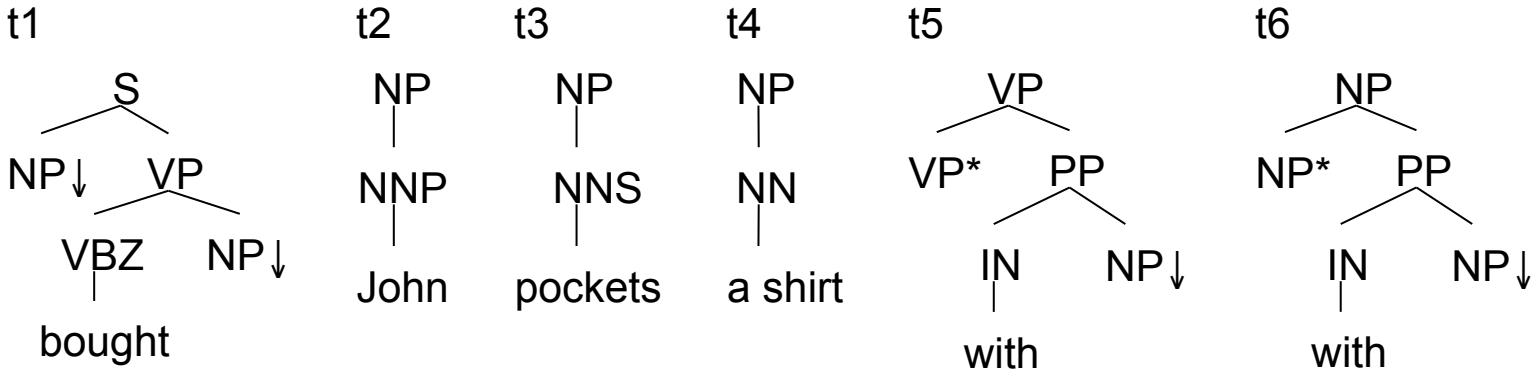
Lexicalized TAG

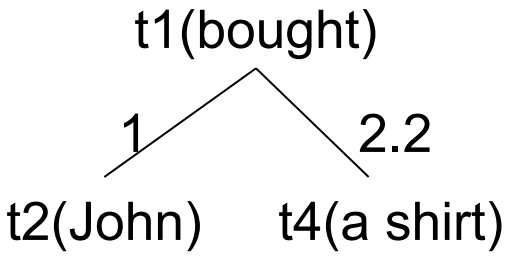
- TAGs do not need to be fully lexical
  - Tree to string rules are sometimes unlexicalized
  - Relative clause formation can be done using unlexicalized trees

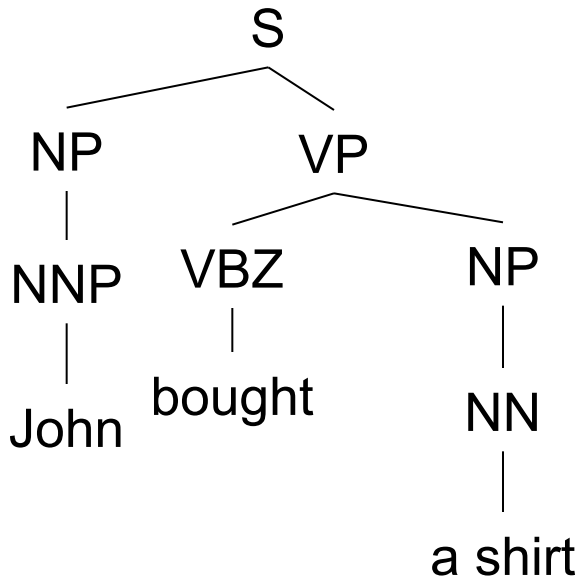
Lexicalized TAG: example



Gorn tree address: an index for each node in the tree

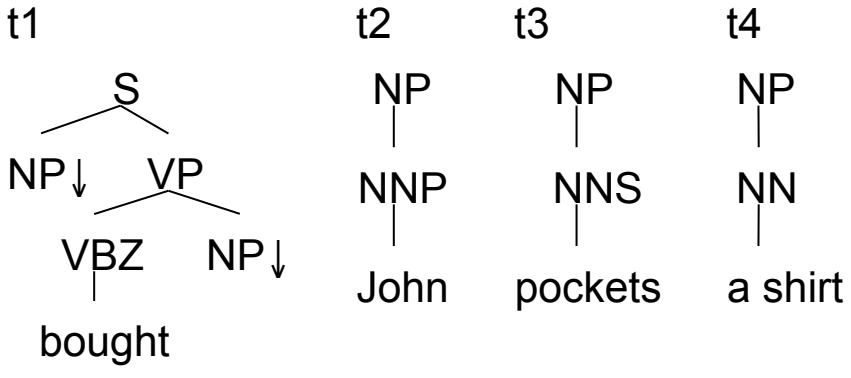


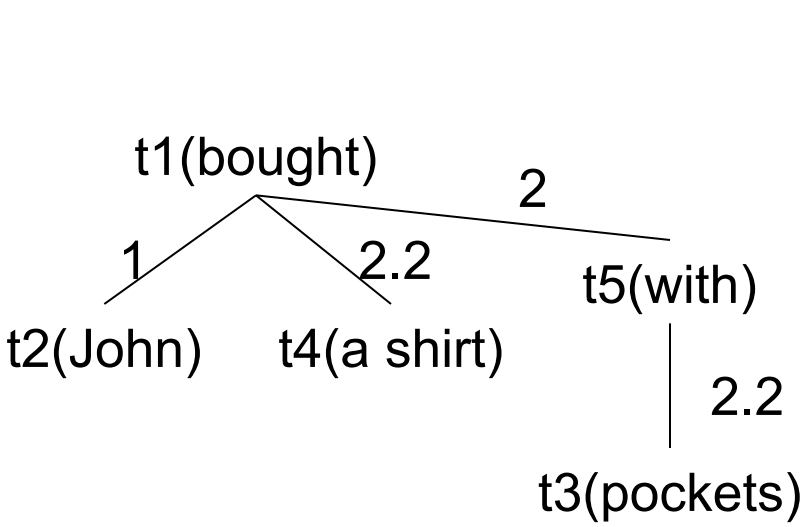


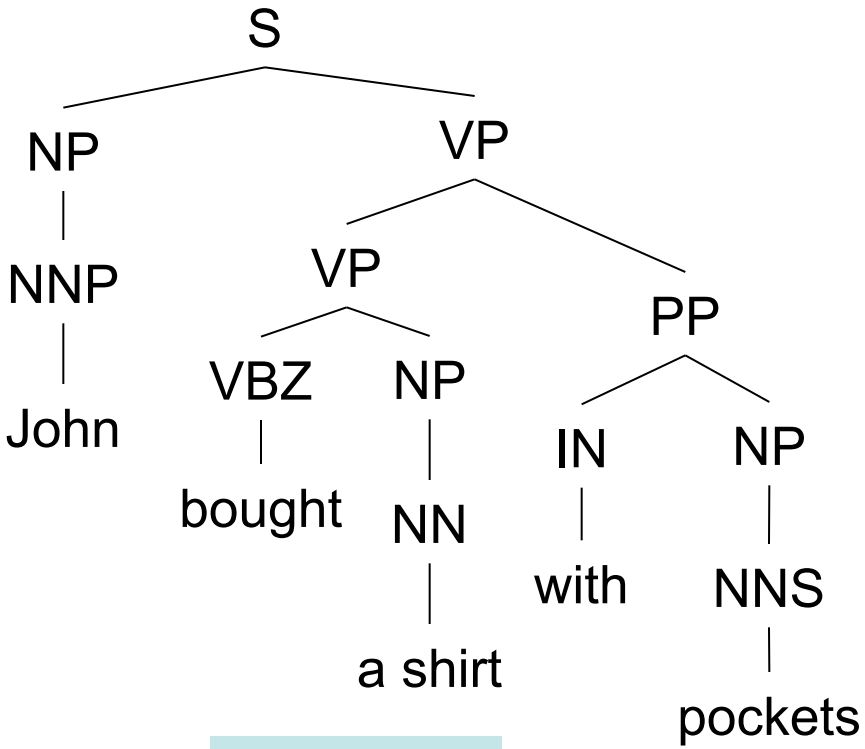


derivation tree

derived tree

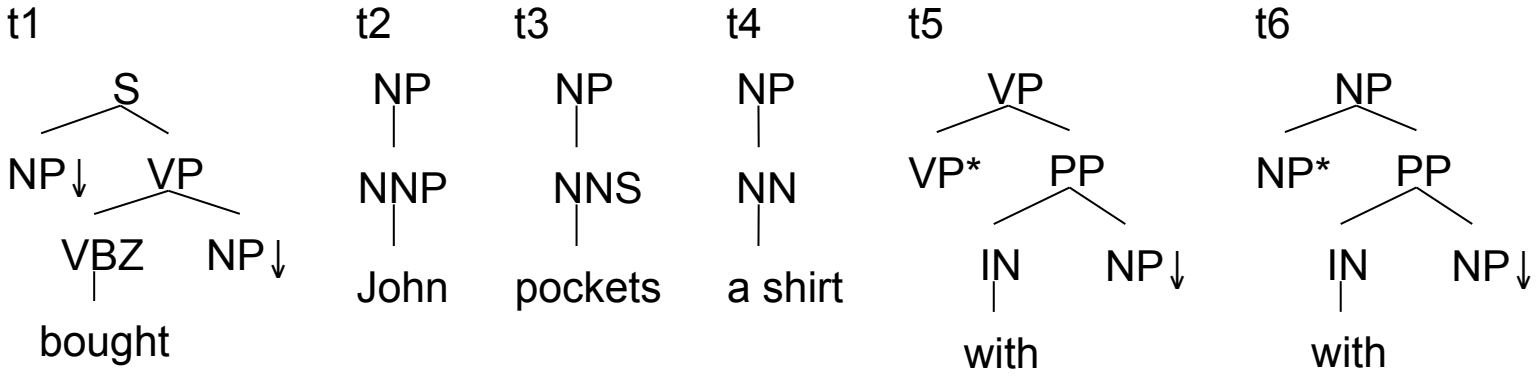


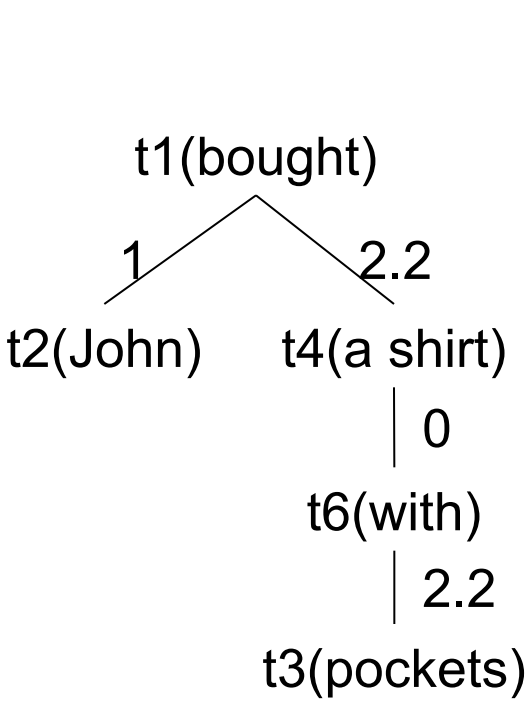




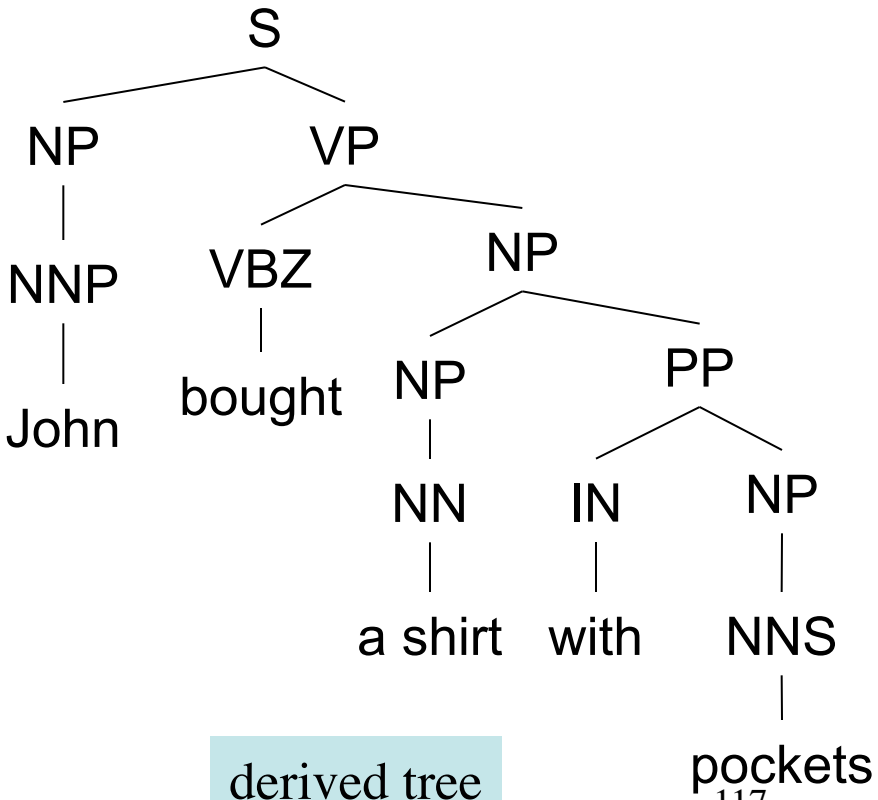
derivation tree

derived tree





derivation tree



derived tree

Comparison with Dependency Grammar

- Compare the derivation tree with the usual notion of a dependency tree
- Note that a TAG derivation tree is a formal representation of the derivation
- In a Lexicalized TAG, it can be interpreted as a particular kind of dependency tree
- Different dependencies can be created by changing the elementary trees
- LTAG derivations relate dependencies between words to detailed phrase types and constituency

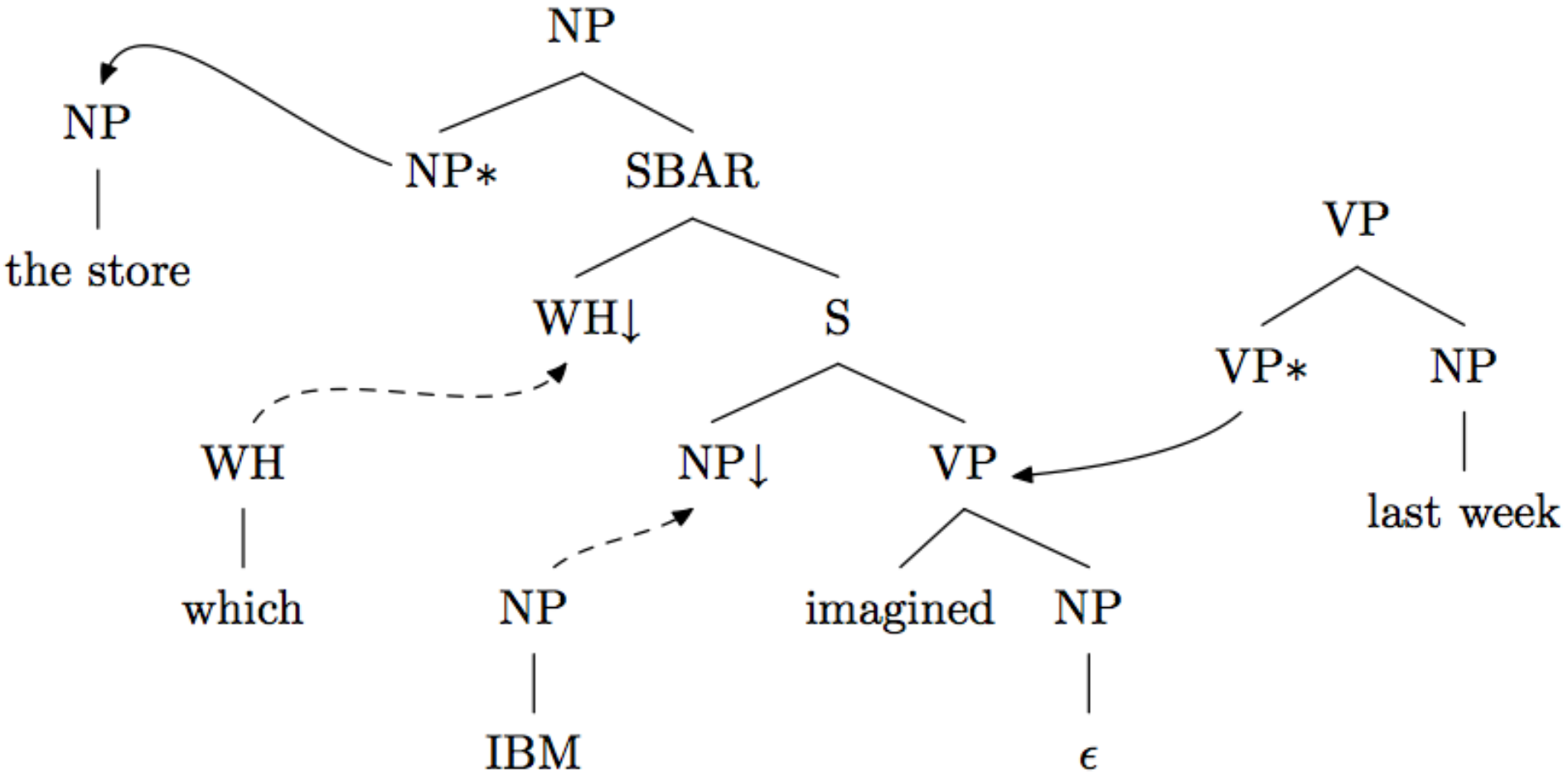
Localization of Dependencies

- Syntactic
  - agreement: person, number, gender
  - subcategorization: *sleeps* (null), *eats* (NP), *gives* (NP NP)
  - filler-gap: *who*$_i$ *did John ask Bill to invite* t_i
  - word order: within and across clauses as in scrambling, clitic movement, etc.

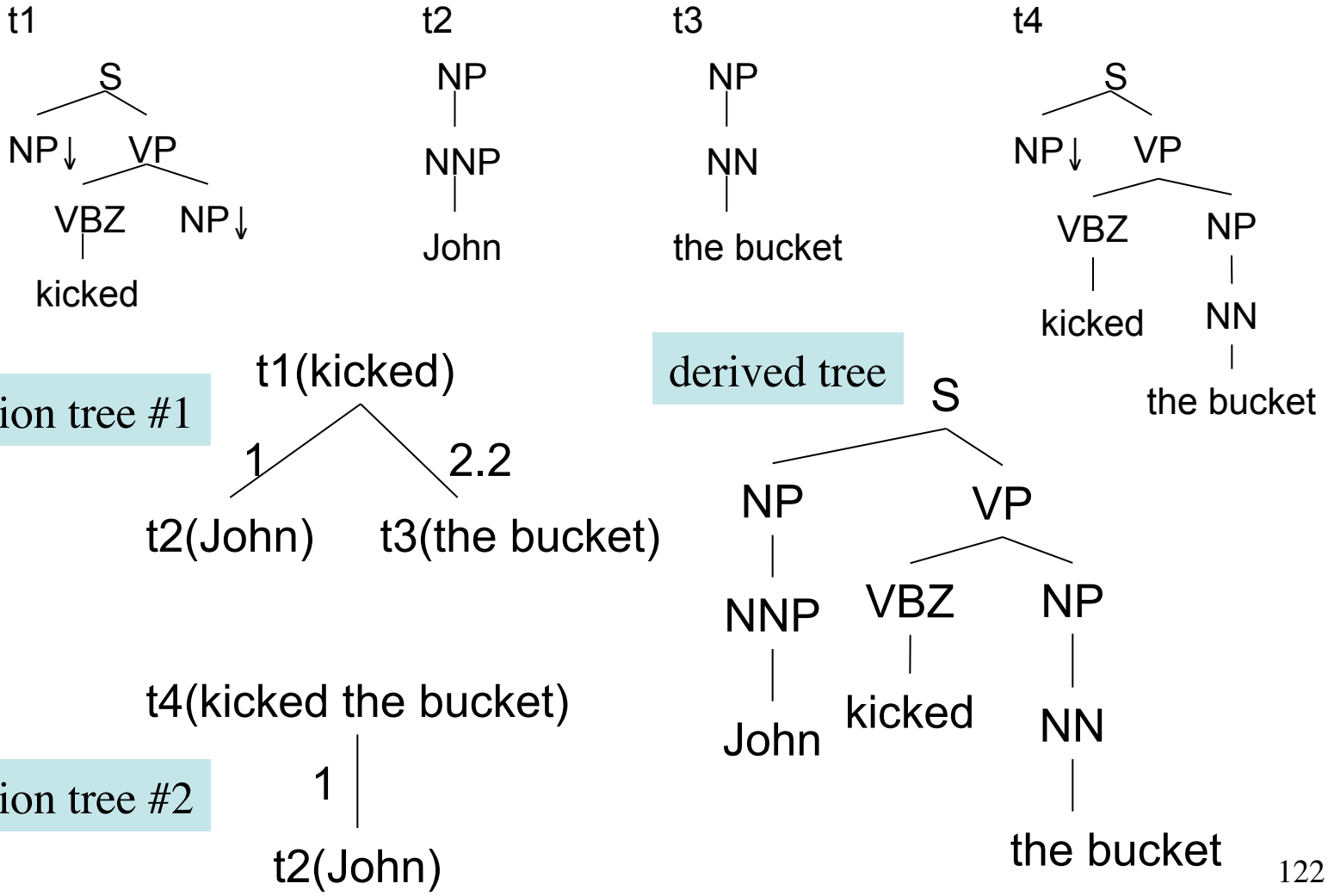
Localization of Dependencies

- Semantic
  - function-argument: all arguments of the word that lexicalizes the elementary tree (also called the *anchor* or *functor*) are localized
  - word clusters (word idioms): non-compositional meaning, e.g. *give a cold shoulder to, take a walk*
  - word co-occurences, lexical semantic aspects of word meaning

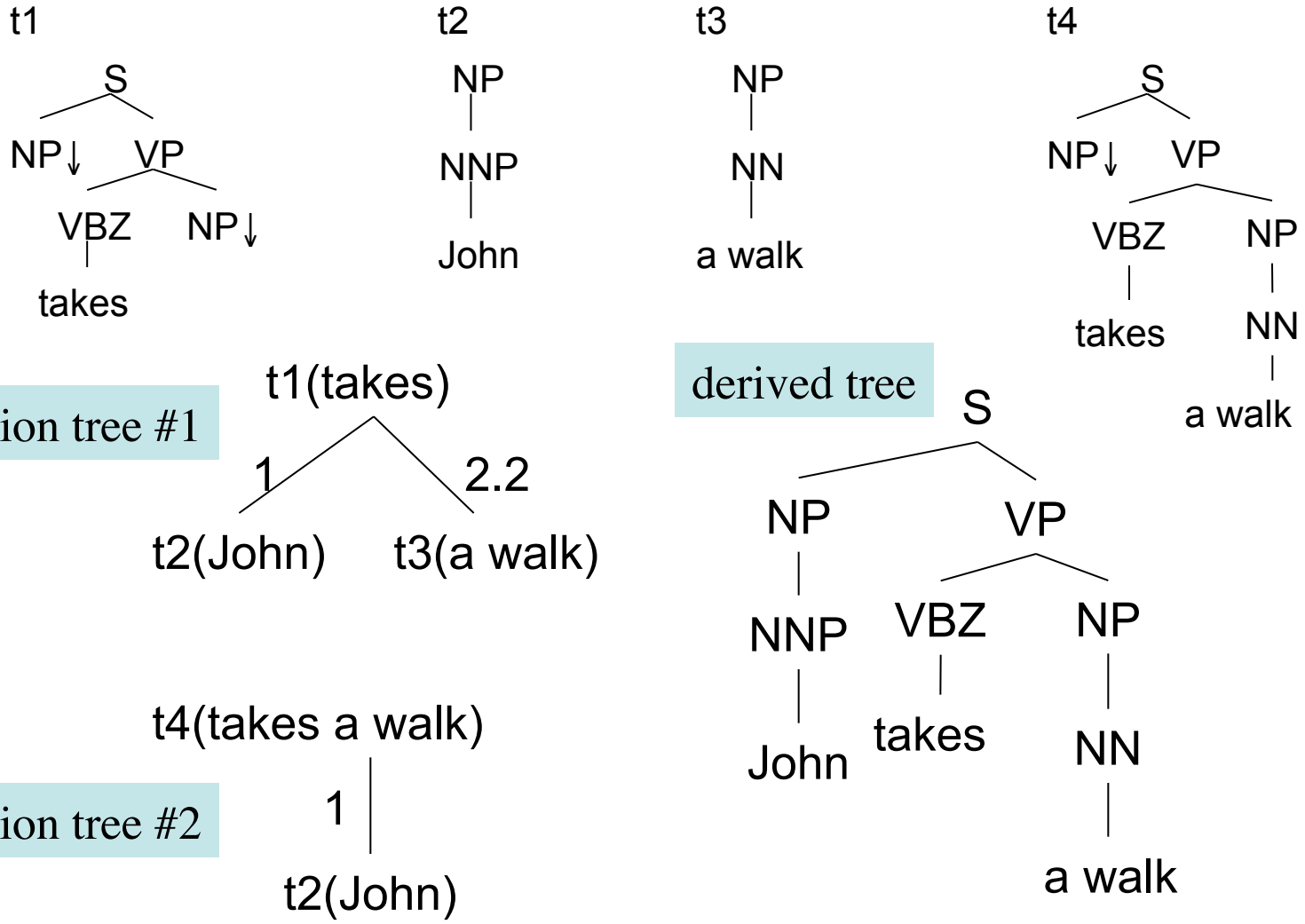
Localization of Dependencies



Idioms


t1
S
NP↓
VP
VBZ
NP↓
kicked
t2
NP
NNP
John
t3
NP
NN
the bucket
t4
S
NP↓
VP
VBZ
NP
kicked
NN
the bucket
derivation tree #1
t1(kicked)
1
2.2
t2(John)
t3(the bucket)
derived tree
S
NP
VP
NNP
VBZ
NP
John
kicked
NN
the bucket
derivation tree #2
t4(kicked the bucket)
1
t2(John)

Phrasal/Light Verbs



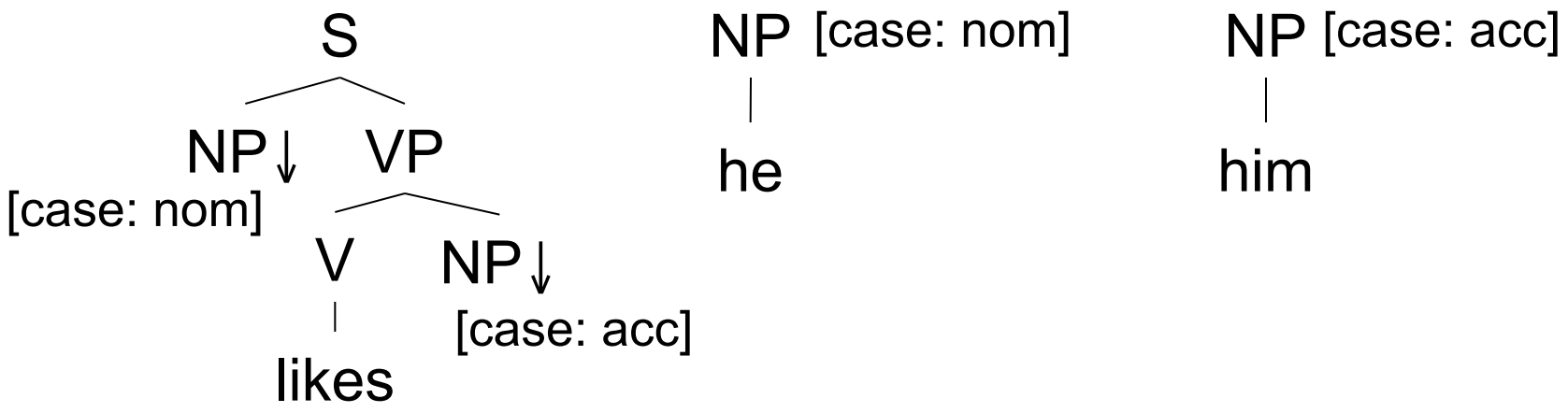
TAG and Generation

- TAG has some useful properties with respect to the problem of NL generation
- Adjunction allows a generation planning system to add useful lexical information to existing constituents
- Makes planning for generation output more flexible
  - e.g. if the system has a constituent *the book*, it can choose to add new information to it: *the red book*, if there is a distractor for that entity for the hearer
- cf. Matthew Stone's SPUD system

TAG with feature structures

TAG with feature structures

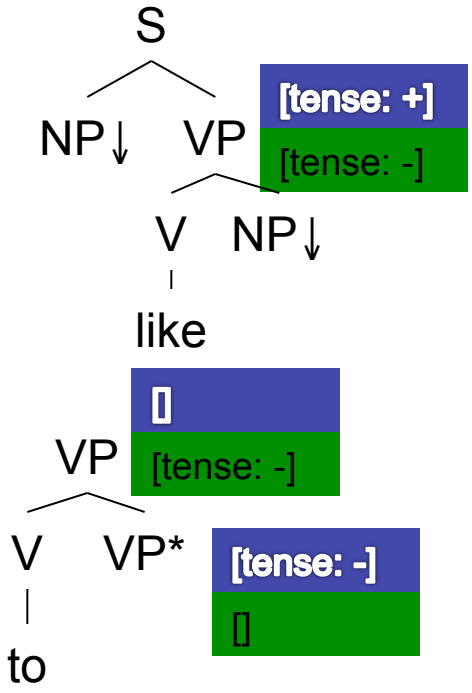
We want to rule out strings like **him likes he*

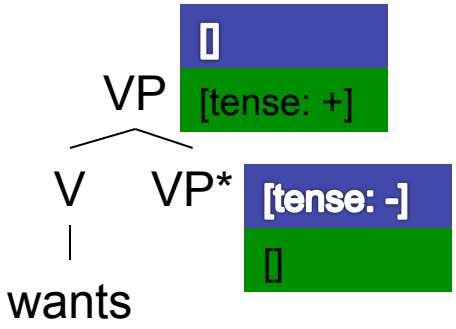


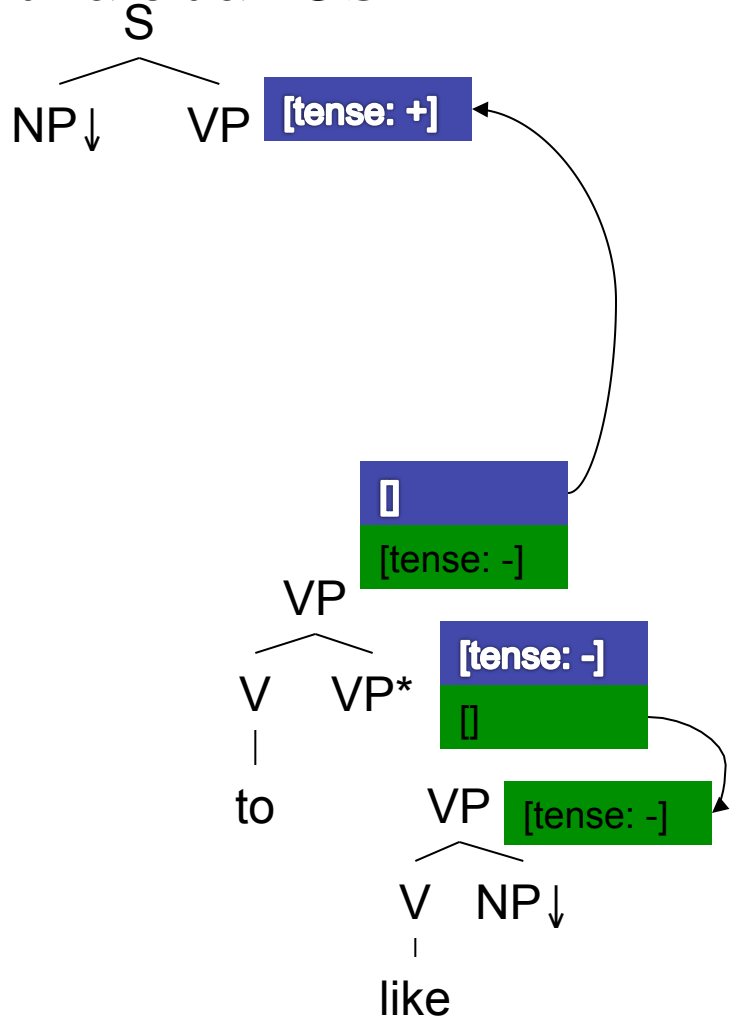
- Use feature structure unification:
- [f:+] ∪ [f:-] = fail, [f:+] ∪ [f:+] = [f:+]
- [f:+] ∪ [f:] = [f:+], [f:+] ∪ [g:+] = [f:+, g:+]
- unification of two feature structures = least most general feature structure that *subsumes* both

TAG with feature structures

We want to derive *he wants to like him*

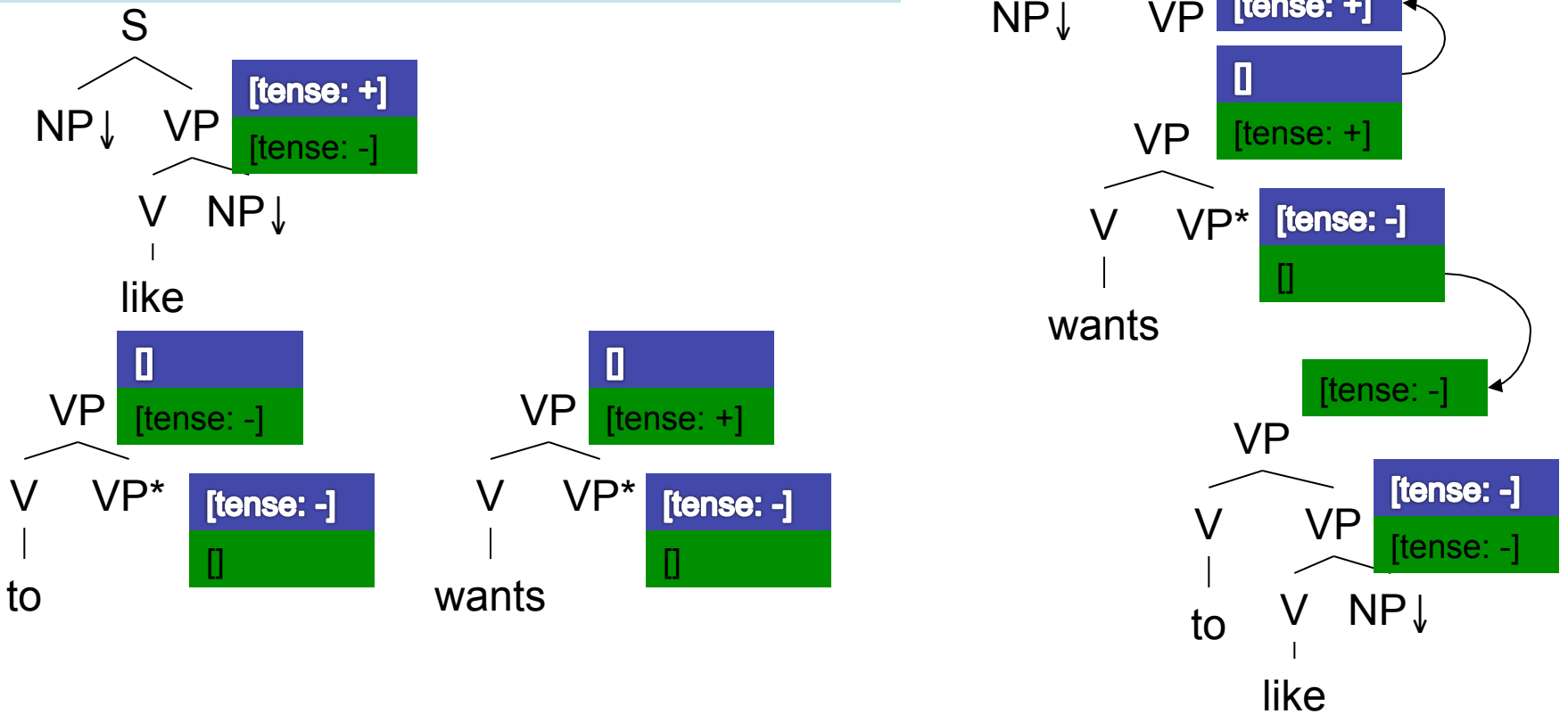






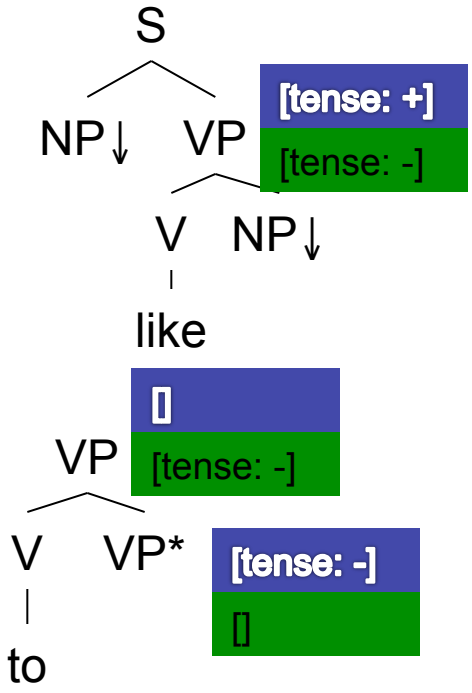
TAG with feature structures

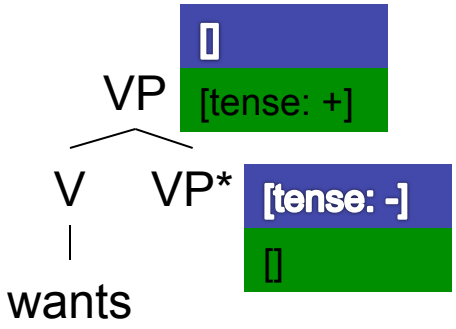
We want to derive *he wants to like him*

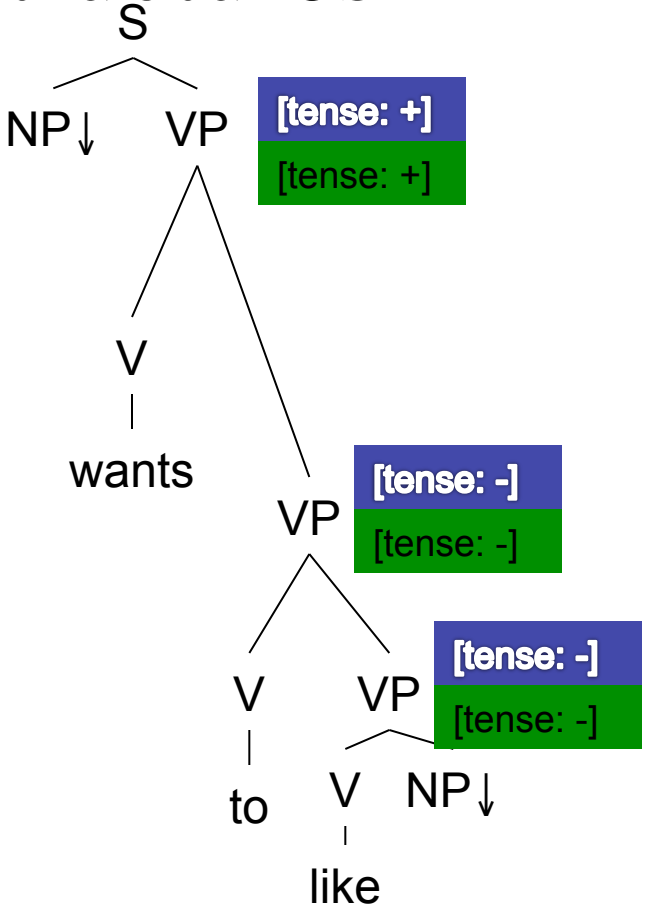


TAG with feature structures

We want to derive *he wants to like him*



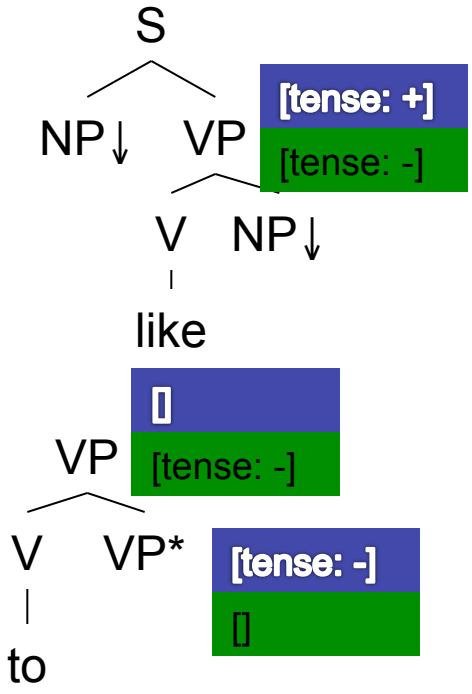


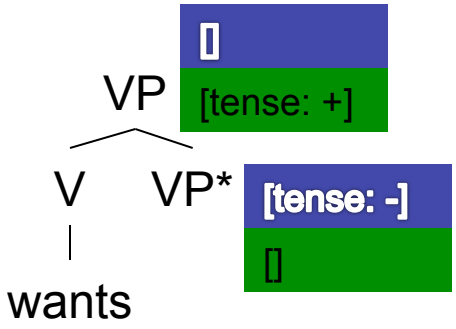


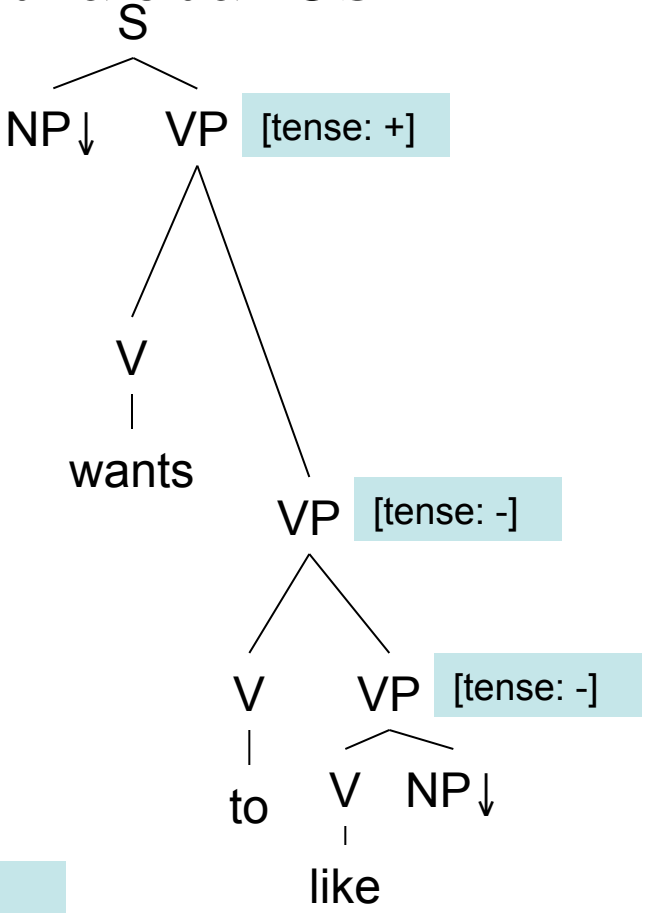
Top and bottom features must unify for a derivation to be valid.

TAG with feature structures

We want to derive *he wants to like him*







Adjunction can be exploited to ensure that the feature structures never grow to an arbitrary size unlike context-free based unification grammars.

Mapping a TreeBank into Lexicalized TAG derivations

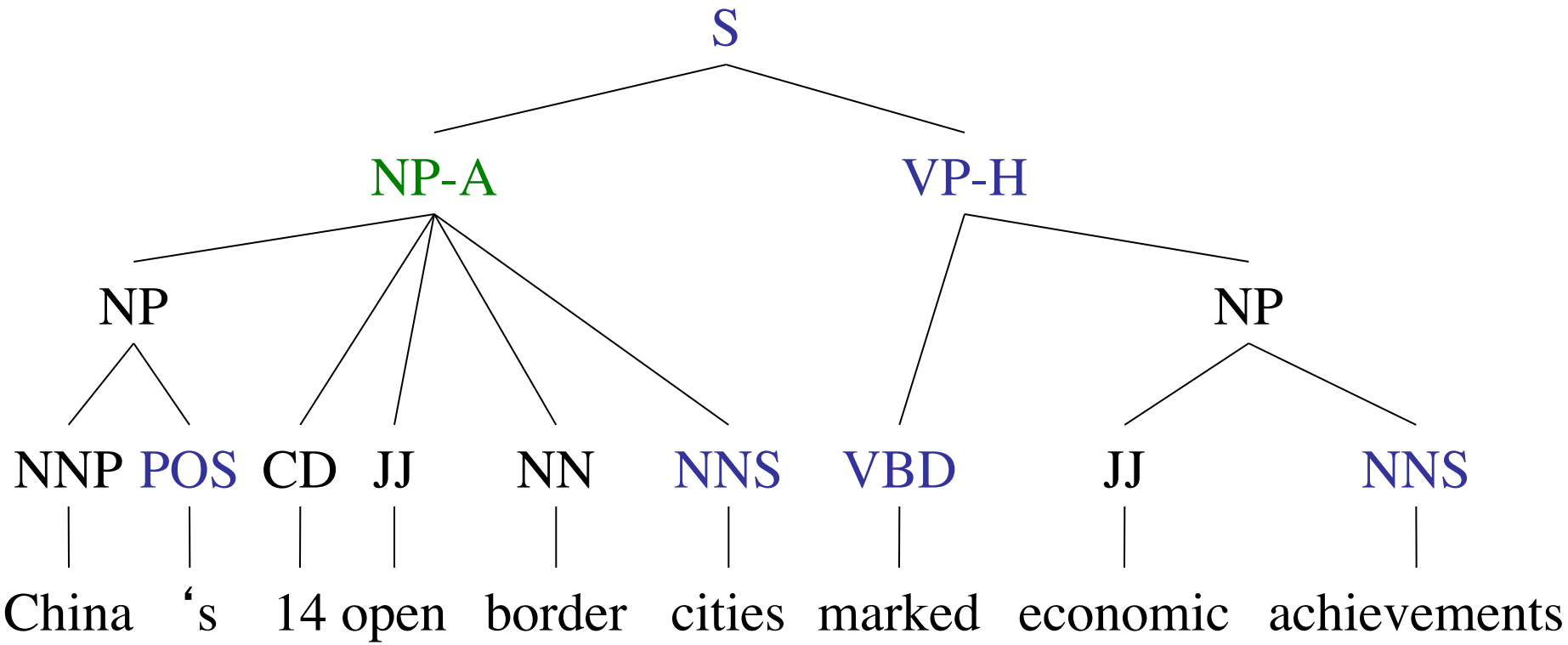
Penn Treebank for English

- Training data: 40K sentences with the correct syntactic phrase-structure tree
- Test data: ~2K sentences
- Evaluation: labeled constituents precision and recall
- Simple PCFGs can get ~84 F1 score.
- Also, Treebanks and TAG parsers exist for other languages: Chinese, Korean, German, French, …

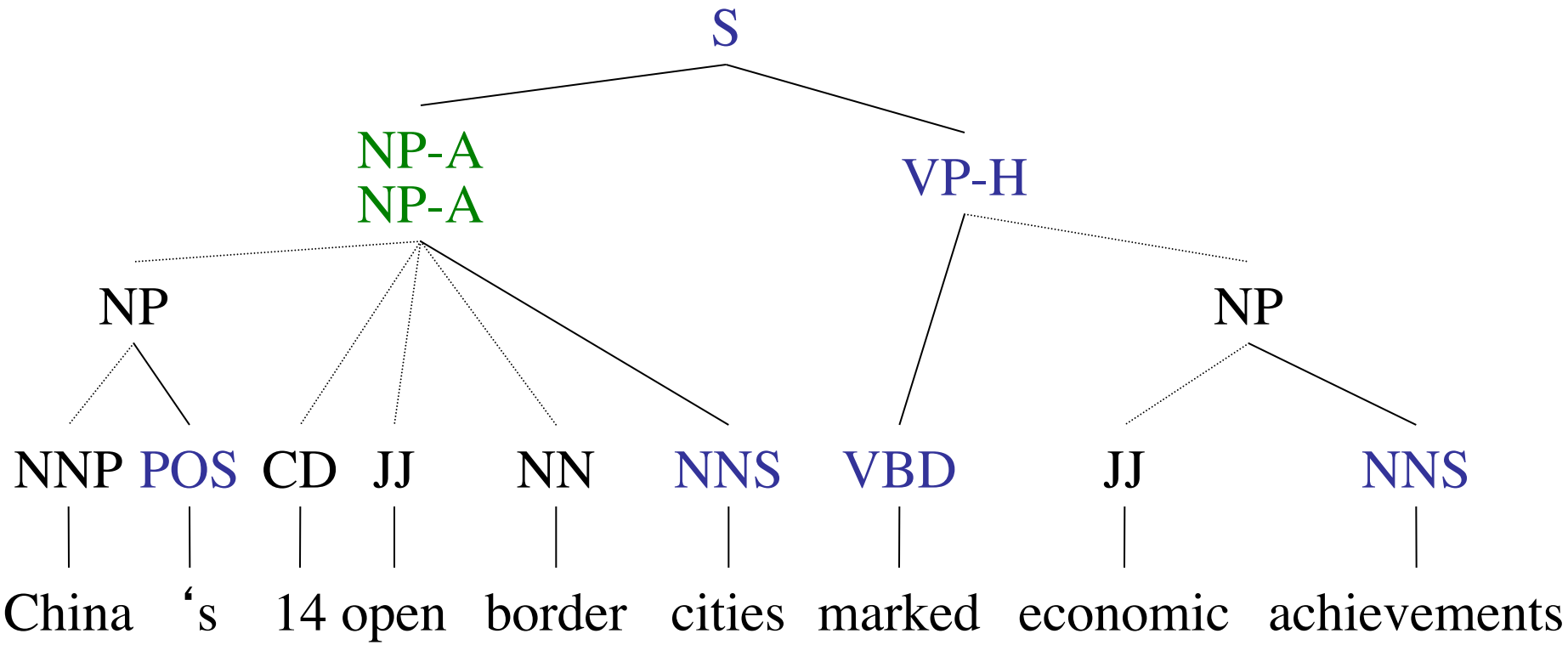
LTAG Derivations from TreeBanks

- TreeBanks contain phrase structure trees or dependency trees
- Converting dependency trees into LTAG is trivial
- For phrase structure trees: exploit head percolation rules (Magerman, 1994) and argument-adjunct heuristic rules
- First mark TreeBank tree with head and argument information
- Then use this information to convert the TreeBank tree into an LTAG derivation
- More sophisticated approaches have been tried in (Xia, 1999) (Chiang, 2000) (Chen, 2000) and (Shen, 2006)

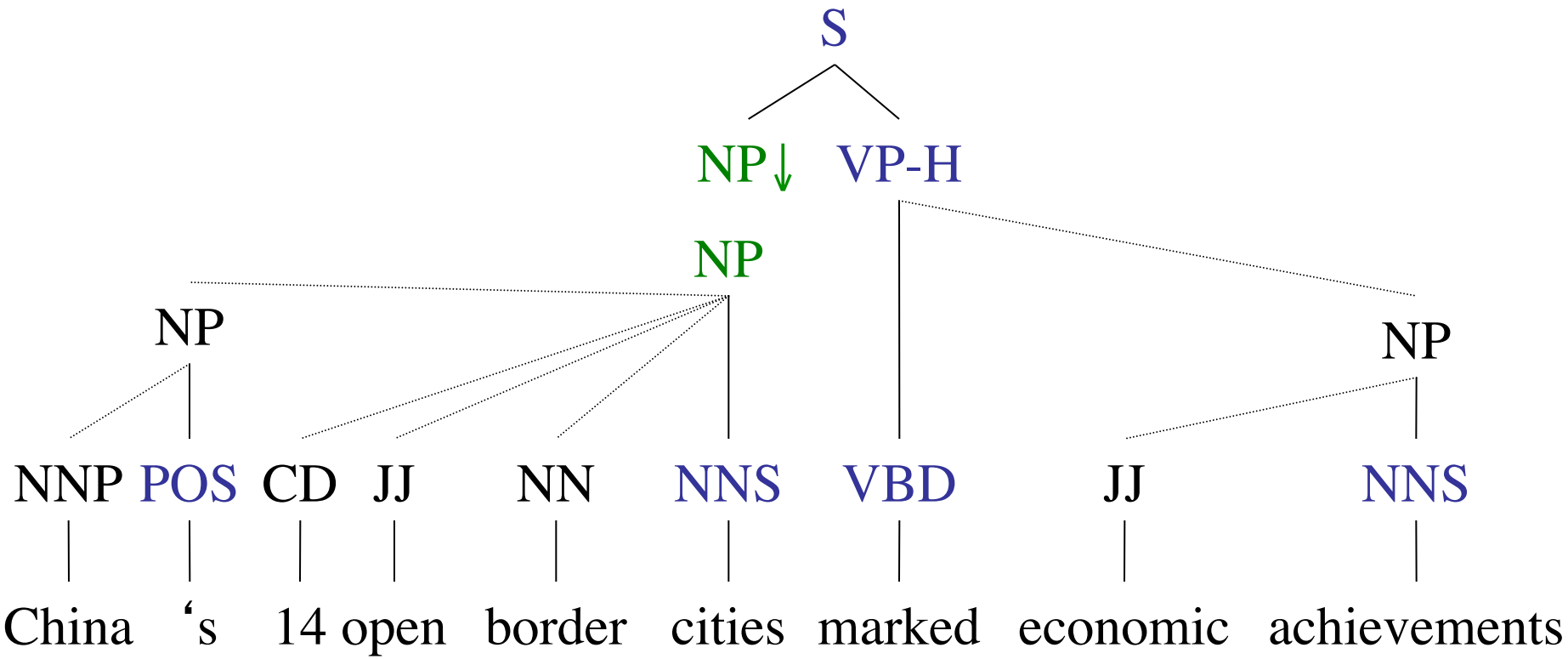
LTAG derivations from TreeBanks



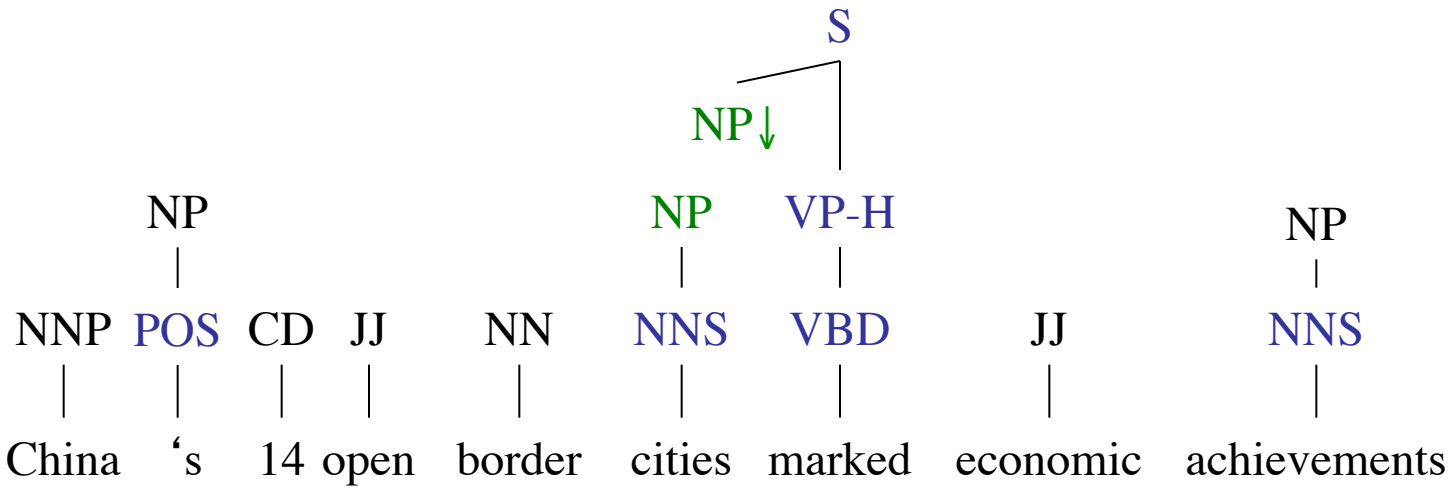
LTAG derivations from TreeBanks

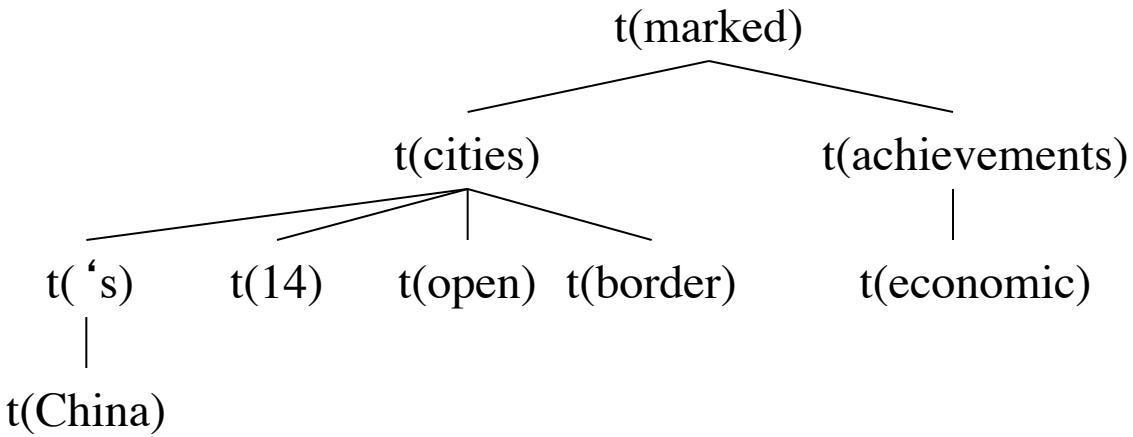


LTAG derivations from TreeBanks



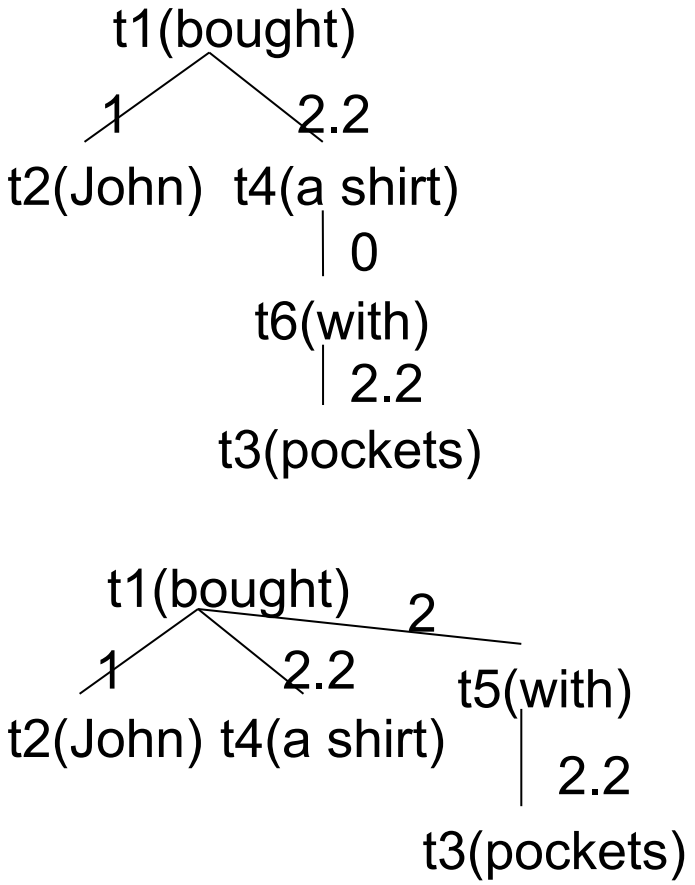
LTAG derivations from TreeBanks





Ambiguity Resolution

Ambiguity Resolution



- Two possible derivations for ***John bought a shirt with pockets.***
- One of them is more plausible than the other.
- Statistical parsing is used to find the most plausible derivation.

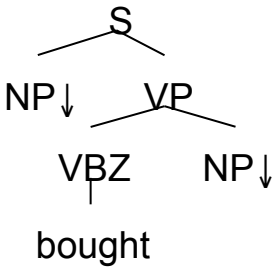
Statistical parsing

- Statistical parsing = ambiguity resolution using machine learning
- S = sentence, T = derivation tree
- Find best parse: $\arg\max_{T} P(T,S)$

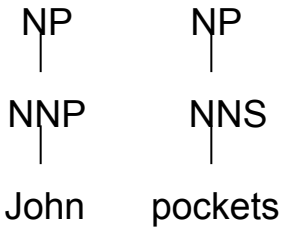
P(T,S) is a generative model: it contains parameters that generate the input string

Statistical Parsing with TAG

t1

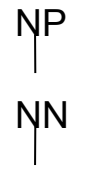


t2

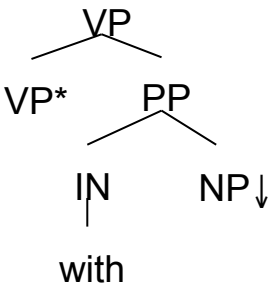


t3

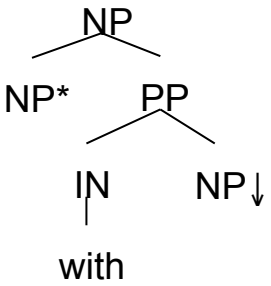
t4

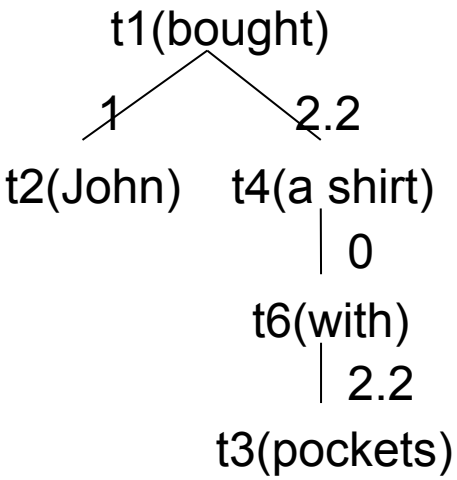


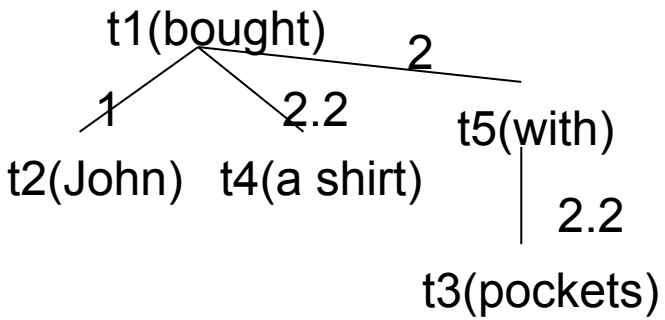
t5



t6







P(tree1, *John bought a shirt with pockets*)=

P(t1) * P(t1#0, NONE | t1#0) *
P(t1#1, t2 | t1#1) *
P(t1#2.2, t4 | t1#2.2) *
P(t1#2, NONE | t1#2) *
P(t2#0, NONE | t2#0) *
P(t4#0, t6 | t4#0) *
P(t6#0, NONE | t6#0) *
P(t6#2, NONE | t6#2) *
P(t6#2.2, t3 | t6#2.2) *
P(t3#0, NONE | t3#0)

P(tree2, *John bought a shirt with pockets*)=

P(t1) * P(t1#0, NONE | t1#0) *
P(t1#1, t2 | t1#1) *
P(t1#2.2, t4 | t1#2.2) *
P(t1#2, t5 | t1#2) *
P(t2#0, NONE | t2#0) *
P(t4#0, NONE | t4#0) *
P(t5#0, NONE | t5#0) *
P(t5#2, NONE | t5#2) *
P(t5#2.2, t3 | t5#2.2) *
P(t3#0, NONE | t3#0)

Statistical Parsing with TAG

$$\arg\max_{T} P(T,S)$$

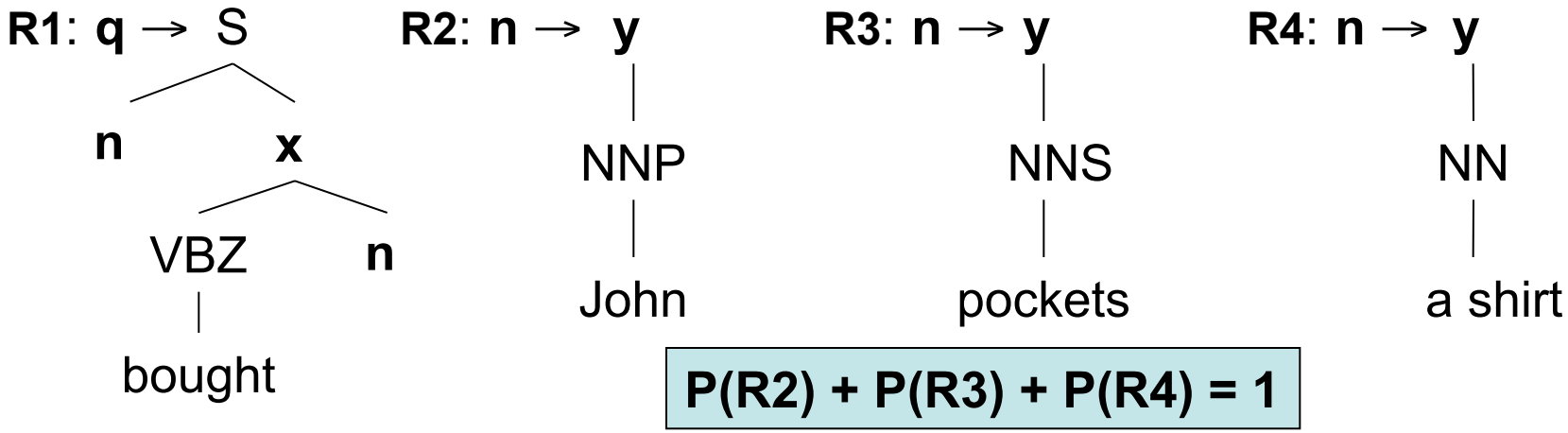
- PCFG
- Let tree T be built out of *r* CFG rules
- Note that in both PCFG and Prob. TAG, T is the *derivation tree*
- (in contrast with DOP models)
- Find all T for given S in $O(G^2n^3)$
- For lexicalized CFG: $O(n^5)$

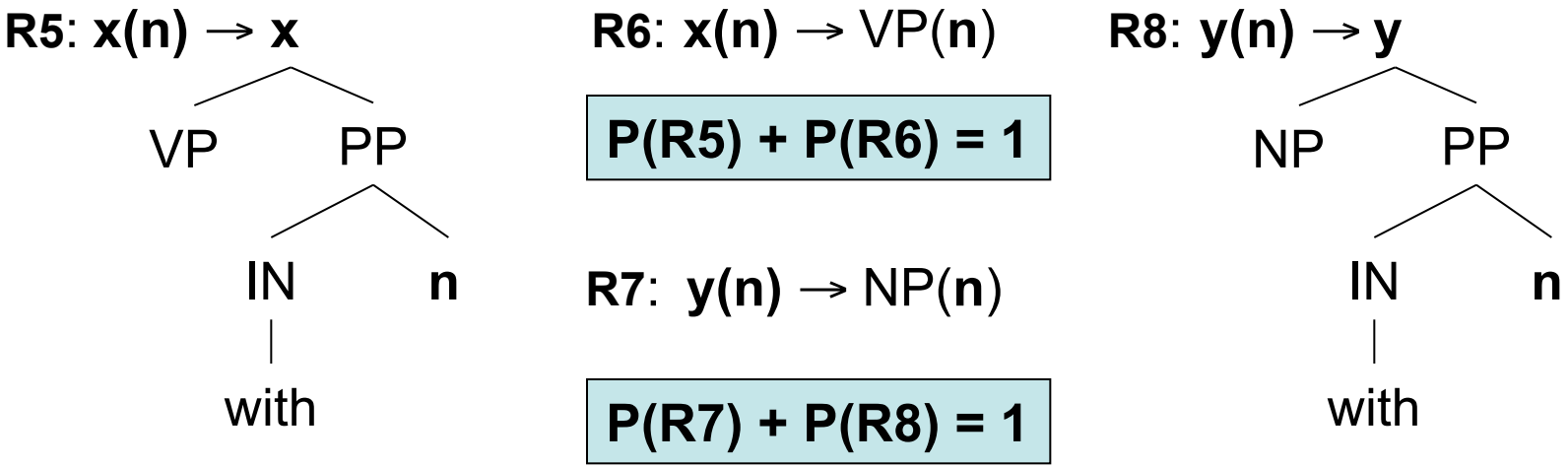
$$P(T,S) = \prod_{i=1}^{r} P(LHS_i \rightarrow RHS_i \mid LHS_i)$$

- Prob. TAG
- Let tree T be built using *r* elementary trees, $t_1 \ldots t_r$
- Let there be *s* nodes where substitution can happen
- And *a* nodes where adjunction can happen
- Find all T for given S in $O(n^6)$

$$P(T,S) = p(i) \times \prod_{i=1}^{s} P(i,t \mid i) \times \prod_{j=1}^{a} P(j,\{t,\text{NONE}\} \mid j)$$

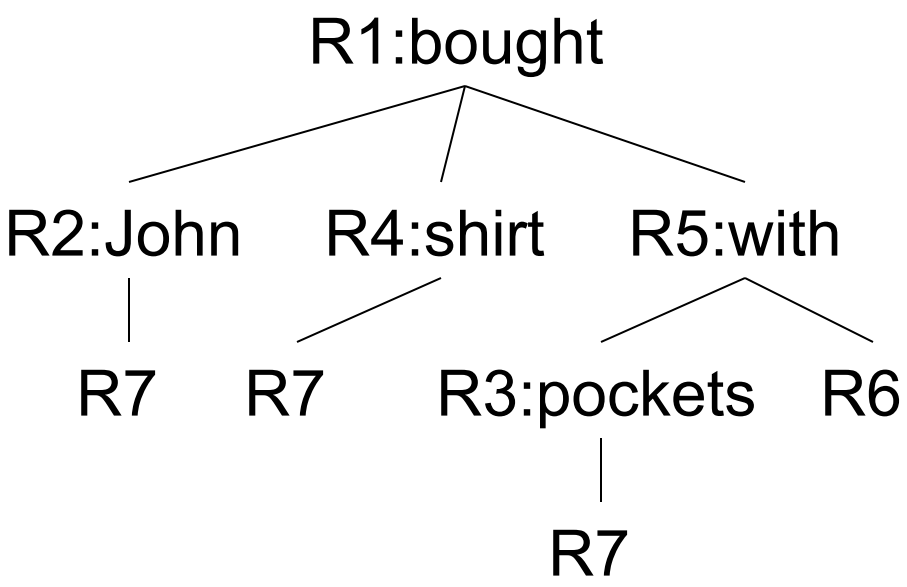
Statistical Parsing with ATGs



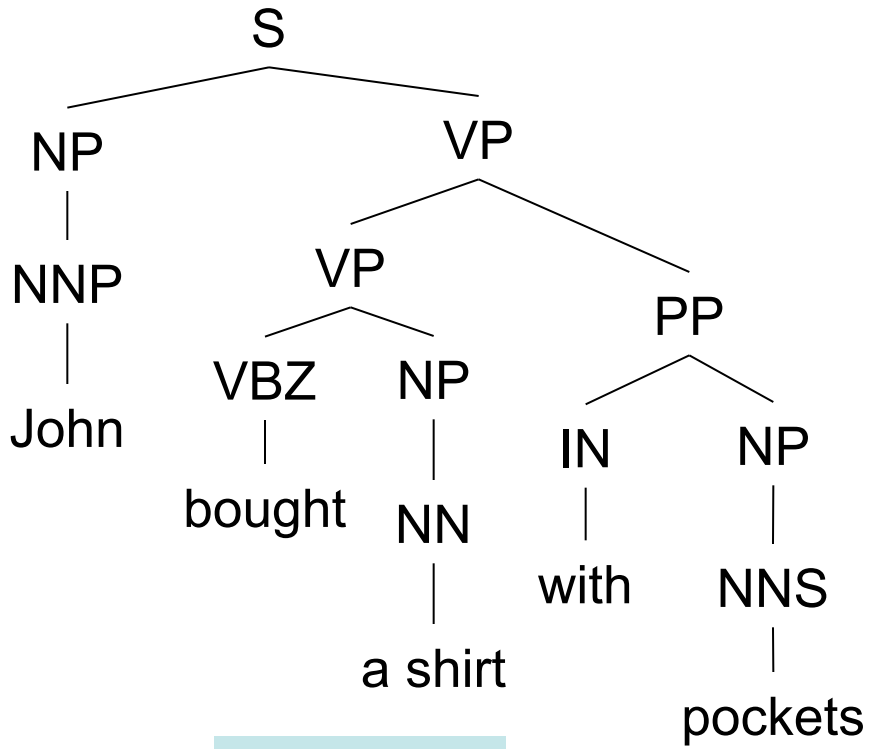


Probabilities do not have to be bi-lexical!

P(R1) * P(R2) * P(R4) * P(R5) * P(R3) * P(R6) * (3 * P(R7))

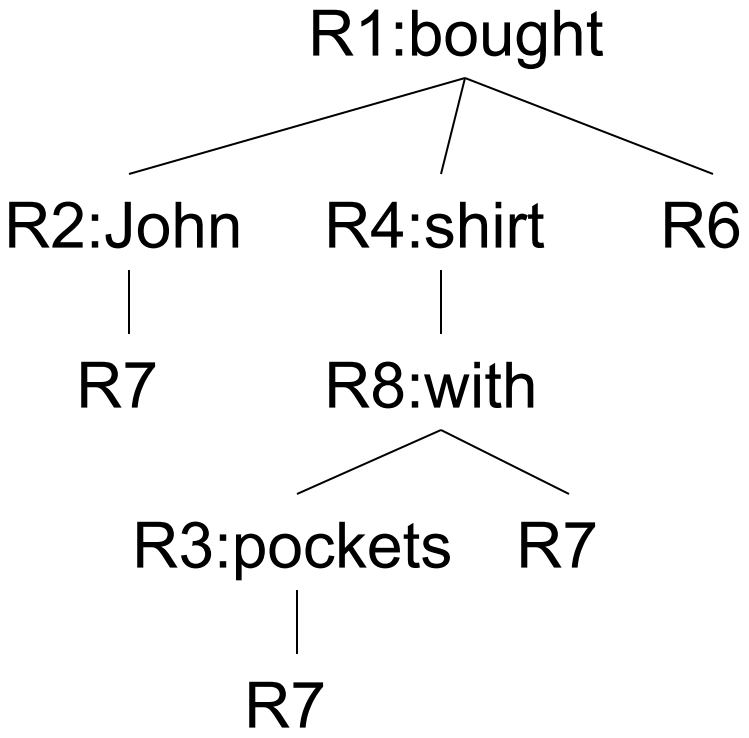


derivation tree

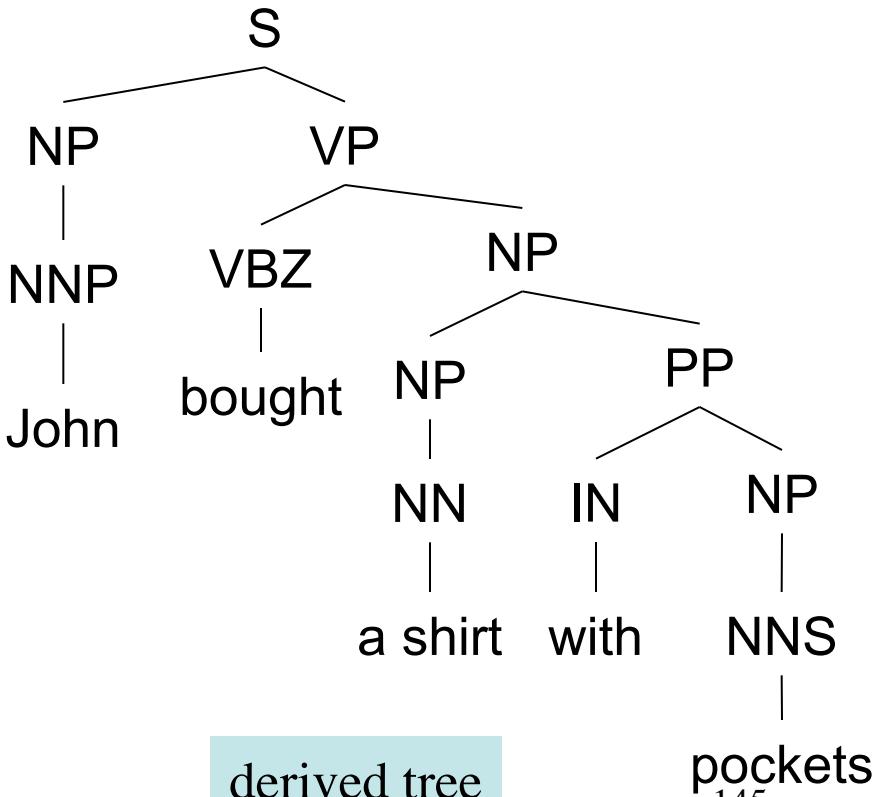


derived tree

P(R1) * P(R2) * P(R4) * P(R8) * P(R3) * P(R6) * (3 * P(R7))



derivation tree



derived tree

Statistical Parsing with TAG

(Carreras, Collins and Koo 2008)

CoNLL 2008 Best Paper

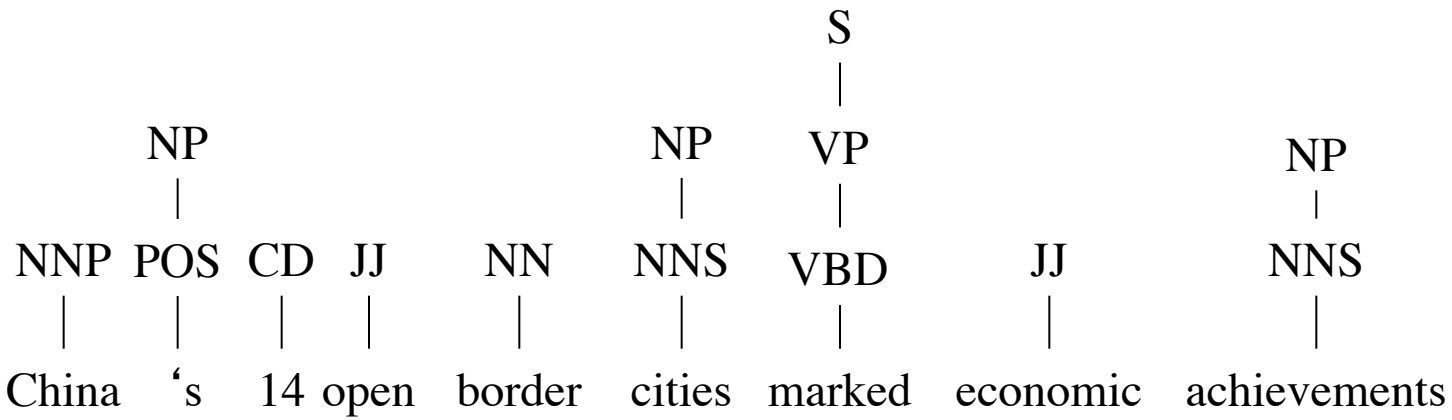
TAG-based discriminative parsing

- State of the art discriminative parsing using TAG
- Combines dependency parsing algorithms with phrase-structure parsing algorithms
- Does better on both phrase-structure and dependency evaluation
  - Phrase structure: 91.1 F1 (versus best 91.7 F1)
  - Dependencies: 93.5% (versus 92.5%)

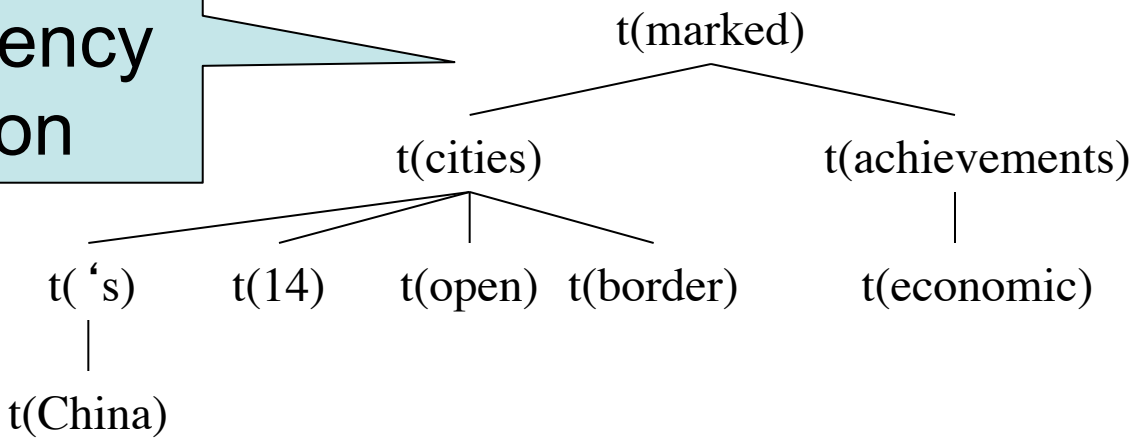
TAG-based discriminative parsing

- Parse trees y for a given sentence x
- Each tree y comes from a set Y(x)
- Training: learn a weight vector **w** using perceptron updates on training data
- Representation: one weight per feature in a feature vector **f**(**x**, y)
- Discriminative parsing at test time: find the y in Y(x) that maximizes **w** . **f**(**x**,y)

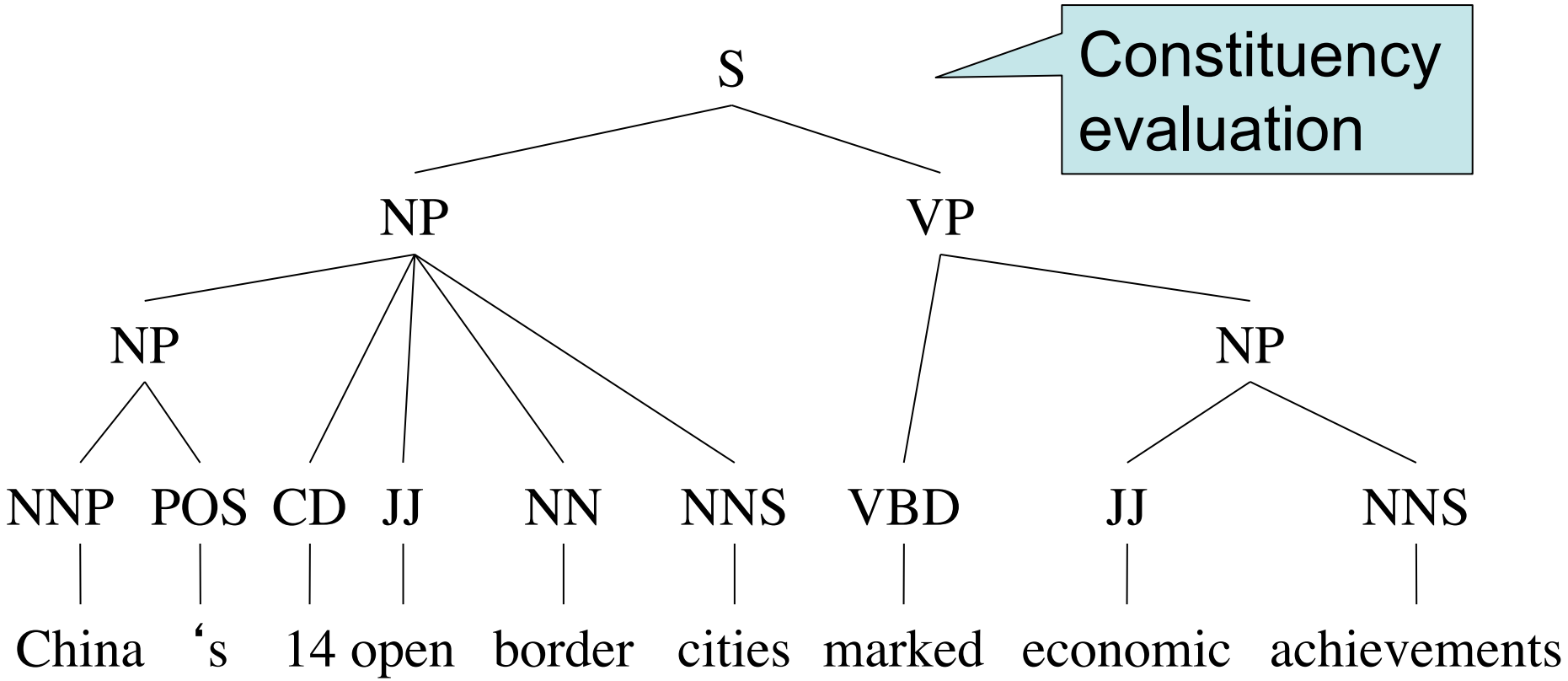
TAG-based discriminative parsing



Dependency evaluation



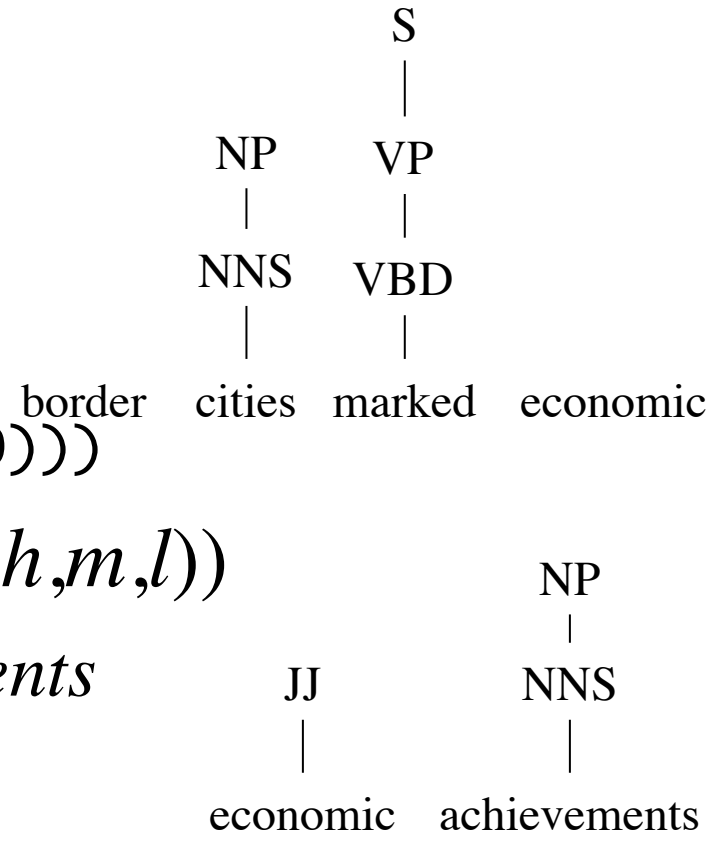
TAG-based discriminative parsing



TAG-based discriminative parsing

- All components have been specified except for **f**(**x**,y)
- Any useful feature functions can be used, aimed at reducing uncertainty in parsing
- (Carreras, Collins and Koo, 2008) used several features from the existing literature
- Plus several TAG based features

Features



- Lexical features **e**(**x**,(*i*,*t*))
  - e.g. marked takes `(S(VP(VBD)))`
- Dependency features **d**(**x**,(*h*,*m*,*l*))
  - e.g. *economic* -> *achievements*
- Grammatical relations
  - e.g. `NP JJ NNS` defines the above dependency
- Sibling dependencies
- Two types of Grandparent dependencies

Efficient Parsing

- The algorithm used for TAG parsing adapts Eisner's dependency parsing algorithm
- Complexity for second order inference (using grandparents) is $O(n^4)$
- Too slow to be tractable in large scale data driven experiments
- (Carreras, Collins and Koo, 2008) use coarse to fine parsing techniques to speed up inference.

TAG for discriminative parsing

- 1st step: Uses dependency parsing (without TAG trees) to eliminate unlikely attachments
- 2nd step: Uses full TAG model for accurate parsing as second step
- State of the art accuracy
  - Phrase structure: 91.1 F1 (versus best 91.7 F1)
  - Dependencies: 93.5% (versus 92.5%)

(arguably with less effort than competing re-ranking methods)

Synchronous TAG

Slides adapted from Chung-hye Han's slides

Synchronous TAG
(Shieber, 1994)

- Just like TAG we have derivation trees
- Except each node in the derivation is not a single elementary tree but rather a pair of trees
- The derivation tree now can be used to build a pair of derived trees
- Synchronous TAG can be used to generate a pair of derived trees or map a source input string to target output string
- Applications: NL semantics (scope ambiguity, etc.) and syntax-based machine translation

Principle of Compositionality

- The Fregean program
- The meaning of a sentence is determined by the meaning of its parts and how they are put together.
- Importance of syntax in the computation of meaning

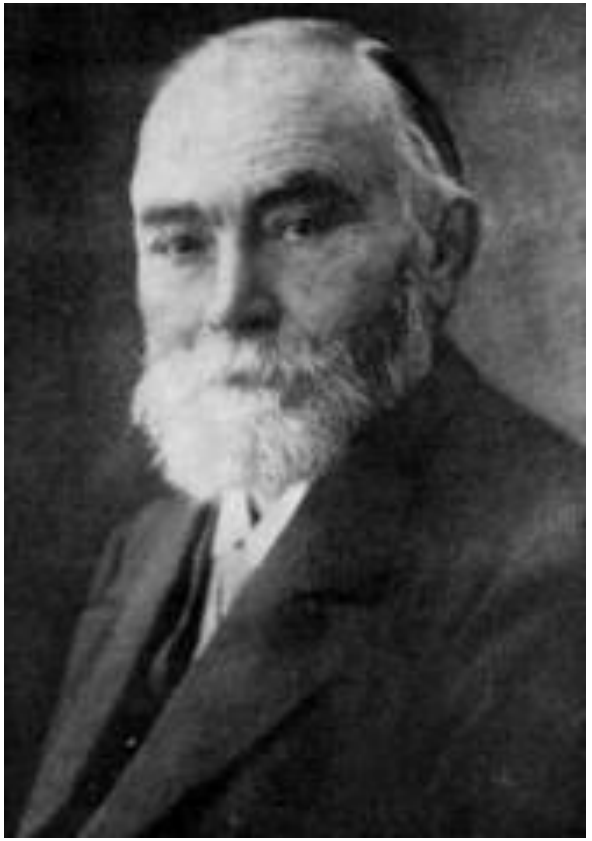
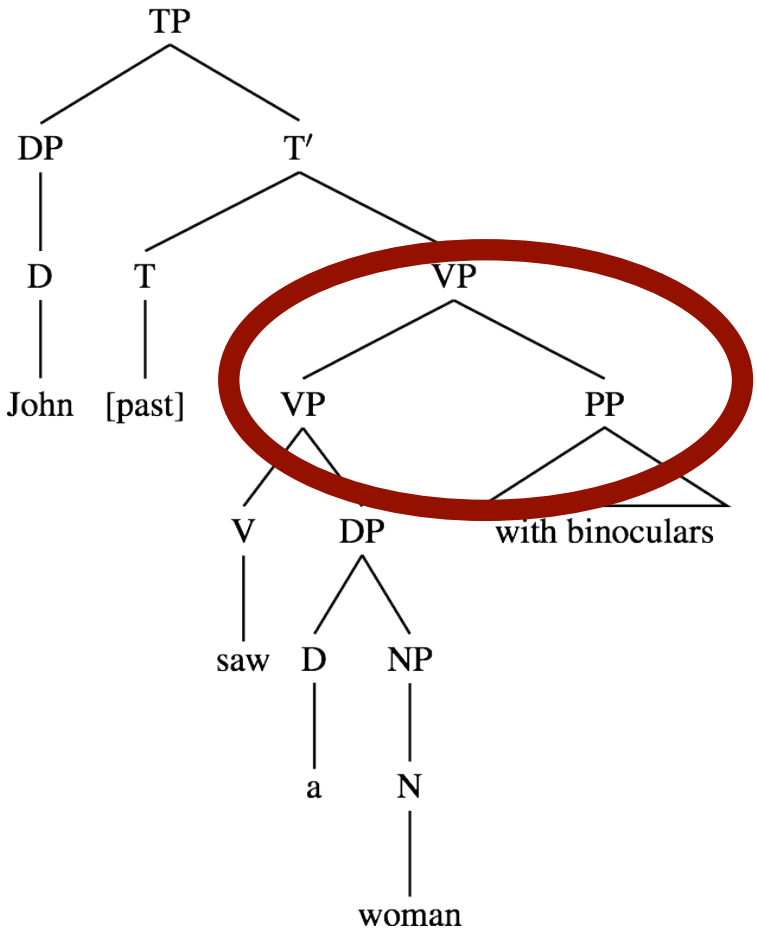
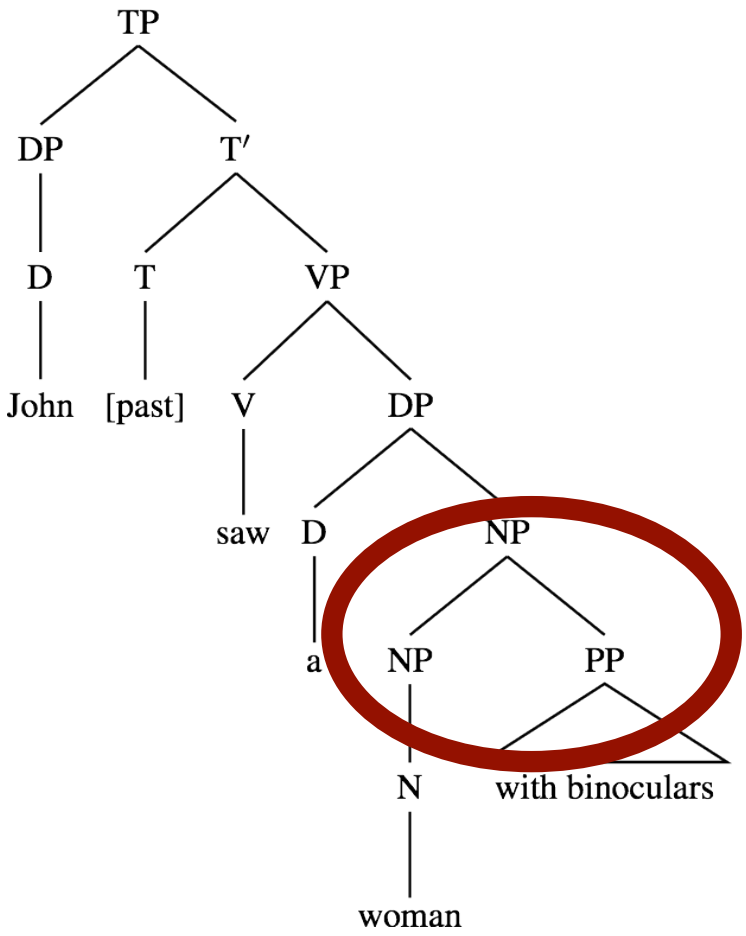


Illustration of Compositionality

John saw a woman with binoculars.

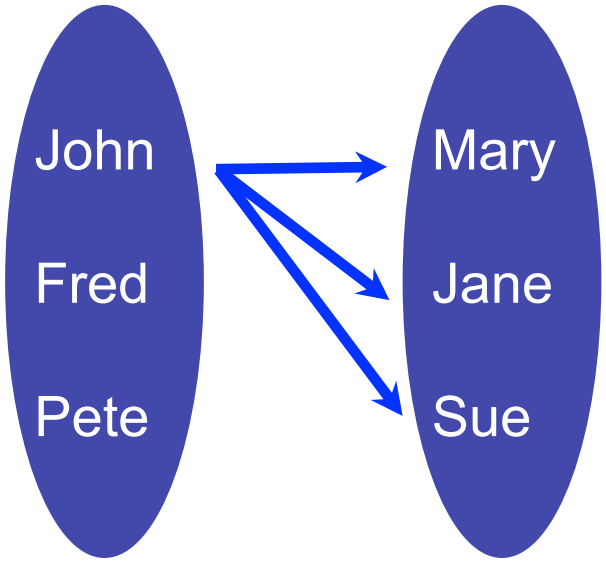




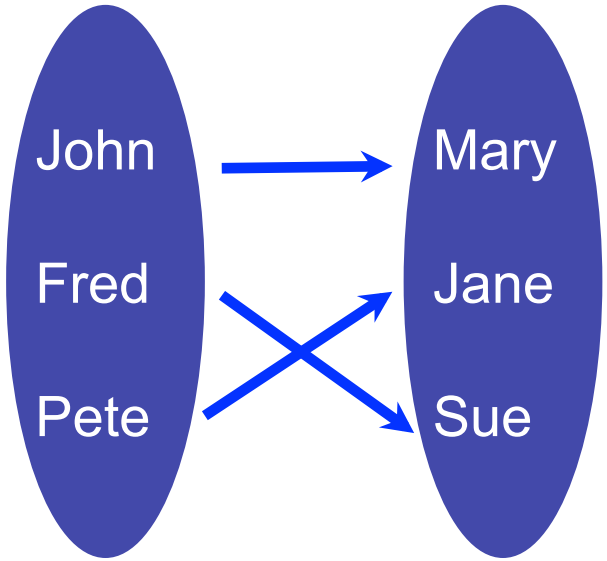
Scope Ambiguity

Some boy is dating every girl.

There is a boy who is dating every girl. (some>every)

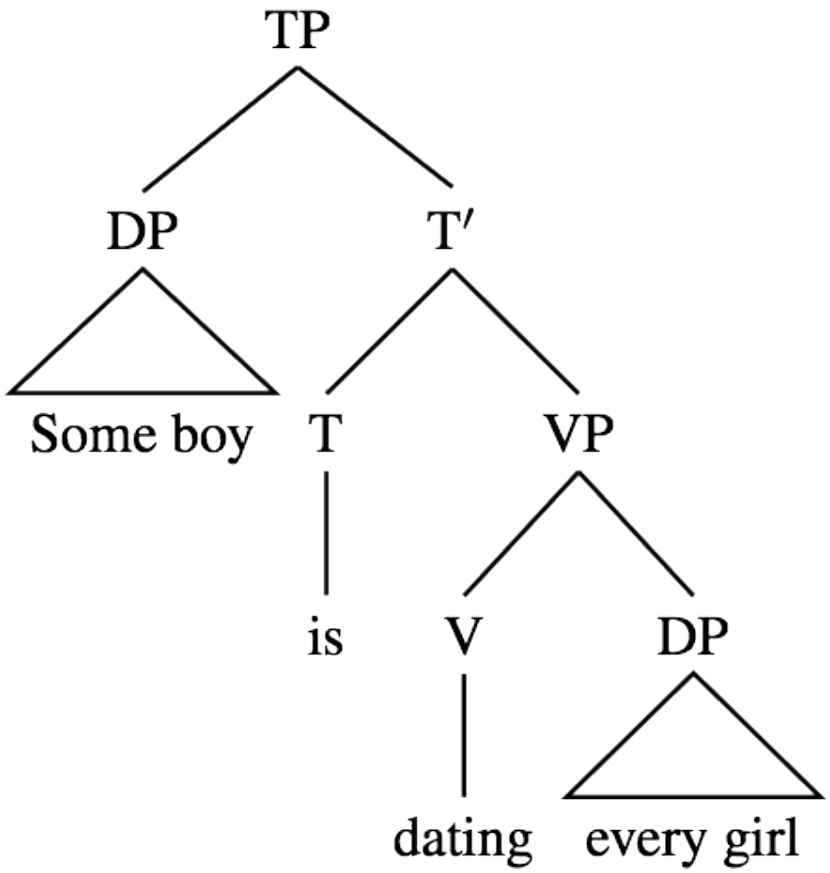


For each girl, there is a boy who is dating her. (every>some)



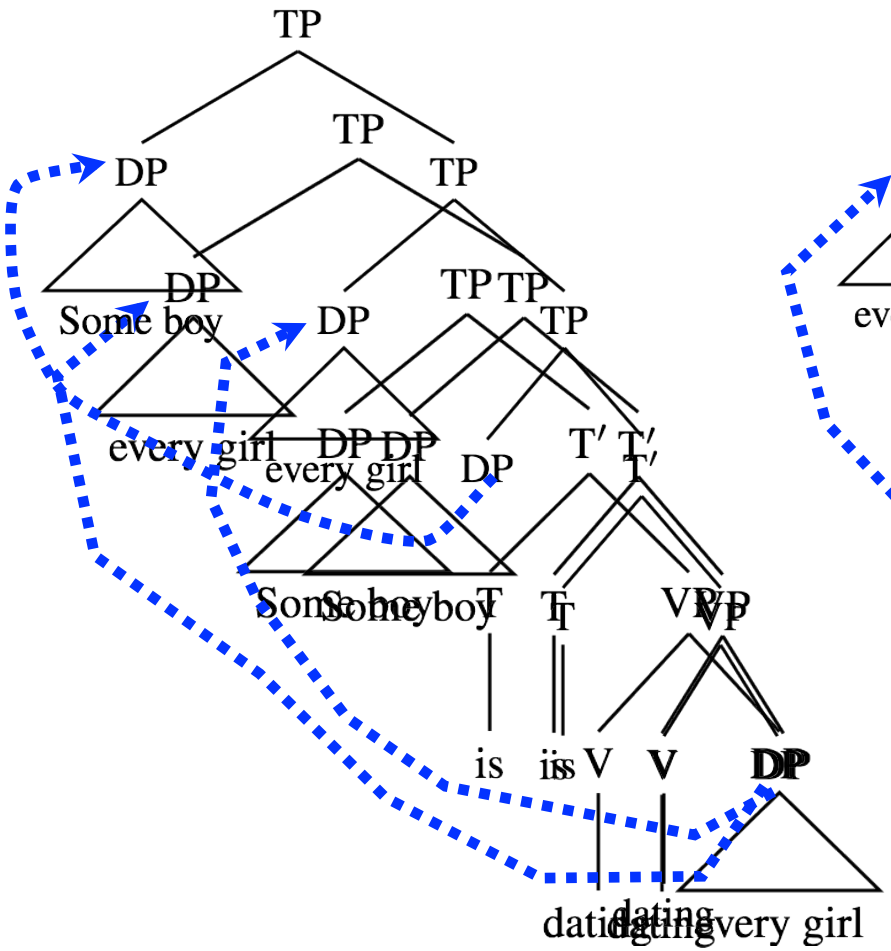
But No Syntactic Ambiguity

Some boy is dating every girl.

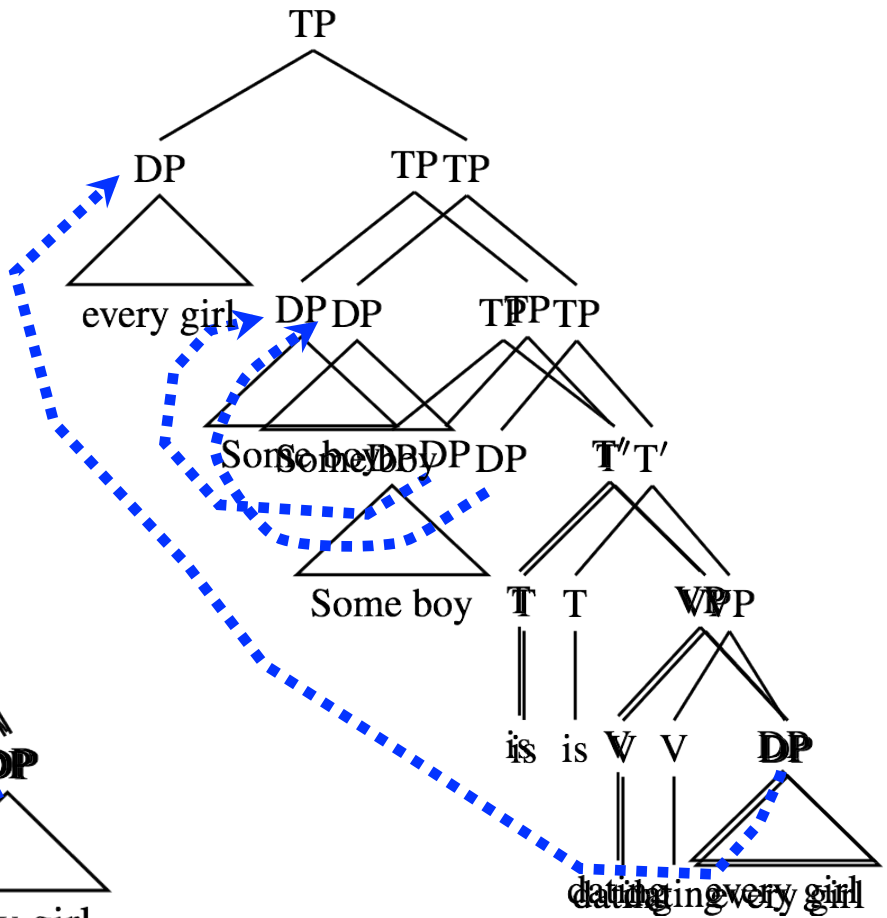


Covert Movement of the Quantified Phrase: Quantifier Raising

- some>every

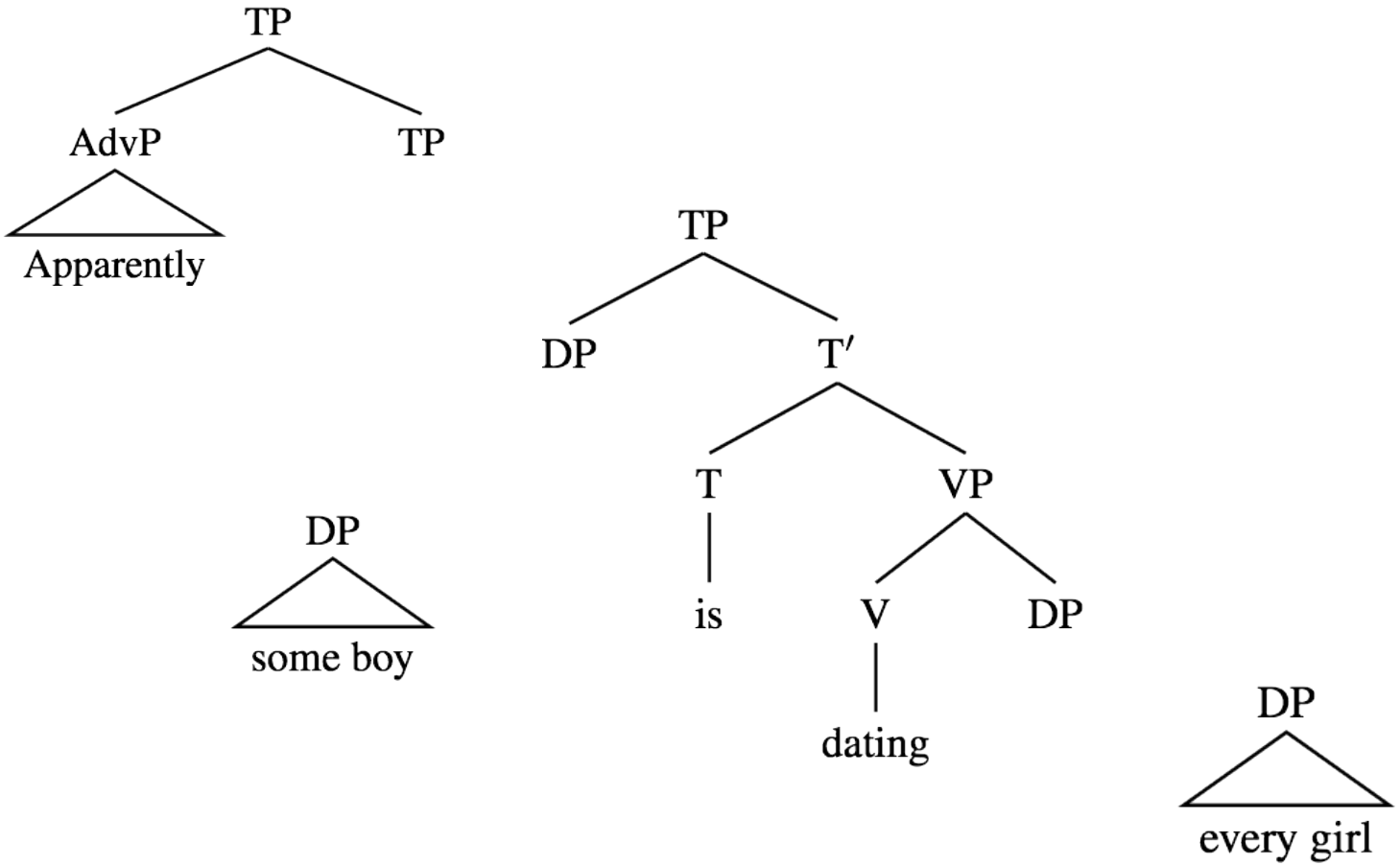


- every>some



Tree Adjoining Grammar: Syntax

Apparently, some boy is dating every girl.

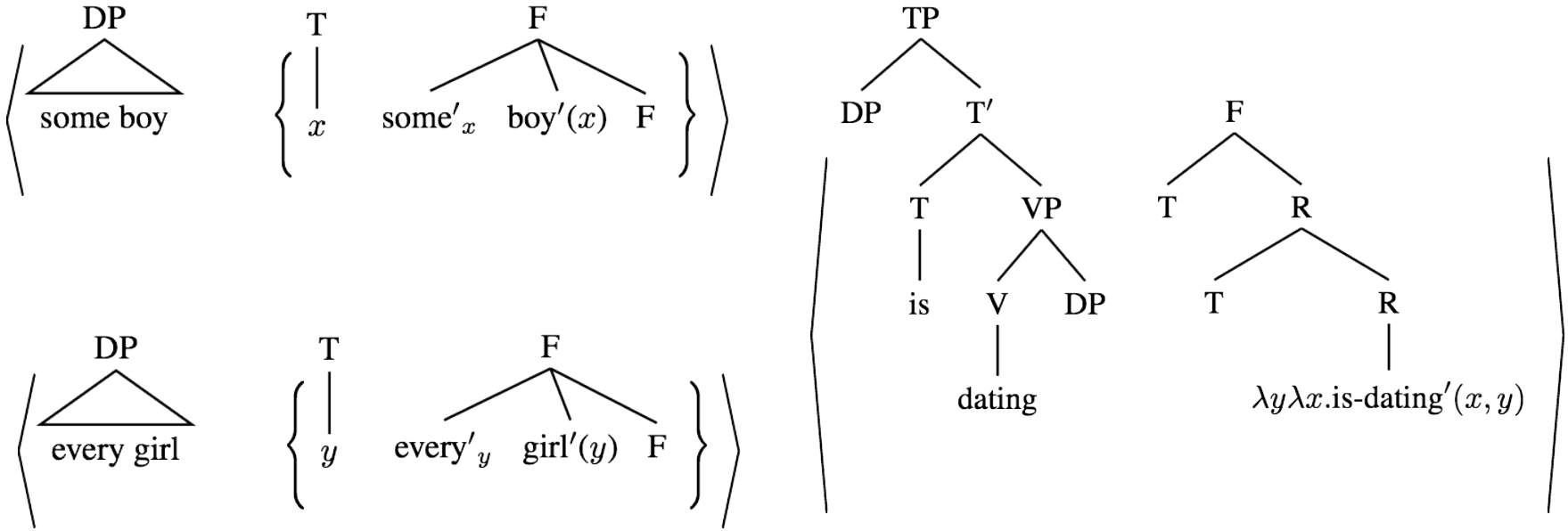


Synchronizing Syntax and Semantics: Synchronous Tree Adjoining Grammar

- A pairing of TAGs, a TAG for syntax and a TAG for semantics (Shieber and Schabes 1990, Shieber 1994)
- Each syntactic elementary tree is paired with one or more semantic trees.
- As syntactic elementary trees compose, corresponding semantic elementary trees compose in parallel.
- Syntactic derivation is isomorphic to semantic derivation.

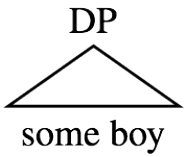
Synchronous Tree Adjoining Grammar (STAG)

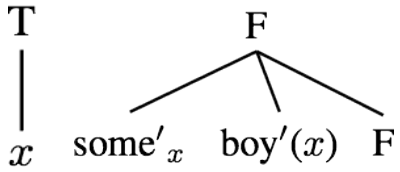
Some boy is dating every girl.

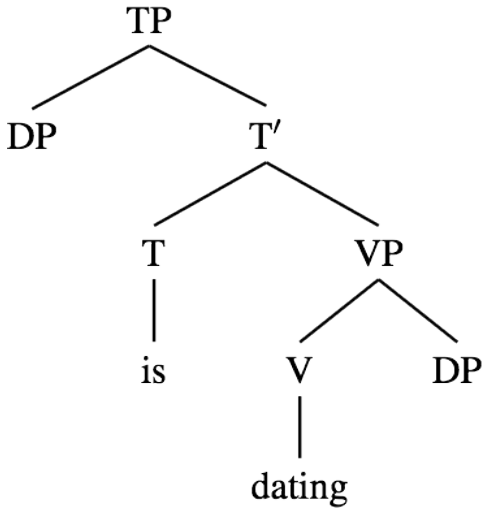


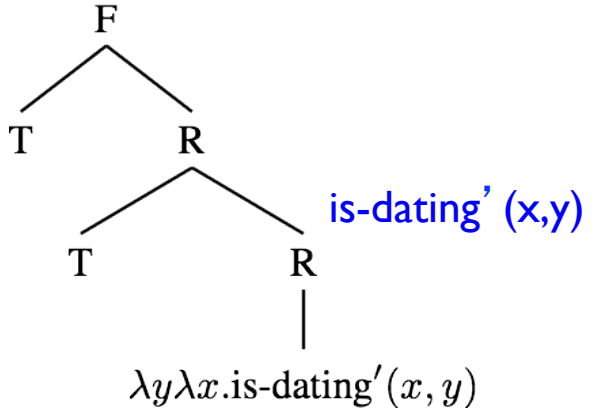
(Shieber and Nesson2006, Han et. al. 2008)

Semantic Composition for *SOME>EVERY*

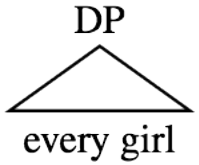


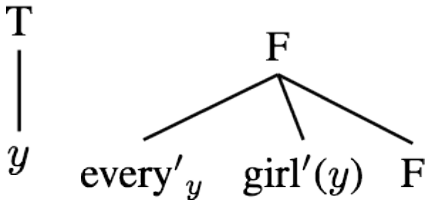




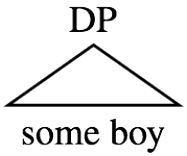


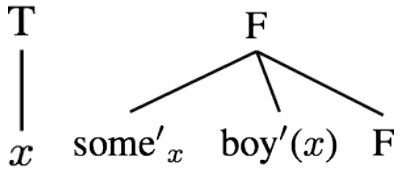
is-dating' (x,y)

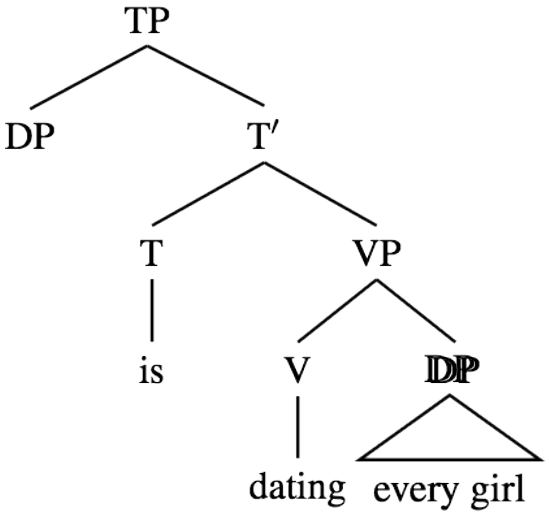


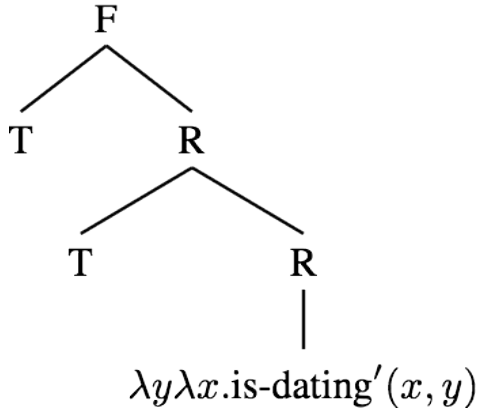


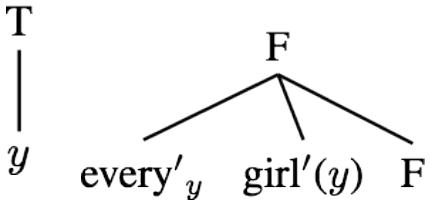
Semantic Composition for *SOME>EVERY*



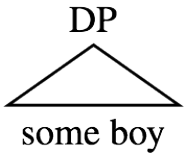


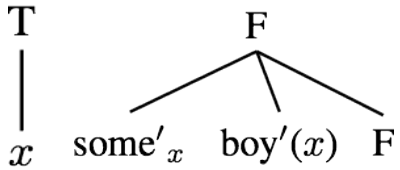


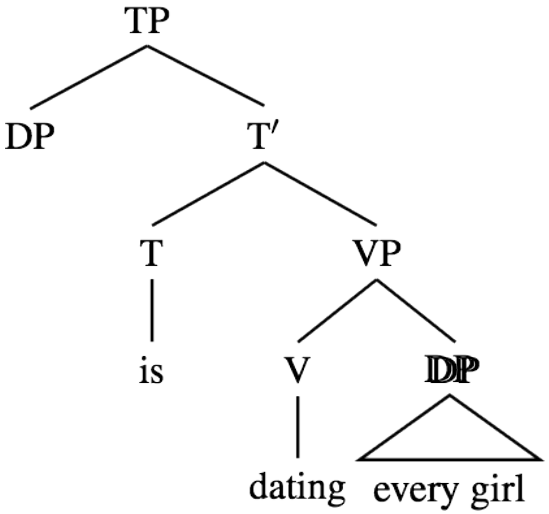


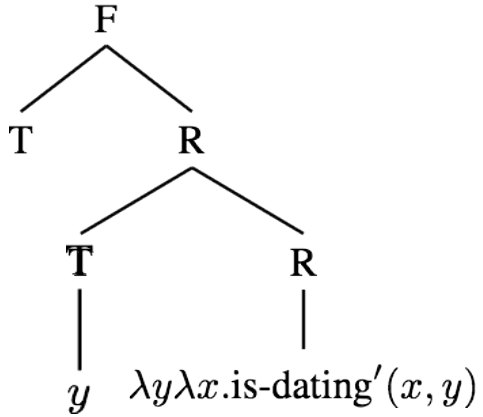


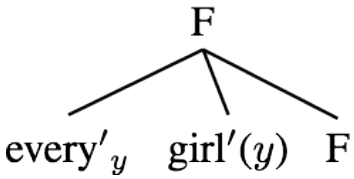
Semantic Composition for *SOME>EVERY*



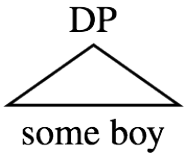


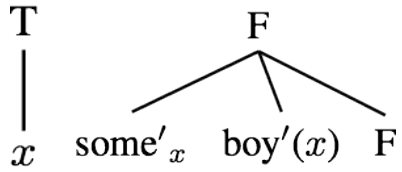


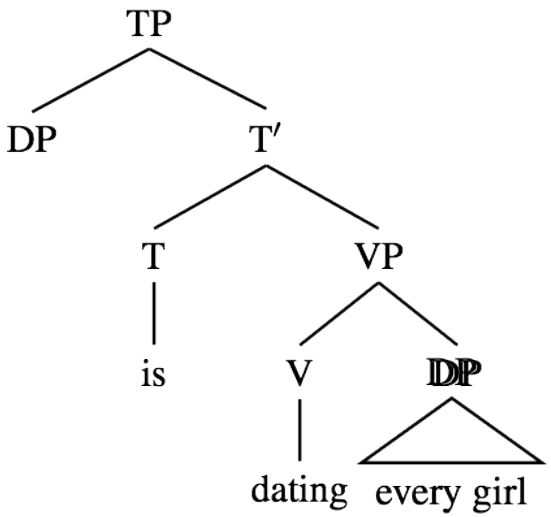


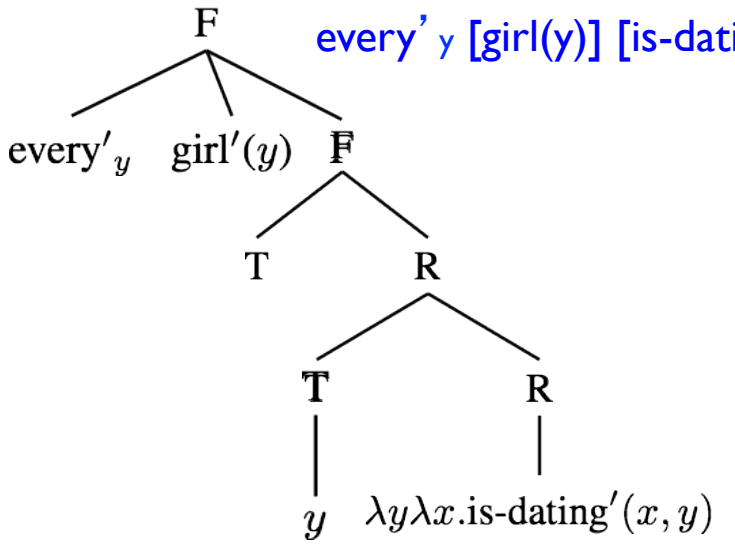


Semantic Composition for *SOME>EVERY*



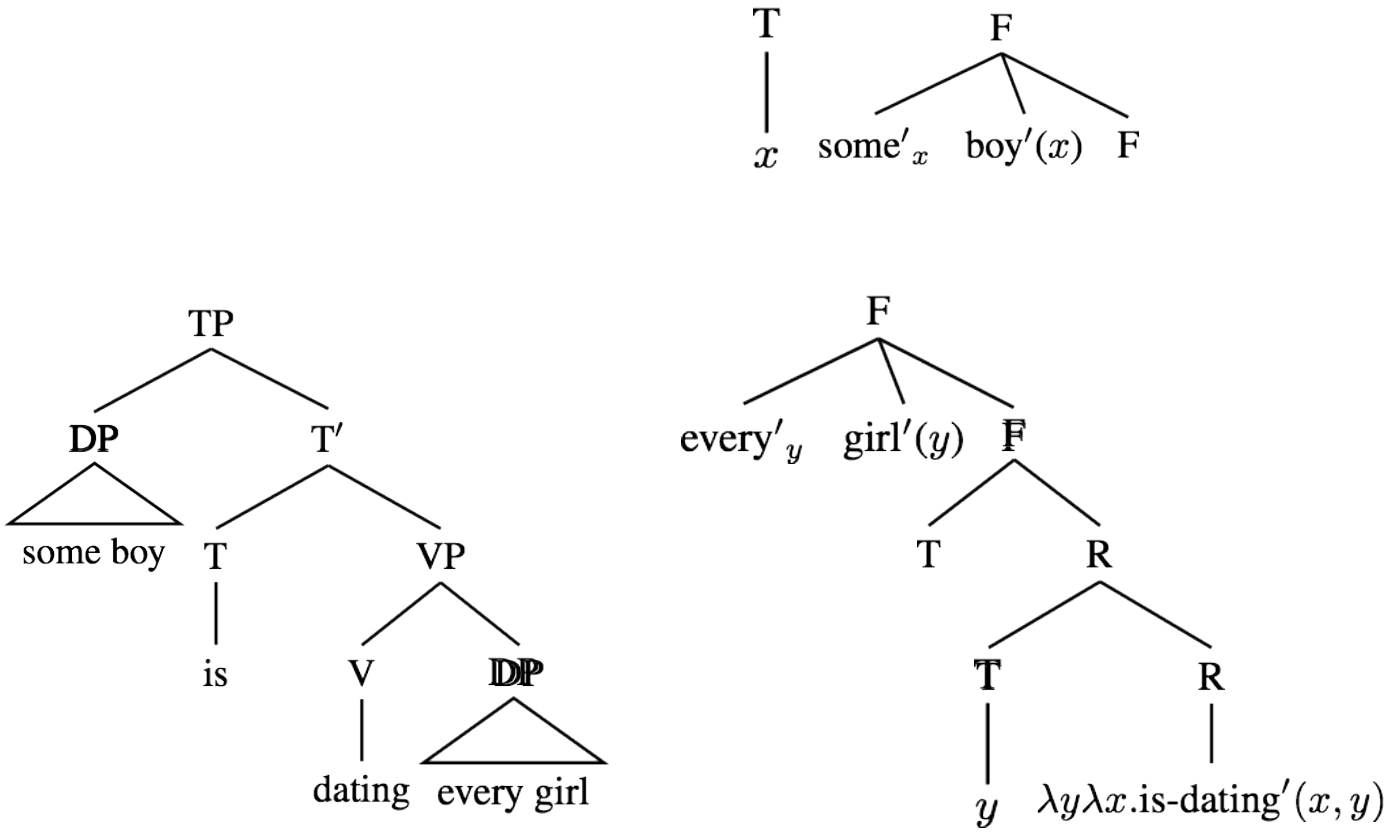




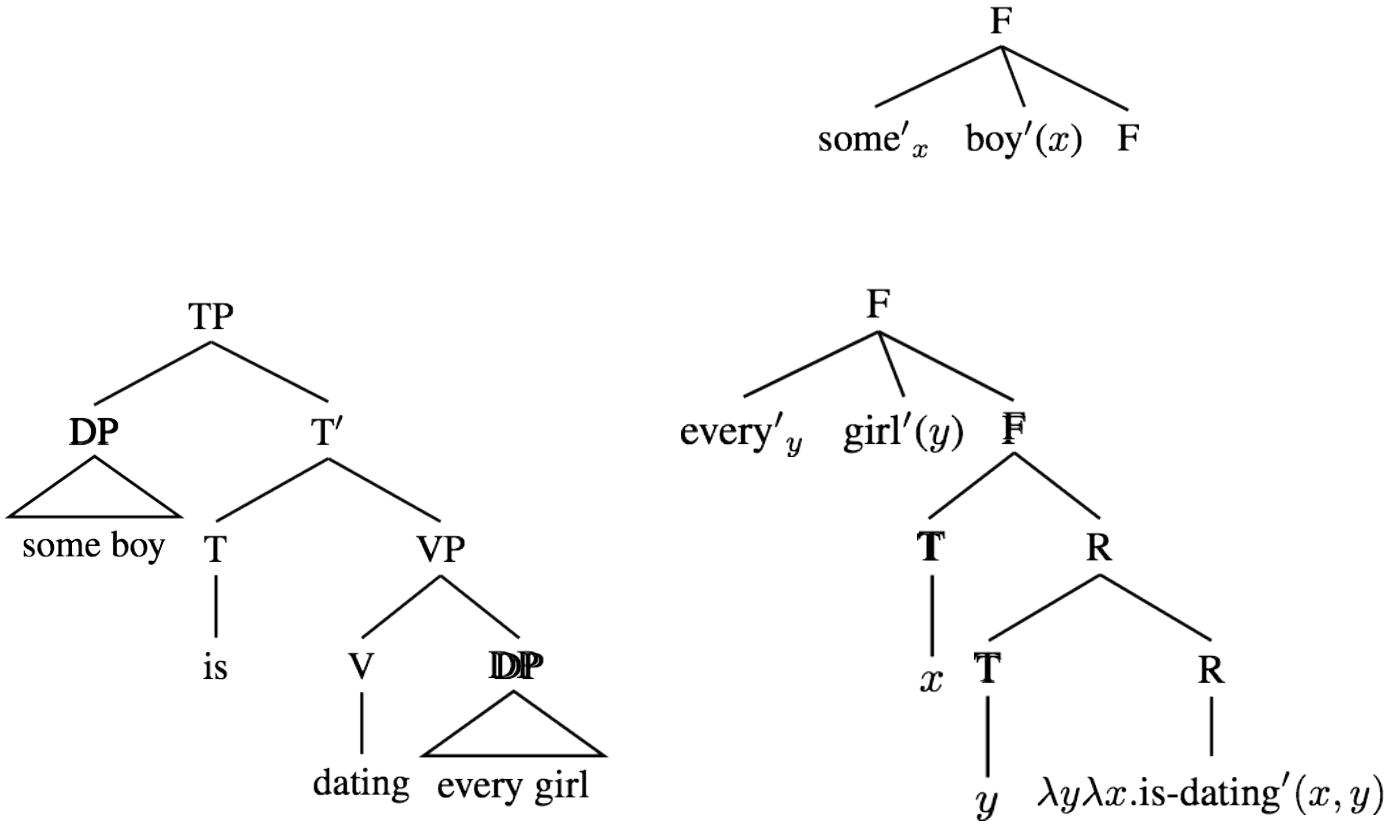


every' y [girl(y)] [is-dating' (x,y)]

Semantic Composition for *SOME>EVERY*

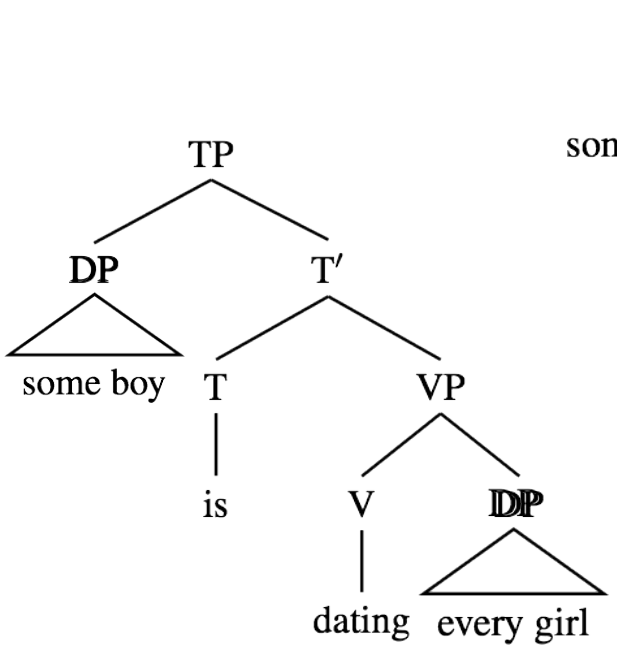


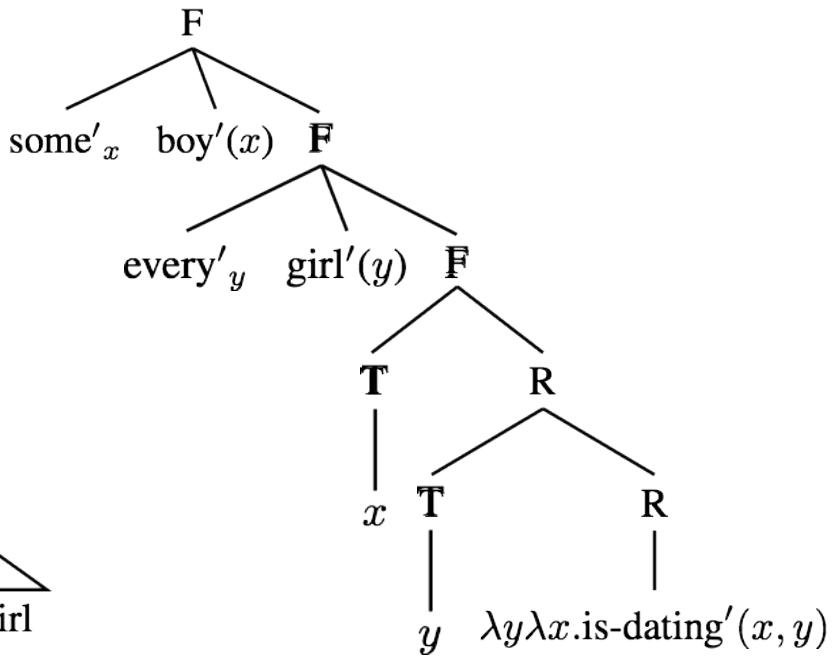
Semantic Composition for *SOME>EVERY*



Semantic Composition for *SOME>EVERY*

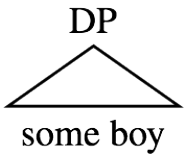
some' x [boy(x)] [every' y [girl(y)] [is-dating' (x,y)]]

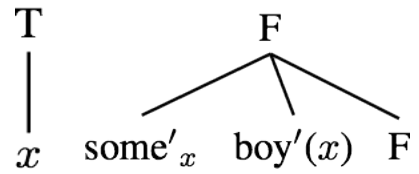


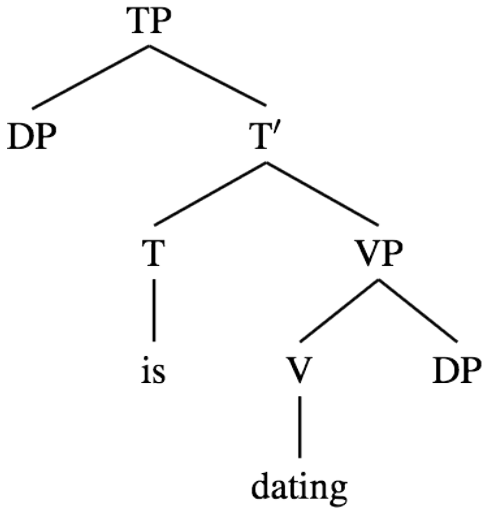


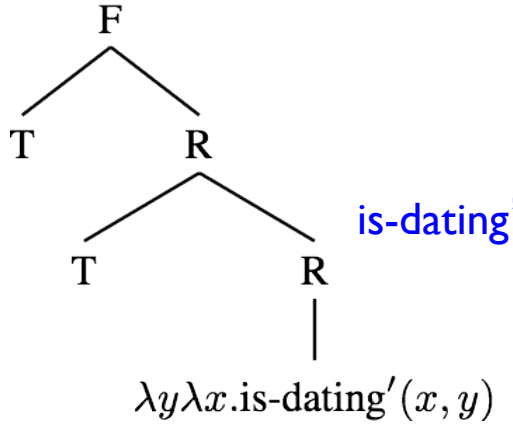
There is a boy who is dating every girl.

Semantic Composition for *EVERY>SOME*

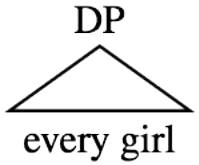


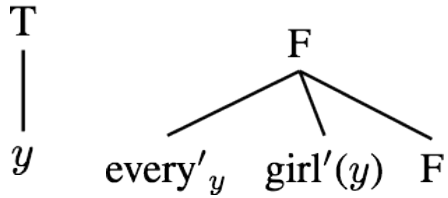




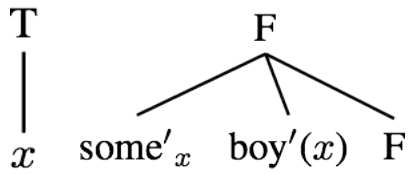


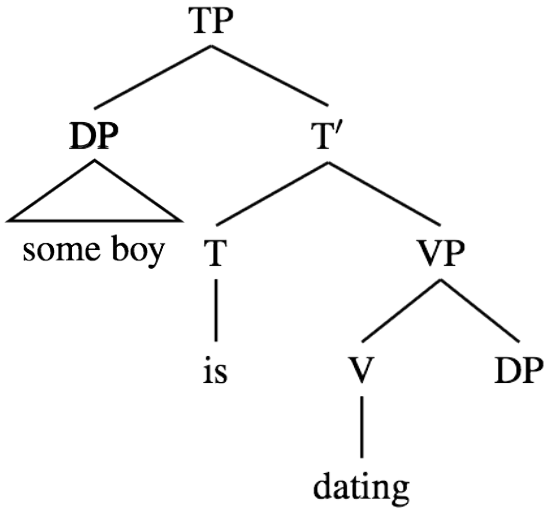
is-dating' (x,y)

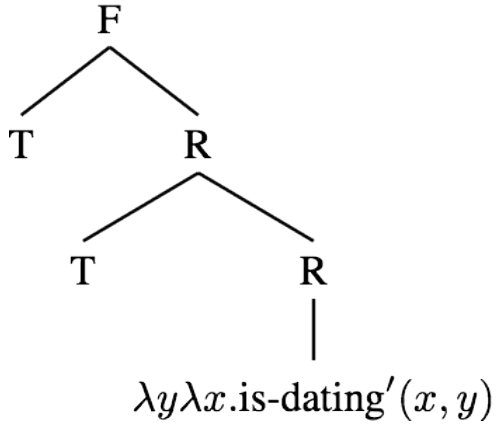


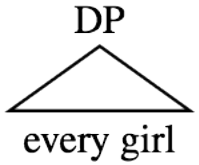


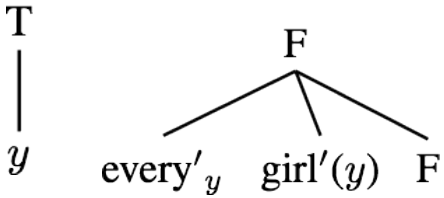
Semantic Composition for *EVERY>SOME*



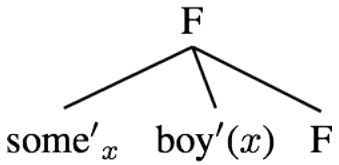


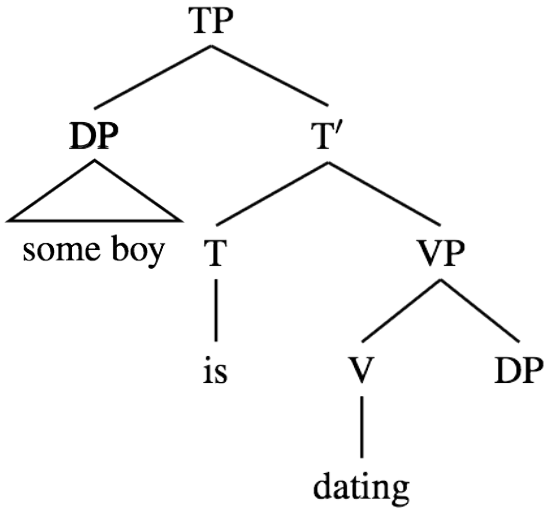


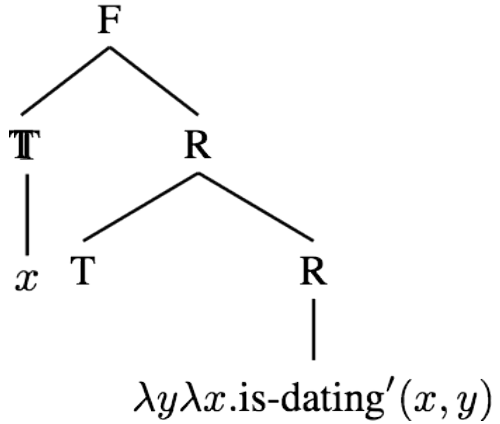


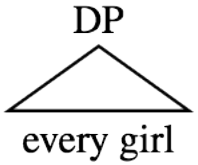


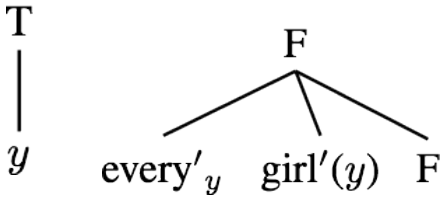
Semantic Composition for *EVERY>SOME*



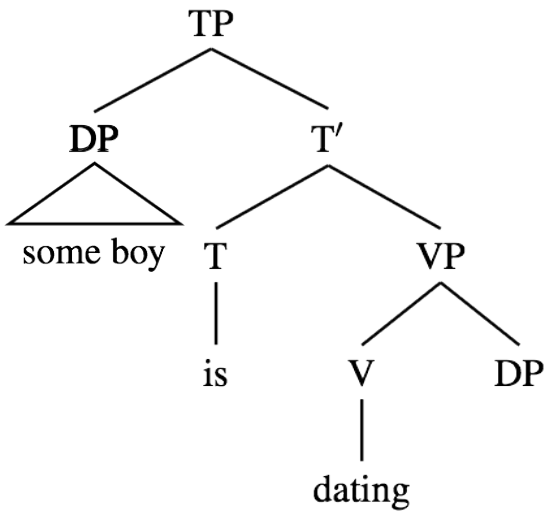




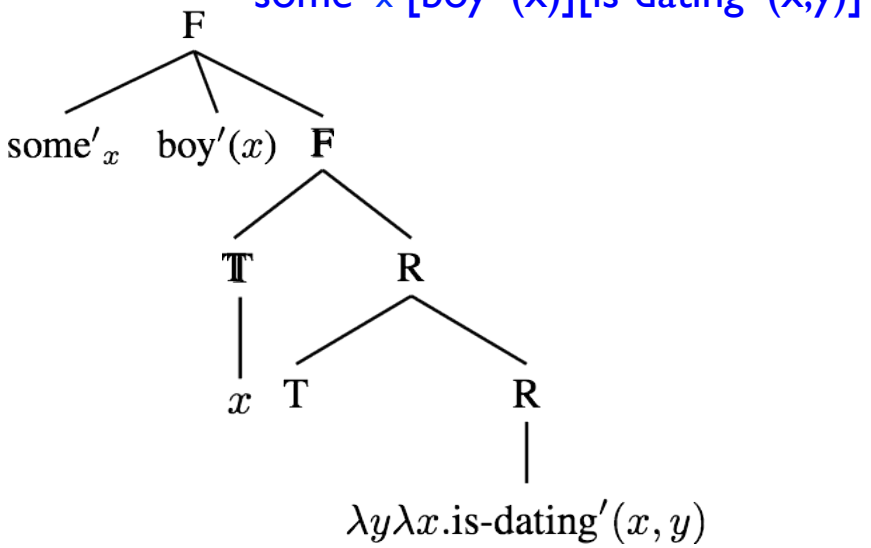


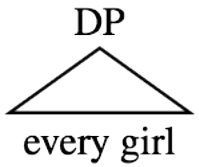


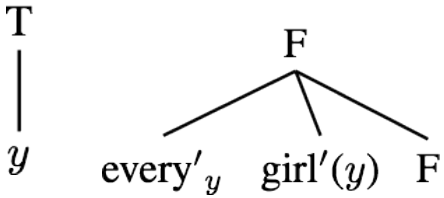
Semantic Composition for *EVERY>SOME*



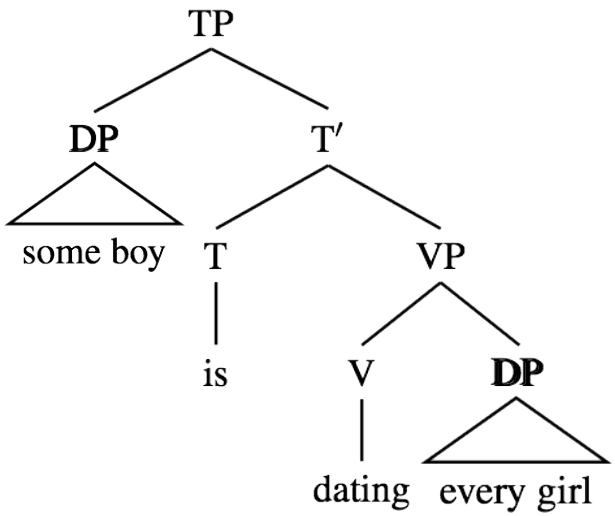
some' x [boy' (x)][is-dating' (x,y)]



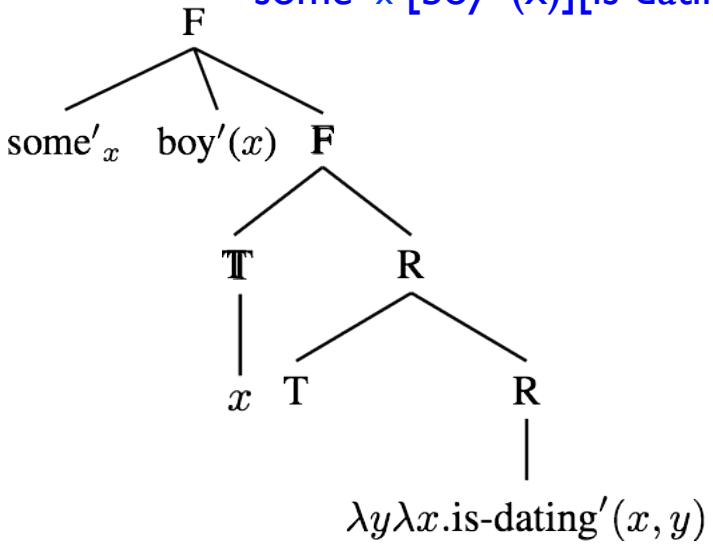


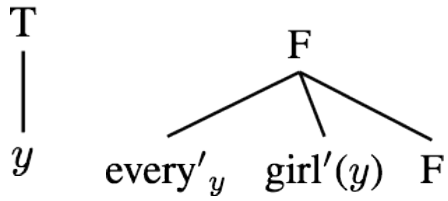


Semantic Composition for *EVERY>SOME*

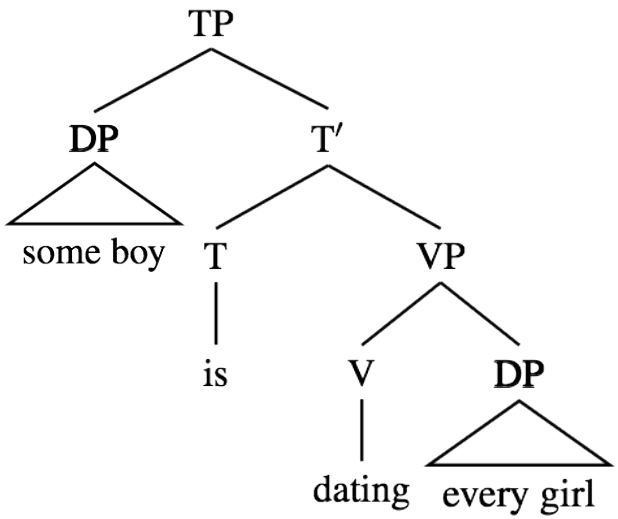


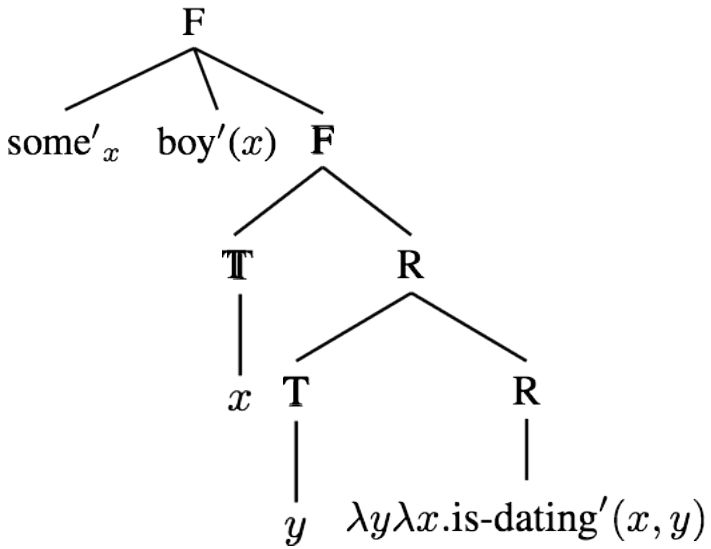
some' x [boy' (x)][is-dating' (x,y)]

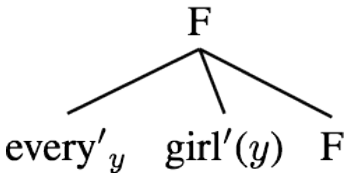




Semantic Composition for *EVERY>SOME*

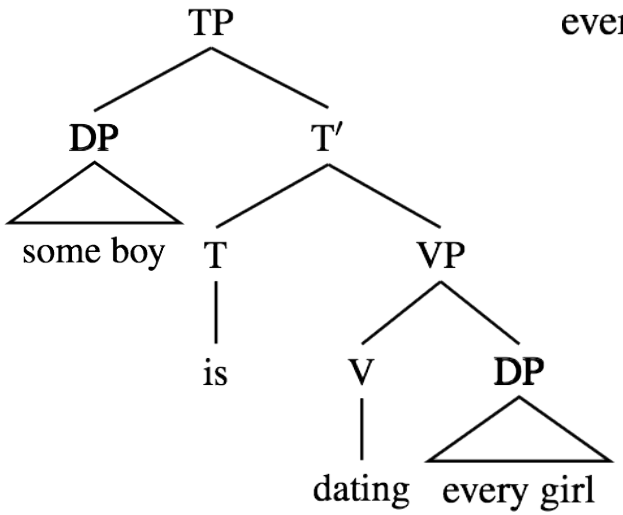


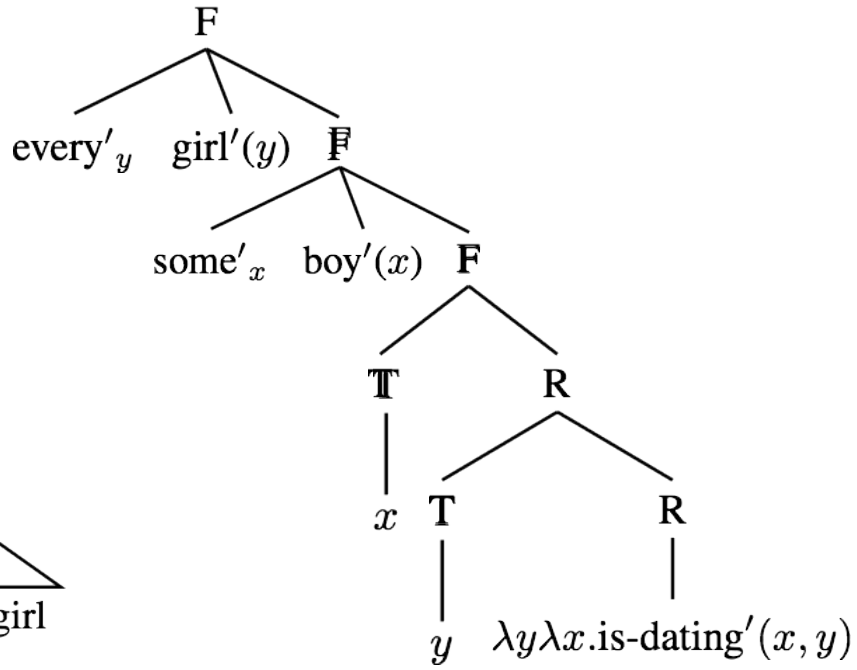




Semantic Composition for *EVERY>SOME*

every' $_y$ [girl(y)] [some' $_x$ [boy' (x)] [is-dating' (x,y)]]





For each girl, there is a boy who is dating her.

STAG as a Theory of Syntax and Semantics Interface

- No intermediate level between syntax and semantics where covert syntactic operations take place
- Syntax directly interfaces with semantics.
- Makes compositional semantics computationally feasible and tractable

Summary

1. Motivating Tree-Adjoining Grammars (TAGs) from a Computational Linguistics perspective
2. Relation to Monadic Simple Context-free Tree Grammars
3. Relation to algebraic notions of crossing dependencies in Dependency Grammar

Summary

4. TAGs as a formalism for natural language syntax
5. Statistical Parsing with weighted TAG
6. Synchronous Tree-adjoining grammar

Bibliography

(Thatcher, 1967): J. W. Thatcher, *Characterizing derivation trees of context-free grammars through a generalization of finite-automata theory*, J. Comput. Sys. Sci., 1 (1967), pp. 317-322

(Rounds, 1970): W. C. Rounds, *Mappings and grammars on trees*, Math. Sys. Theory 4 (1970), pp. 257-287

(Peters & Ritchie, 1969): P. S. Peters and R. W. Ritchie, *Context sensitive immediate constituent analysis -- context-free languages revisited*, Proc. ACM Symp. Theory of Computing, 1969.

(Joshi & Levy, 1977): A. K. Joshi and L. S. Levy, *Constraints on Structural Descriptions: Local Transformations*, SIAM J. of Comput. 6(2), June 1977.

(May & Knight, 2006): J. May and K. Knight, *Tiburon: a weighted automata toolkit*, In Proc. CIAA, Taipei, 2006

(Graehl & Knight, 2004): J. Graehl and K. Knight, *Training tree transducers*, In Proc. of HLT-NAACL, Boston, 2004

Bibliography

(Joshi, 1994): A. K. Joshi, *From Strings to Trees to Strings to Trees ...*, Invited Talk at ACL'94, June 28, 1994.

(Joshi & Schabes, 1997): Joshi, A.K. and Schabes, Y.; Tree-Adjoining Grammars, in *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa (eds.), Vol. 3, Springer, Berlin, New York, 1997, 69 - 124.

(Schabes & Shieber, 1994): Yves Schabes and Stuart M. Shieber. An Alternative Conception of Tree-adjoining Derivation. *Computational Linguistics*, 20(1), March 1994.

(Shieber, 1994): Stuart M. Shieber. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371-385, November 1994. cmp-lg/9404003.

Bibliography

(Brainerd, 1969): W. S. Brainerd, Tree generating regular systems, *Inf. and Contr.* 14 (1969), 217-231

(Fujiyoshi & Kasai, 2000): A. Fujiyoshi and T. Kasai. Spinal-formed context-free tree grammars. *Theory of Comp. Sys.* 33, 59-83. 2000.

(Fujiyoshi, 2005): A. Fujiyoshi. Linearity and nondeletion on monadic context-free tree grammars. *Inf. Proc. Lett.* 93 (2005), 103-107.

(Fischer, 1968): Michael J. Fischer. 1968. Grammars with Macro-Like Productions. Ph.D. dissertation. Harvard University.

(Joshi & Levy, 1977): A. K. Joshi and L. S. Levy, Constraints on Structural Descriptions: Local Transformations, *SIAM J. of Comput.* 6(2), June 1977.

(Joshi, 1994): A. K. Joshi, From Strings to Trees to Strings to Trees ..., Invited Talk at ACL' 94, June 28, 1994.

(Joshi & Schabes, 1997): Joshi, A.K. and Schabes, Y.; Tree-Adjoining Grammars, in *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa (eds.), Vol. 3, Springer, Berlin, New York, 1997, 69 - 124.

(Kanazawa, 2006): M. Kanazawa. *Mathematical Linguistics*, lecture notes. 2006. http://research.nii.ac.jp/~kanazawa/Courses/2006/MathLing/

Bibliography

(Kepser & Mönnich, 2006): S. Kepser and U. Mönnich. Closure properties of linear CFTLs with an application to optimality theory. *Theor. Comp. Sci.* 354, 82-97. 2006.

(Kepser & Rogers, 2007): S. Kepser and J. Rogers. The equivalence of TAG and Monadic Linear CFTGs. In MOL-10. Jul 28-30, 2007.

(Lang, 1994): B. Lang. Recognition can be harder than parsing. *Computational Intelligence,* Vol 10, No 4, Nov 1994.

(May & Knight, 2006): J. May and K. Knight, Tiburon: a weighted automata toolkit, In Proc. CIAA, Taipei, 2006.

(Mönnich, 1997): Uwe Mönnich. Adjunction as substitution: An algebraic formulation of regular, context-free and tree adjoining languages. In Proceedings of the Third Conference on Formal Grammar. 1997.

(Peters & Ritchie, 1969): P. S. Peters and R. W. Ritchie, Context sensitive immediate constituent analysis -- context-free languages revisited, Proc. ACM Symp. Theory of Computing, 1969.

(Rounds, 1970): W. C. Rounds, Mappings and grammars on trees, *Math. Sys. Theory* 4 (1970), pp. 257-287

Bibliography

(Rounds, 1973): William C. Rounds. Complexity of recognition in intermediate-level languages. In 14th Annual IEEE Symposium on Switching and Automata Theory, pages 145-158. 1973.

(Thatcher, 1967): J. W. Thatcher, Characterizing derivation trees of context-free grammars through a generalization of finite-automata theory, *J. Comput. Sys. Sci.*, 1 (1967), pp. 317-322