

Parsing with Tree-Adjoining Grammars

Anoop Sarkar

Simon Fraser University

anoop@cs.sfu.ca

ACL/HCSNet Advanced Program in
Natural Language Processing
Melbourne, Australia July 13-14, 2006

1

High Level Overview

- 1st session
 - Specific motivations for Tree-Adjoining Grammars (TAGs)
 - Lexicalized TAG: definition, examples and extensions
 - Lexicalized TAGs from TreeBanks
 - Synchronous TAG

2

High Level Overview

- 2nd session
 - Statistical parsing: generative models for TAG
 - TAG-based shallow parsing: SuperTagging and Stapling
 - Bootstrapping between CFG and LTAG parsers

3

Preliminaries

4

Formal Languages and NLP

Formal Language Theory	NLP
Language (possibly infinite)	Text Data, Corpus (finite)
Grammar	Grammar (usually inferred from data, produces infinite set)
Automata	Recognition/Generation algorithms

5

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 $w_{-2}, w_{-1}, t_{-2}, t_{-1}, w_0, w_{+1}, w_{+2}, t_{+2}, t_{+1}$, 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

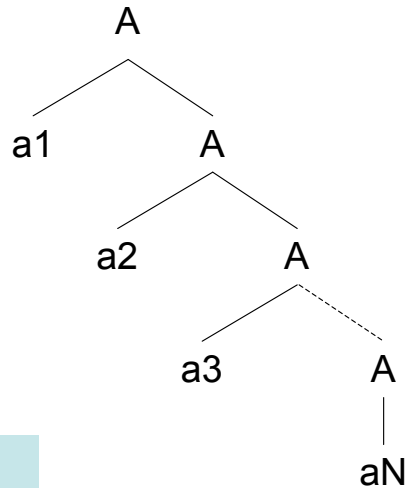
6

Finite State Grammars

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

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

Terminal symbol: a
Non-terminals symbol: A



7

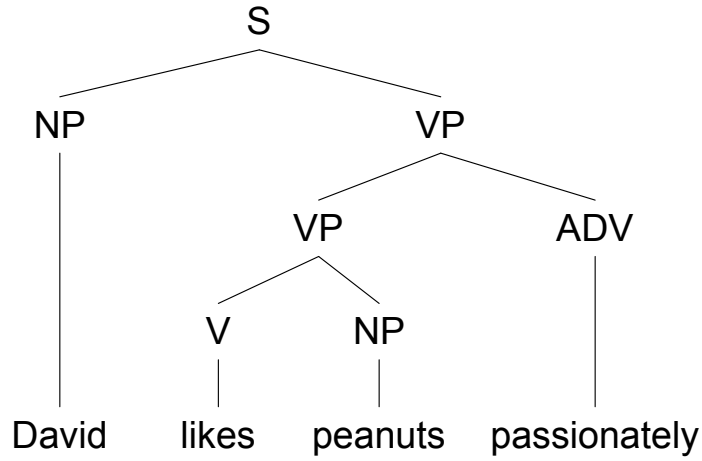
Context-Free Grammars

$S \rightarrow NP VP$
 $VP \rightarrow V NP \mid VP ADV$
 $NP \rightarrow David \mid peanuts$
 $V \rightarrow likes$
 $ADV \rightarrow passionately$

- 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

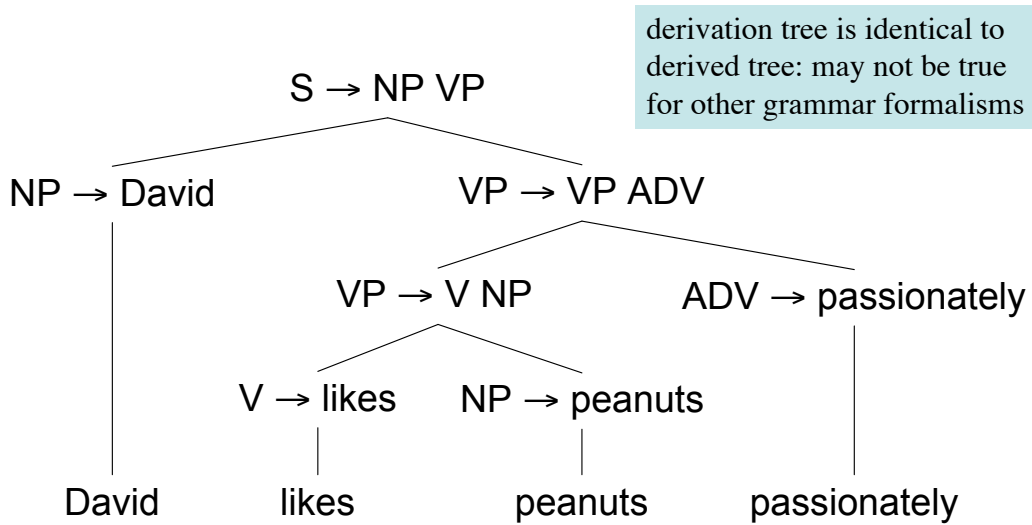
8

CFG: Derived/Parse Tree



9

CFG: Derivation Tree



10

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

11

Motivation #1

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

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

12

Strong vs. Weak Generative Capacity

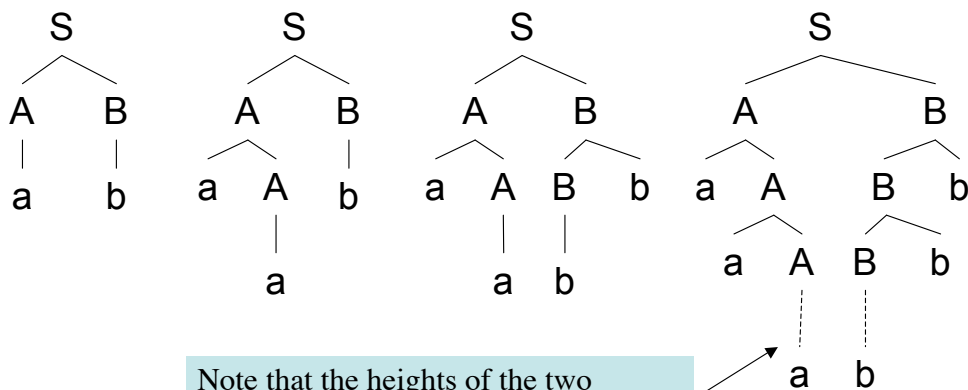
- A property of a formal grammar, e.g. of a regular grammar or a CFG
- **Weak Generative Capacity** of a grammar is the set of strings or the language
- **Strong Generative Capacity** of a grammar is the set of structures (usually the set of trees) produced by the grammar
- The tree for an utterance/sentence is the source of semantics, so the set of trees is more relevant for CL/NLP

13

Tree Sets

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

This grammar generates the tree set shown



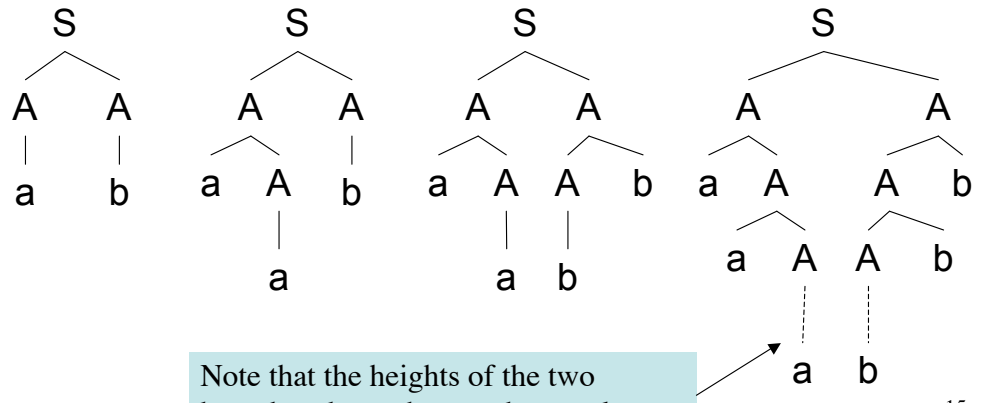
Note that the heights of the two branches do not have to be equal

14

A Tree Set with no CFG

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

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

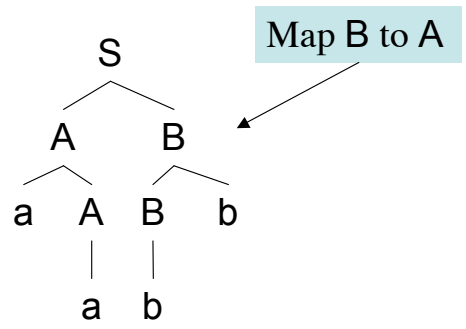


Note that the heights of the two branches do not have to be equal

Generating Tree Sets

- 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) and (Rounds, 1970) provided such a generative device

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

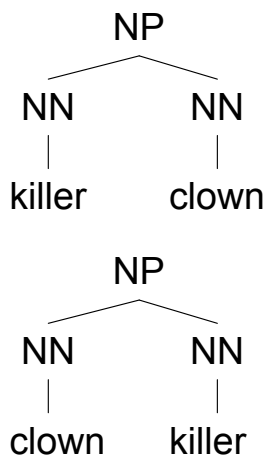


Some definitions

- A **local set** is defined as the set of the set of trees generated by each CFG
- A **recognizable set** is the set of trees generated by each CFGs plus a re-labeling homomorphism
- The recognizable sets strictly contain the local sets
- Recognizable sets provide surprising connections between automata theory, decidability and logic
- They also provide insights into NLP tasks like parsing and syntax-based MT

17

Regular Tree Grammars

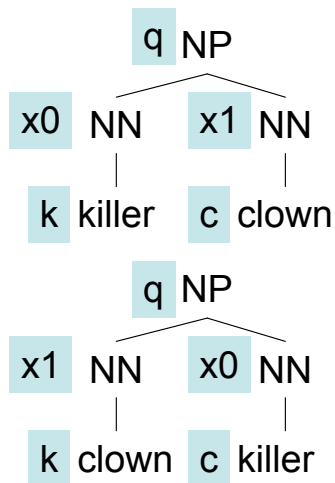


- Consider a simple tree set with two trees for the strings { *killer clown*, *clown killer* }
- No CFG can recognize this simple tree set without also recognizing trees for *clown clown* and *killer killer*
- A Regular Tree Grammar recognizes this tree set (analogy with regular grammars on strings)

Example from (May & Knight, 2006)

18

Regular Tree Grammars



start state: q
 $q \rightarrow NP(x_0 x_1)$
 $q \rightarrow NP(x_1 x_0)$
 $x_0 \rightarrow NN(k)$
 $x_1 \rightarrow NN(c)$
 $k \rightarrow killer$
 $c \rightarrow clown$

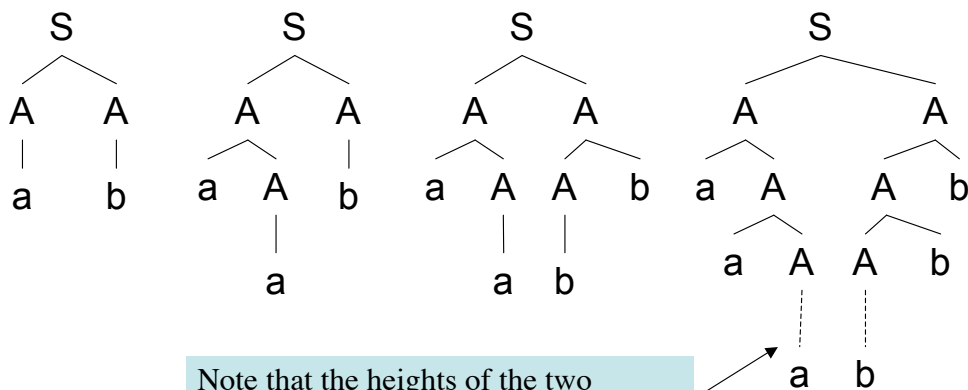
note: can be a tree of any size!

- RTGs = Top-down tree automata
- **Can generate infinite tree sets**
- **Currently used in syntax-based MT**
- for more: (Thatcher, 1967) (Rounds, 1970) (Graehl & Knight, 2004)

Example from (May & Knight, 2006)

Another RTG Example

$q \rightarrow S(x_0 x_1)$
 $x_0 \rightarrow A(a x_0) \mid a$
 $x_1 \rightarrow A(x_1 b) \mid b$



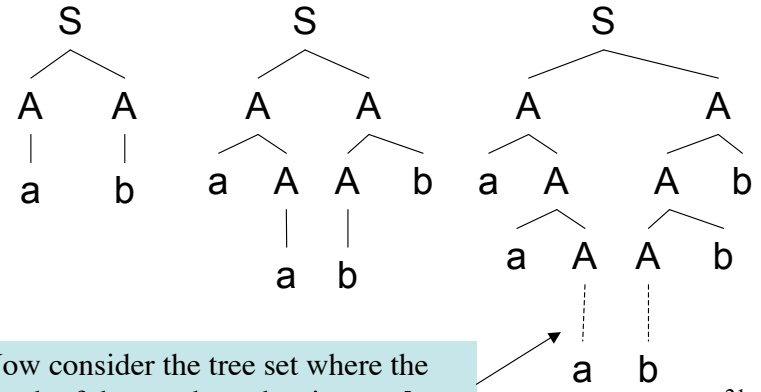
Note that the heights of the two branches do not have to be equal

A Tree Set with no RTG

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

~~$S \rightarrow S(x_0 x_1)$
 $x_0 \rightarrow A(a x_0) \mid a$
 $x_1 \rightarrow A(x_1 b) \mid b$~~

RTG is like a regular grammar, the state cannot count how many times it was reached

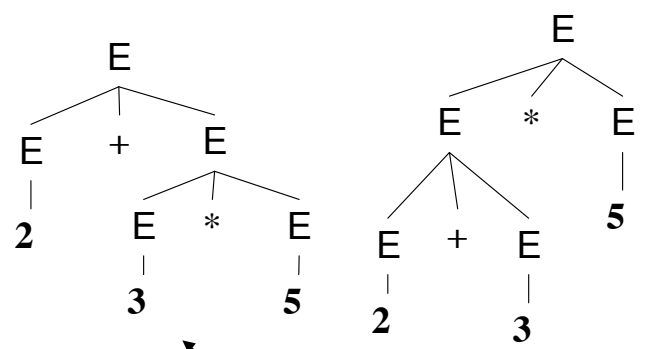


Now consider the tree set where the depth of the two branches is **equal**

Tree Sets: Another Example

A more practical example

- $E \rightarrow E + E$
- $E \rightarrow E * E$
- $E \rightarrow (E)$
- $E \rightarrow N$



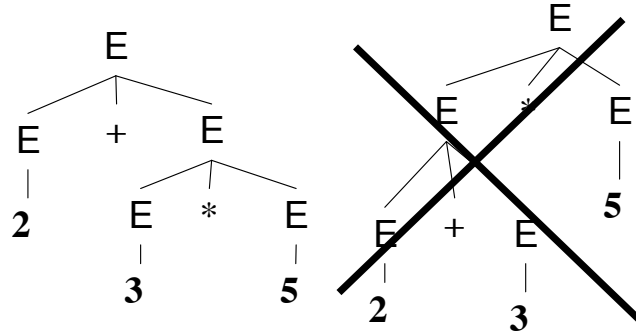
$2+3*5$ is ambiguous either **17** or **25**

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

Tree Sets: Context-sensitivity

Eliminating ambiguity

$E \rightarrow E + E$
 $\neg(+_)\wedge\neg(*_)\wedge\neg(_*)$
 $E \rightarrow E * E$
 $\neg(*_)$
 $E \rightarrow (E)$
 $E \rightarrow N$



similar to context-sensitive grammars!

23

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 $\{ a^{2^i} : i > 0 \}$
- This kind of computational power is unlikely to be useful 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?

24

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
- So a CSG can be used to accept a tree set
- Can we precisely describe this set of tree languages?

25

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)

26

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) result shows that **local transformations** used in early linguistic formalisms can be all written down as weakly equivalent CFGs
- Important caveat: we assume some source generating trees which are then validated

27

Locality of CSG predicates

- An analysis of the kinds of CSG grammars used to define linguistic analyses in practice showed an interesting fact
- All the CSG predicates were very local
- They did not include in the context various parts of the tree that were arbitrarily far apart
- Long distance dependencies were expressed by chaining together many local CSG predicates
- This insight can be used to generate trees from an input string and validate them using CSG predicates

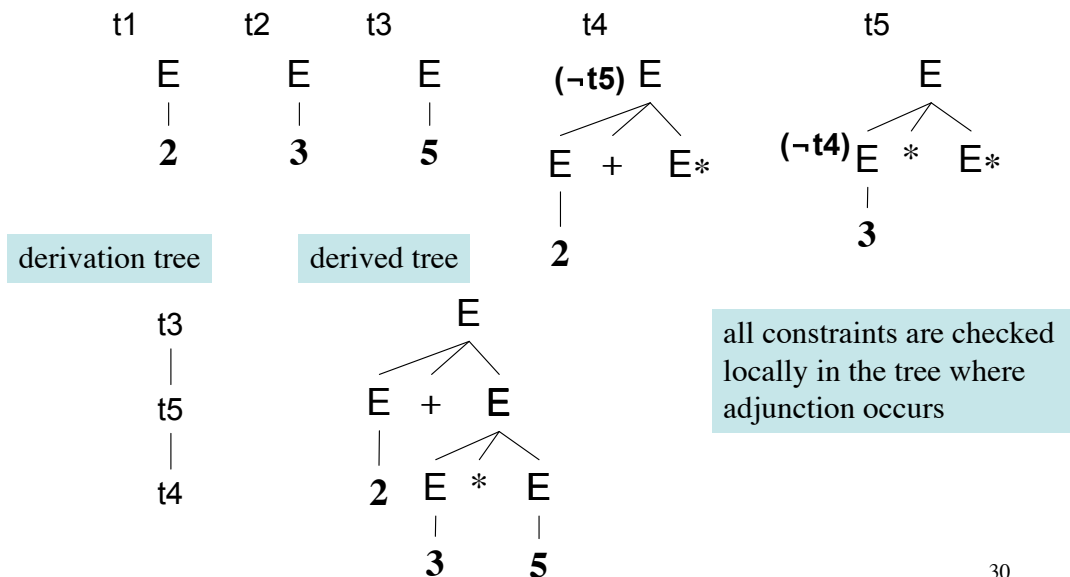
28

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

29

Tree-Adjoining Grammars



30

Motivation #2

Lexicalization of Context-Free Grammars

33

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 $NP \rightarrow NP$ has this effect
- Lexicalization of a grammar means that each rule or elementary object in the grammar is associated with some terminal symbol

34

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)

35

Lexicalization of CFGs

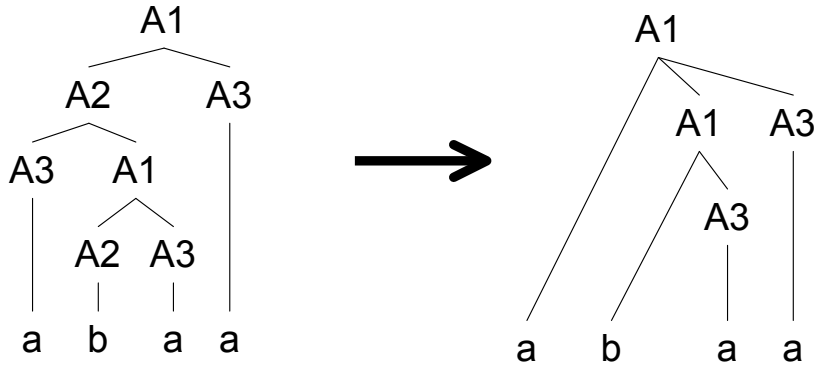
- A **normal form** is a grammar transformation that does not change the language of the grammar
- Can we transform every CFG to a normal form where there is guaranteed to be a terminal symbol on the right hand side of each rule
- Answer: yes - using Greibach Normal Form (GNF)
- GNF: every CFG can be 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

36

$T(G) \neq T(GNF(G))$

$A1 \rightarrow A2 A3$
 $A2 \rightarrow A3 A1 \mid b$
 $A3 \rightarrow A1 A2 \mid a$

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

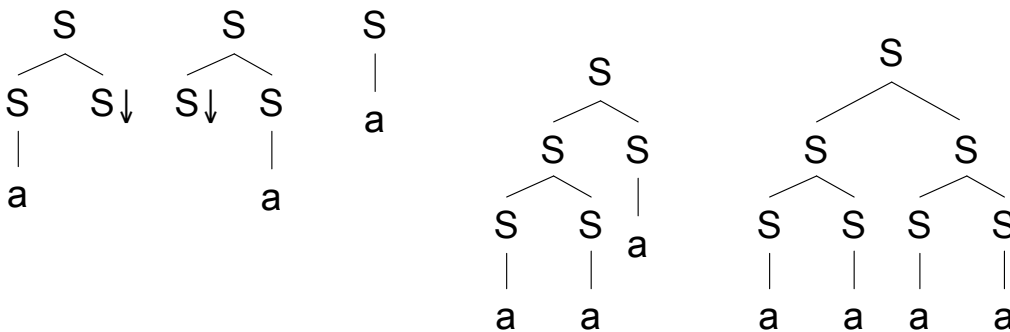


37

Tree Substitution Grammar

$S \rightarrow S S$
 $S \rightarrow a$

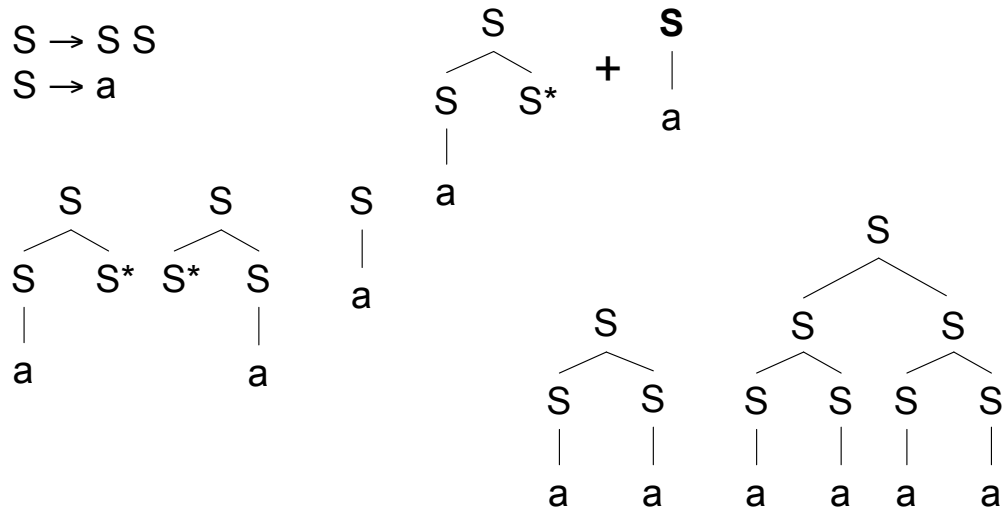
Consider a simple expansion of each context-free rule into a tree fragment where each fragment is lexicalized



this tree cannot be derived

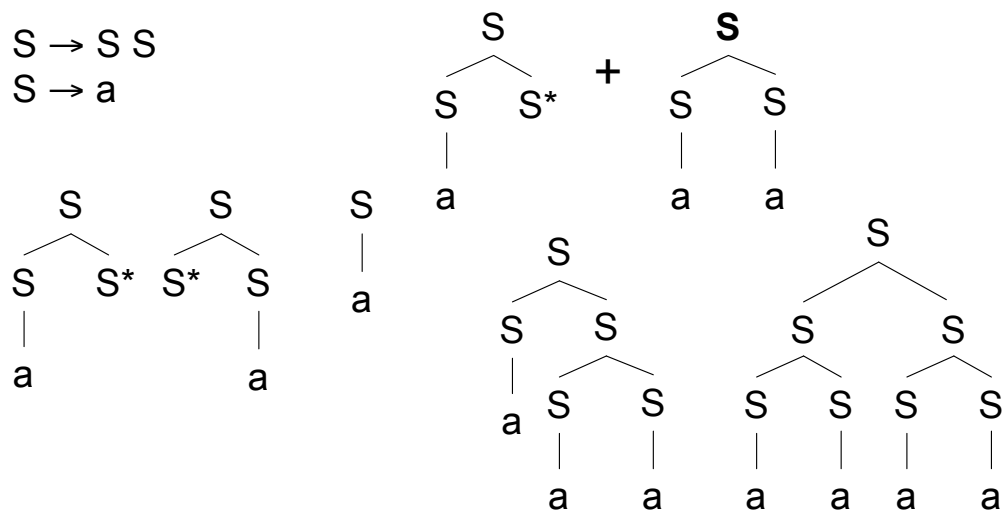
38

Tree Adjoining Grammar



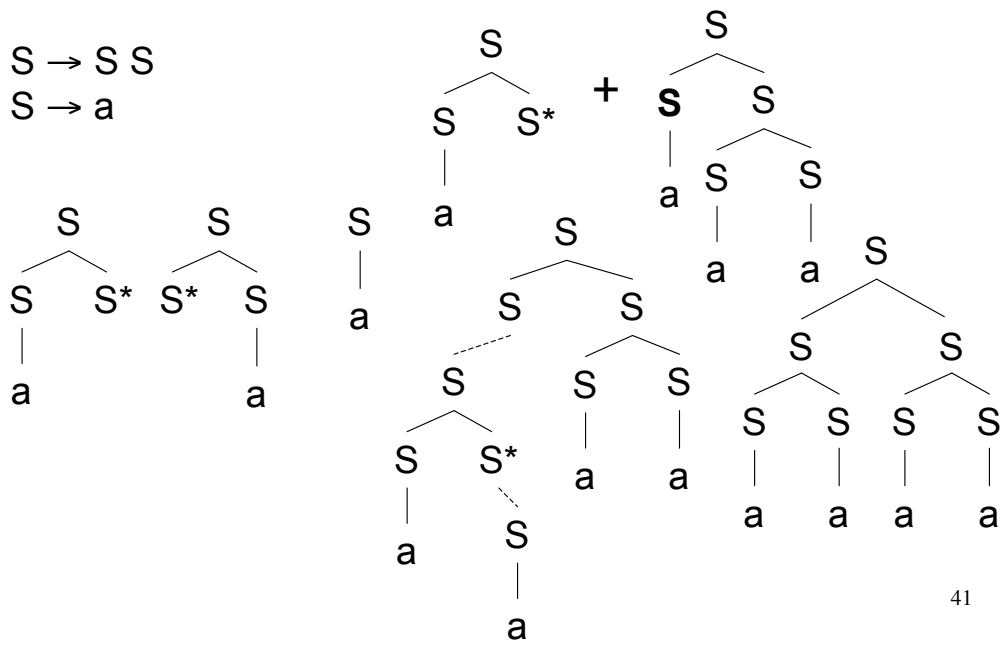
39

Tree Adjoining Grammar

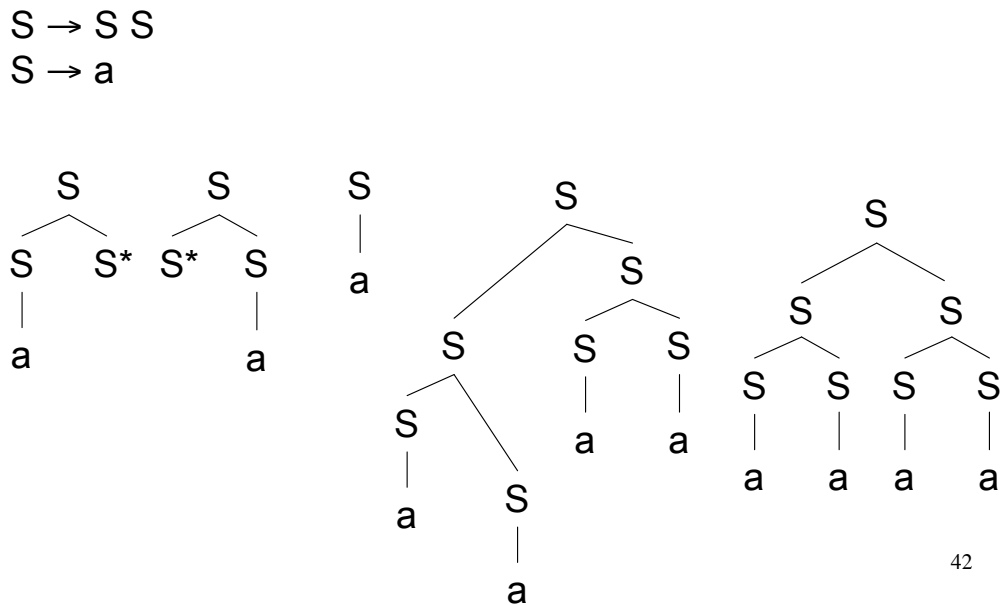


40

Tree Adjoining Grammar



Tree Adjoining Grammar

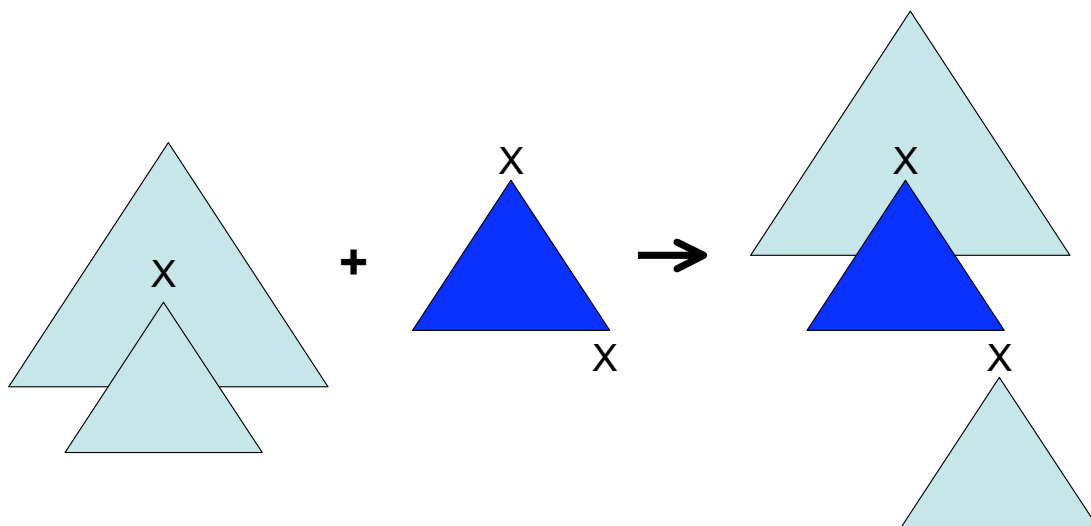


Lexicalization Through TAG

- 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) explain how every CFG can be strongly lexicalized by TAG
- (Joshi & Schabes, 1997) show that Tree-Adjoining Languages are closed under lexicalization: every TAL has a lexicalized TAG grammar

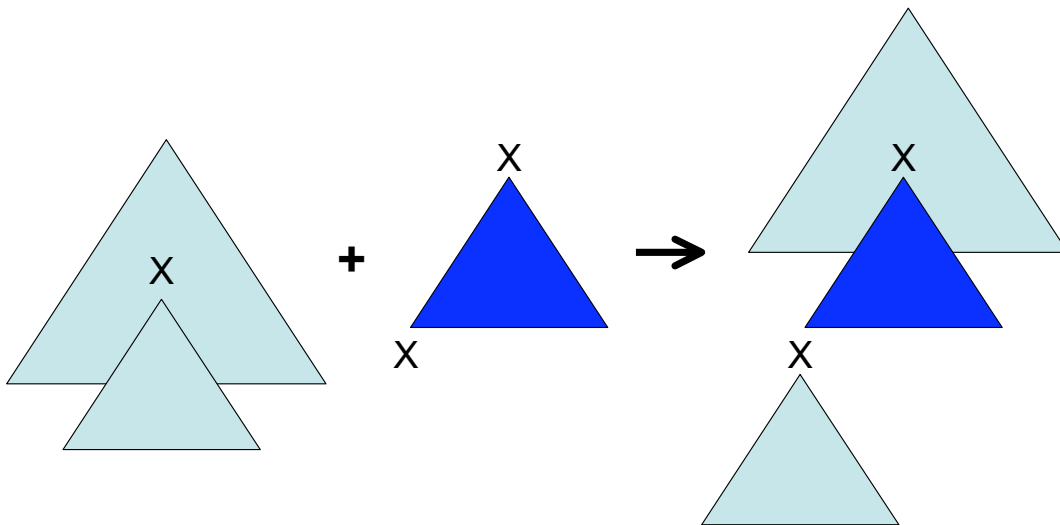
43

Tree-Insertion with adjoining



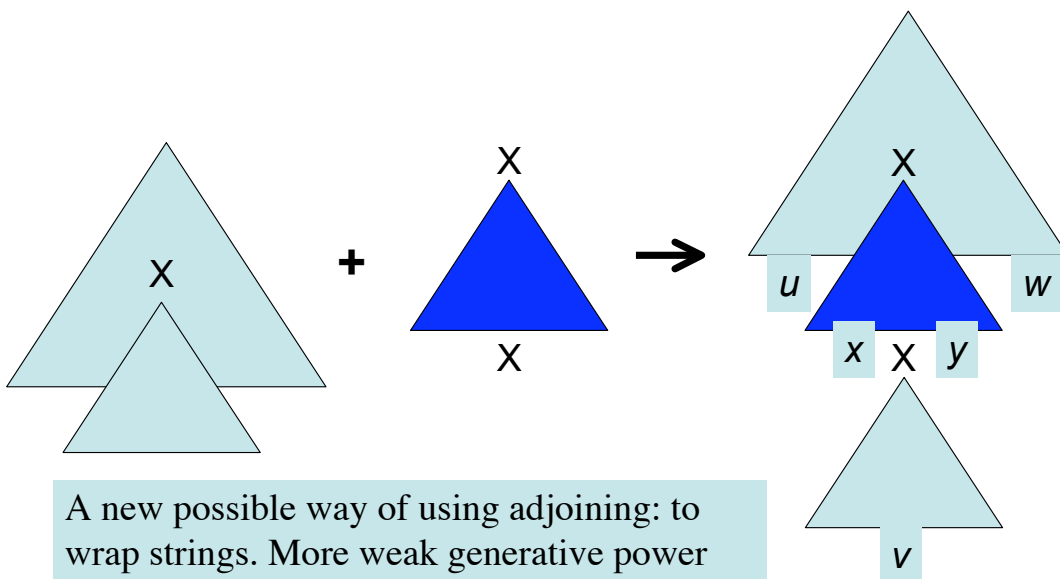
44

Tree-Insertion with adjoining



45

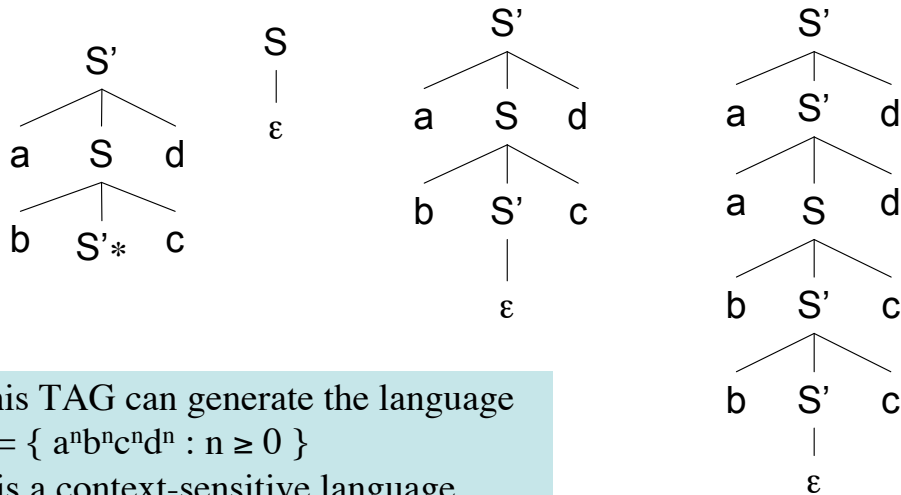
Wrapping with adjoining



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

46

Wrapping with Adjoining



47

Motivation #3

Is Human Language Regular,
Context-free or Beyond?

48

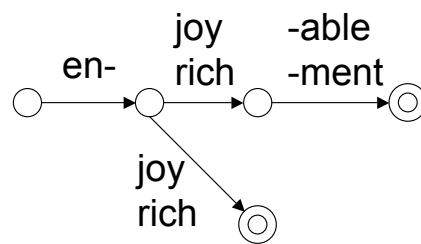
Natural Language & Complexity

- 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

49

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



$V \rightarrow X$
 $A \rightarrow X \text{-able} \mid X \text{-ment}$
 $X \rightarrow \text{en-} \mid \text{NA}$
 $\text{NA} \rightarrow \text{joy} \mid \text{rich}$

Regular grammars can be exponentially larger than equivalent CFGs

50

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)

51

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

52

Is NL regular: strong GC

- Assume that in principle we could process infinitely nested dependencies:
competence assumption
 - $S \rightarrow a S b$
 - $S \rightarrow \epsilon$
- The reason we cannot is because of lack of memory in pushdown automata: **performance** can be explained
 - $S1 \Rightarrow a1 S2 b1$
 - $\Rightarrow a1 a2 S3 b2 b1$
 - $\Rightarrow a1 a2 \dots aN S bN \dots b2 b1$
 - $\Rightarrow a1 a2 \dots aN bN \dots b2 b1$
- CFGs can easily obtain nested dependencies

53

Is NL regular: Weak GC

- Consider the following set of strings (sentences):
 - $S = \textit{if } S \textit{ then } S$
 - $S = \textit{either } S \textit{ or } S$
 - $S = \textit{the man who said } S \textit{ is arriving today}$
- 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*

54

Is NL regular: Weak GC

- The language is the set of strings
 $L = \{ ww' : w \text{ from } (a|b)^* \text{ and } w' \text{ is reversal of } w \}$
- L can be shown to be non-regular using the pumping lemma for regular languages
- L is context-free

55

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

56

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

57

Generating Crossing Dependencies

1: $S \rightarrow S B C$	$S1 \Rightarrow \mathbf{S2 B1 C1}$ (1)
2: $S \rightarrow a C$	$\Rightarrow \mathbf{S3 B2 C2 B1 C1}$ (1)
3: $a B \rightarrow a a$	$\Rightarrow \mathbf{a3 C3 B2 C2 B1 C1}$ (2)
4: $C B \rightarrow B C$	$\Rightarrow a3 \mathbf{B2 C3 C2 B1 C1}$ (4)
5: $B a \rightarrow a a$	$\Rightarrow \mathbf{a3 a2 C3 C2 B1 C1}$ (3)
6: $C \rightarrow b$	$\Rightarrow a3 a2 C3 \mathbf{B1 C2 C1}$ (4)
	$\Rightarrow a3 a2 \mathbf{B1 C3 C2 C1}$ (4)
	$\Rightarrow a3 \mathbf{a2 a1 C3 C2 C1}$ (3)
	$\Rightarrow a3 a2 a1 \mathbf{b3 C2 C1}$ (6)
	$\Rightarrow a3 a2 a1 b3 \mathbf{b2 C1}$ (6)
	$\Rightarrow a3 a2 a1 b3 b2 \mathbf{b1}$ (6)

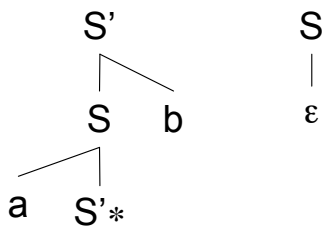
58

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

59

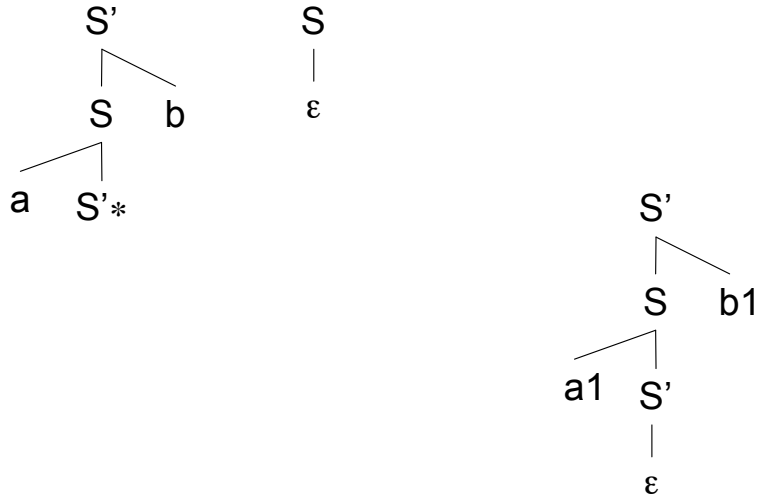
Crossing Dependencies with Adjoining



S
|
 ϵ

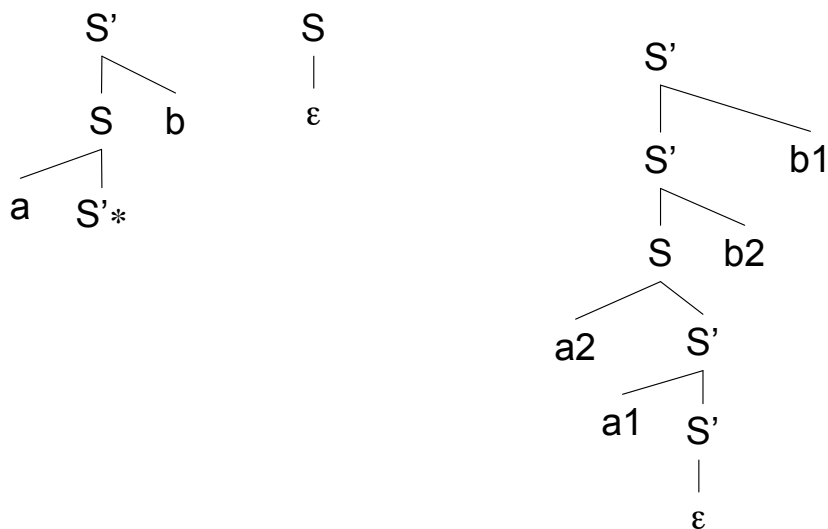
60

Crossing Dependencies with Adjoining



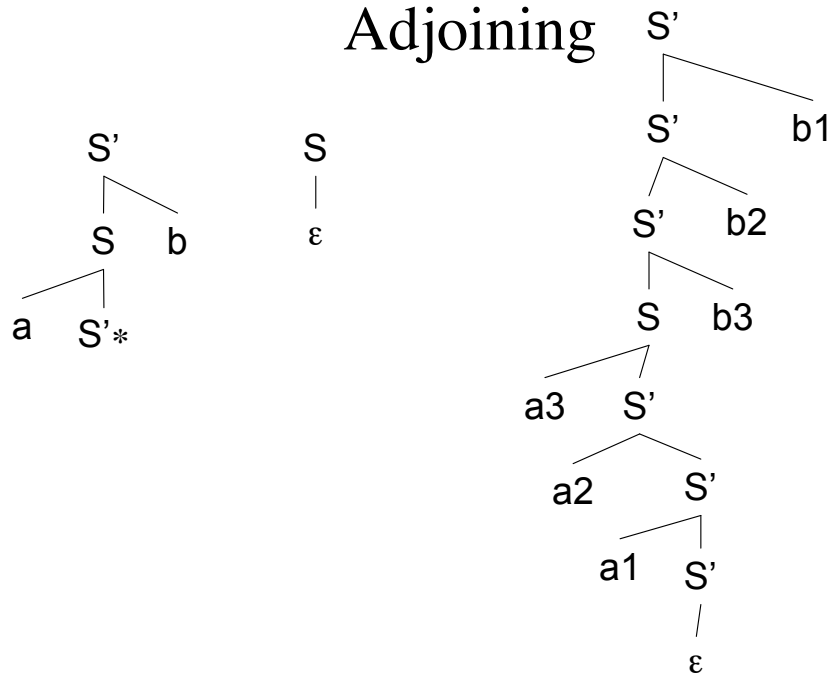
61

Crossing Dependencies with Adjoining



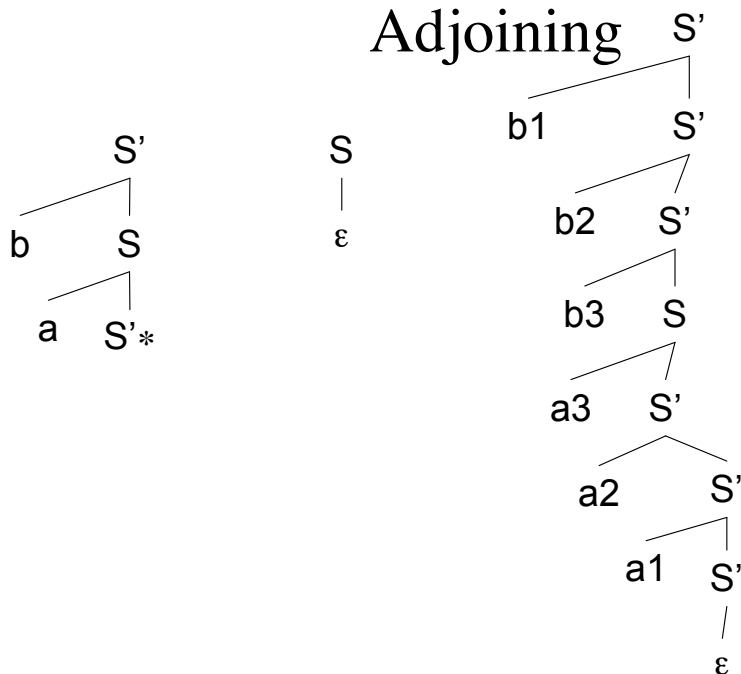
62

Crossing Dependencies with Adjoining



63

Nested Dependencies with Adjoining



64

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: $O(n)$

65

Tractable Descriptions

- Another route to lexicalization of CFGs: categorial grammars (CG is not strongly equivalent to CFGs but can lexicalize them)
- Several different mathematically precise formal grammars were proposed to deal with the motivations presented here
- Some examples: head grammars (HG does string wrapping); combinatory categorial grammars (CCG; extends CG); linear indexed grammars (LIG; less powerful than indexed grammars)
- Using formal methods introduced with TAG, (Vijay-Shanker, 1987) and (Weir, 1988) showed that HG, CCG, LIG and TAG are all *weakly equivalent!*

66

Tree-Adjoining Grammars: Definition and Application to NL

67

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*

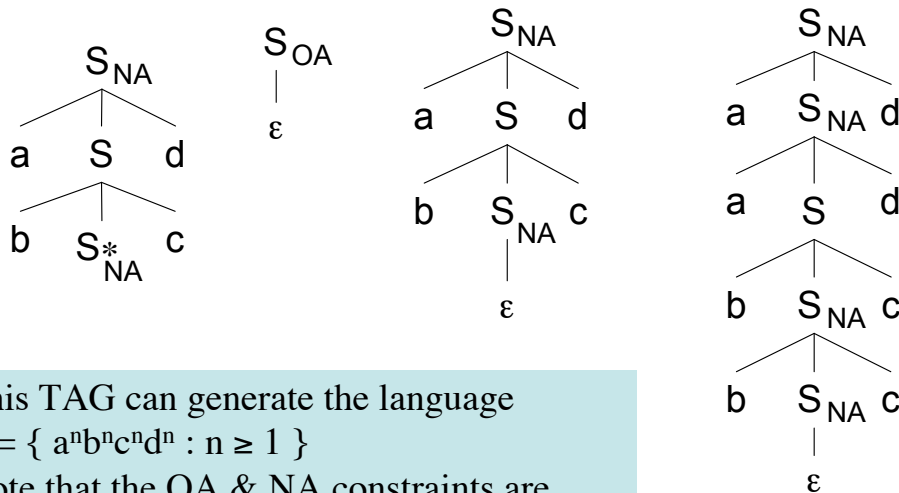
68

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 NLP and for formal closure properties
- 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

69

Adjunction Constraints

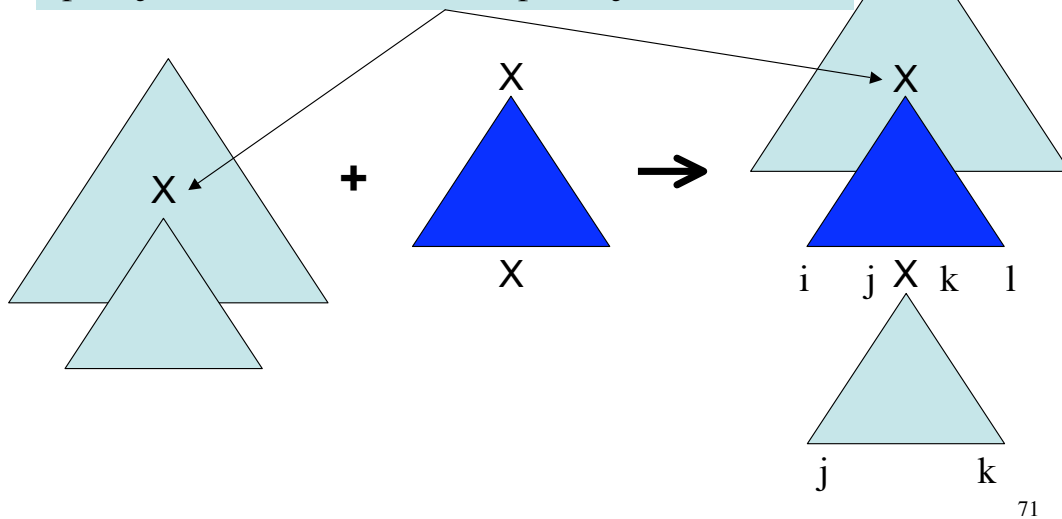


This TAG can generate the language
 $L = \{ a^n b^n c^n d^n : n \geq 1 \}$
 Note that the OA & NA constraints are crucial to obtain the correct language

70

Parsing Complexity: CKY for TAG

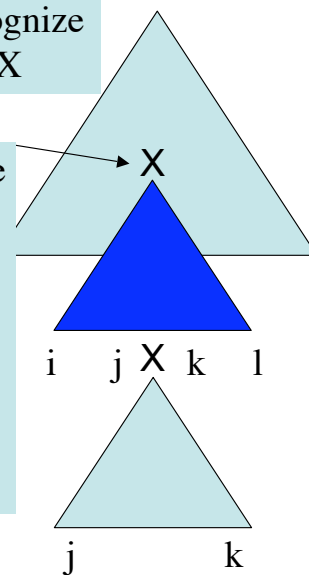
To recognize X with span (i,l) , we need to recognize span (j,k) and also deduce the span (i,j,k,l) for X



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

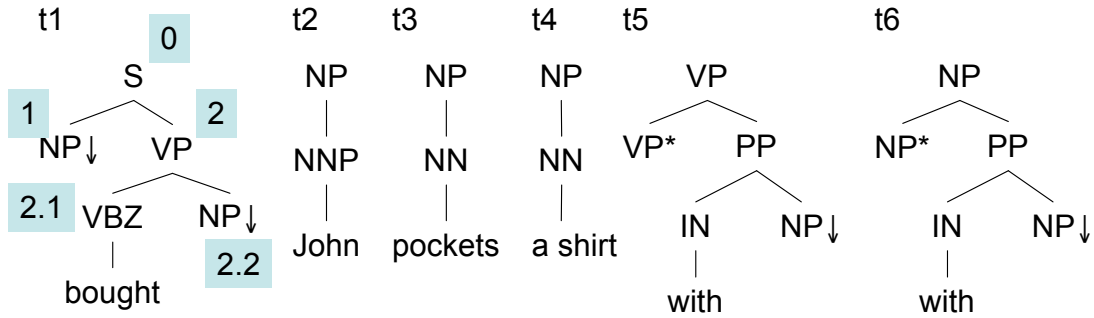
73

Lexicalized TAG

- A Lexicalized TAG (LTAG) is a TAG where each elementary tree has at least one terminal symbol as a leaf node
- A non-lexicalized TAG can always be converted to a lexicalized TAG (Joshi & Schabes, 1997)
- 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

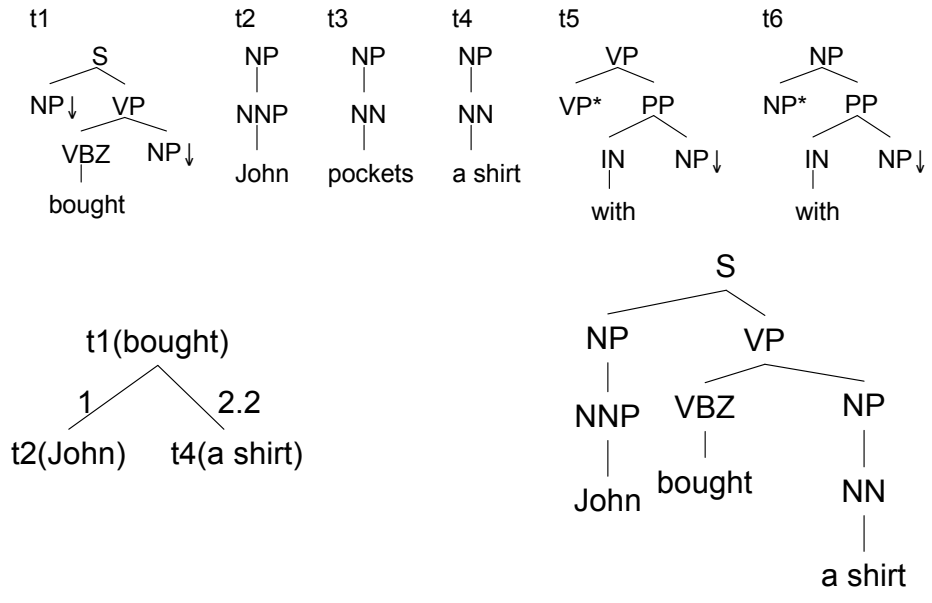
74

Lexicalized TAG: example



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

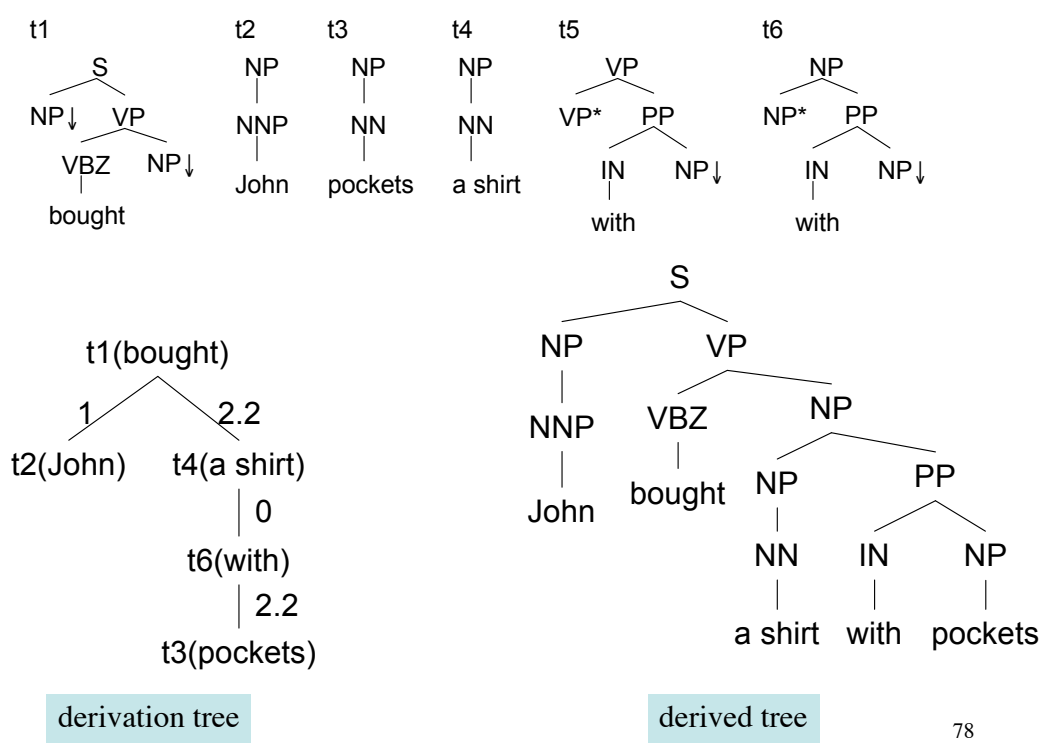
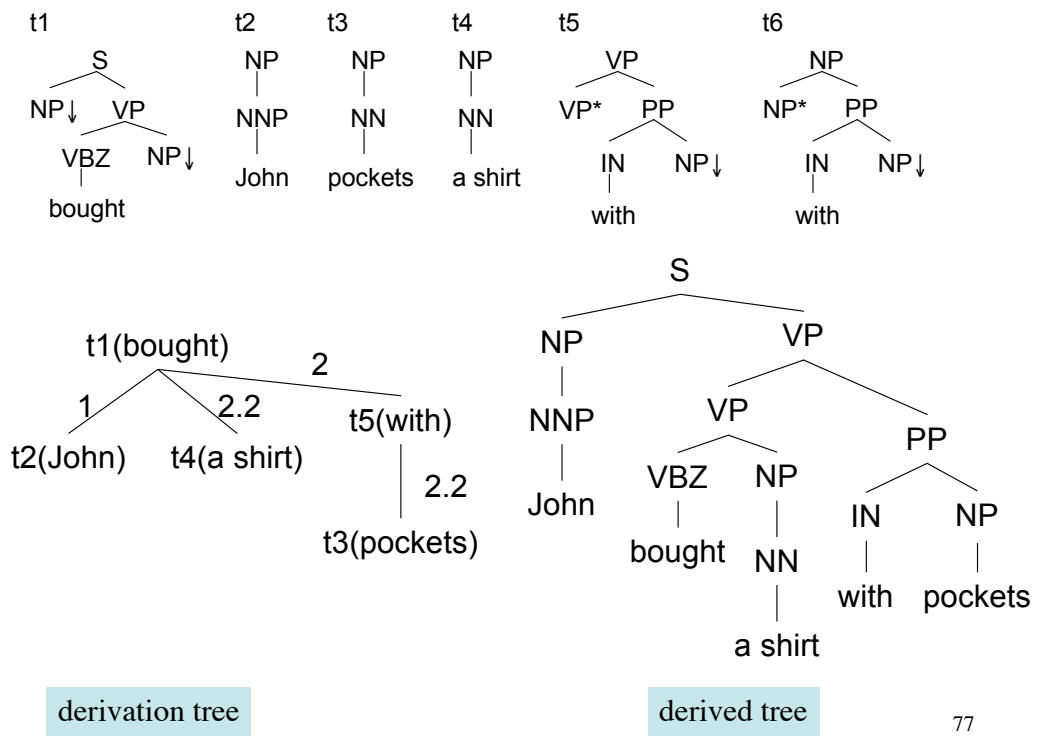
75



derivation tree

derived tree

76



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

79

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.

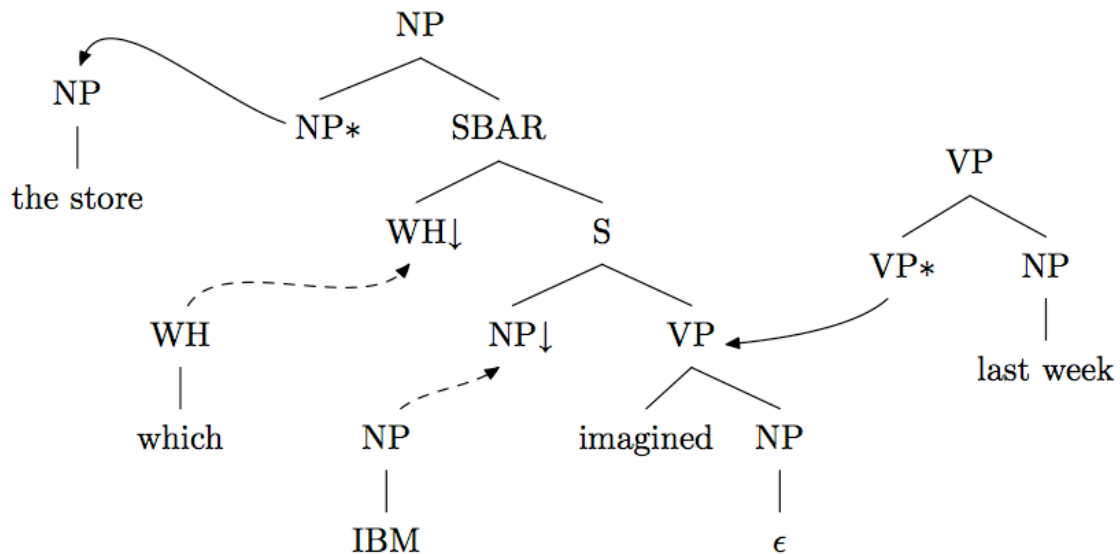
80

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-occurrences, lexical semantic aspects of word meaning

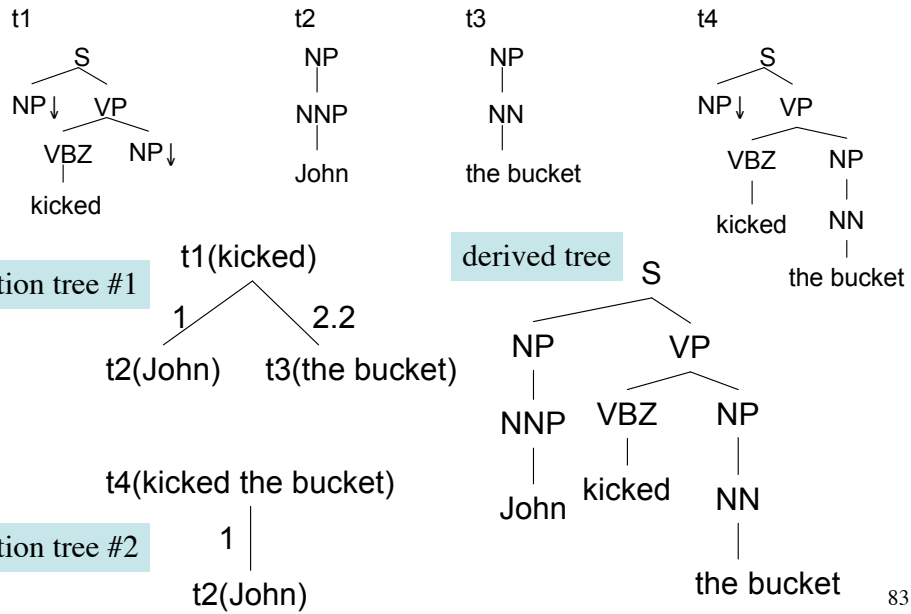
81

Localization of Dependencies



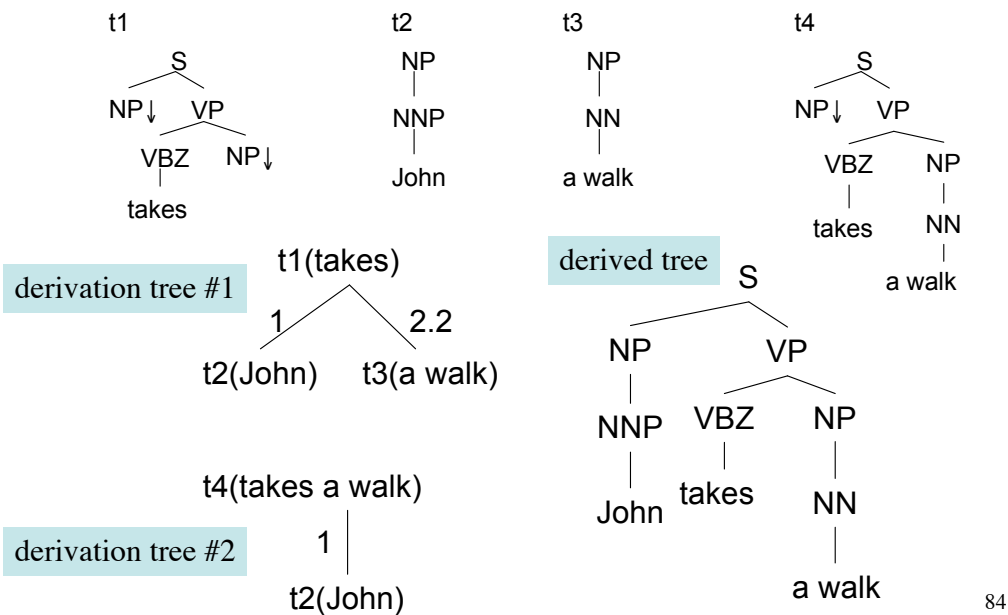
82

Idioms



83

Phrasal/Light Verbs



84

Mapping a TreeBank into Lexicalized TAG derivations

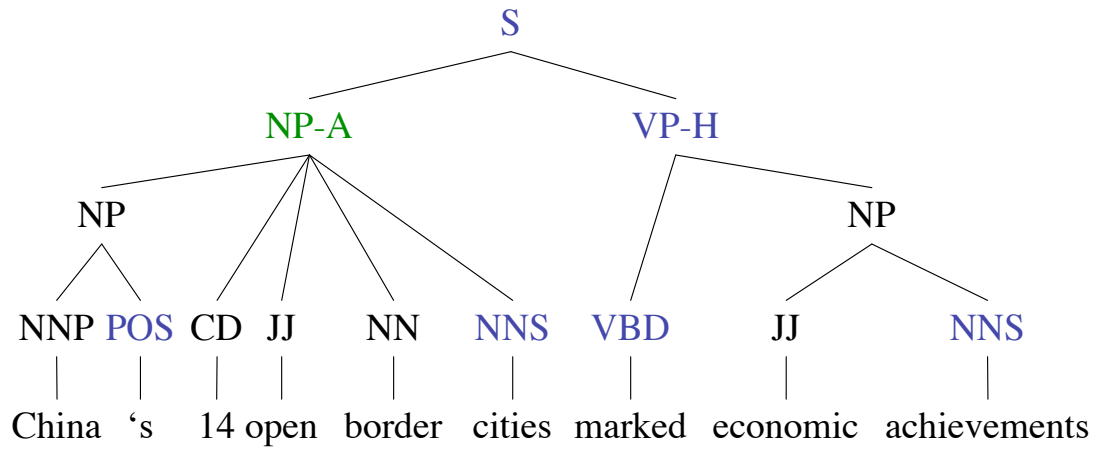
85

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) and (Chen, 2000)

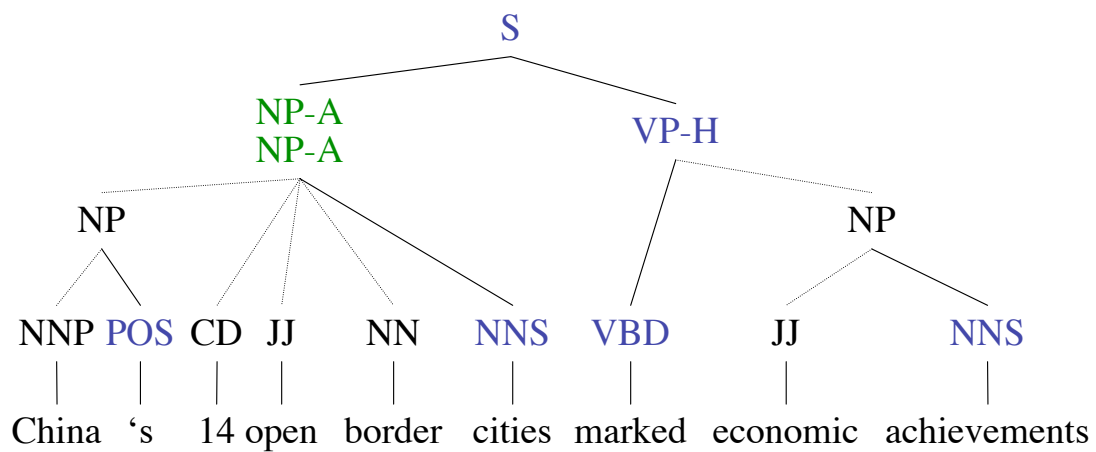
86

LTAG derivations from TreeBanks



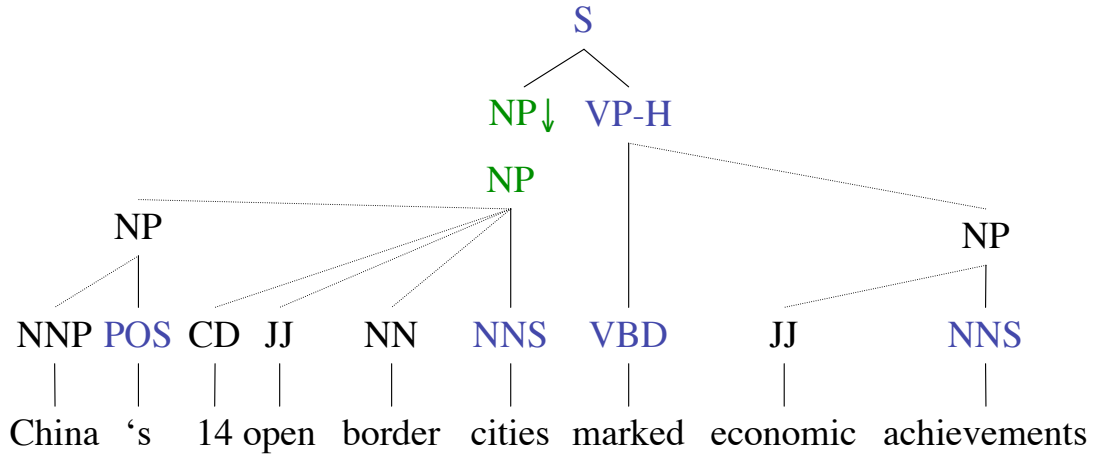
87

LTAG derivations from TreeBanks



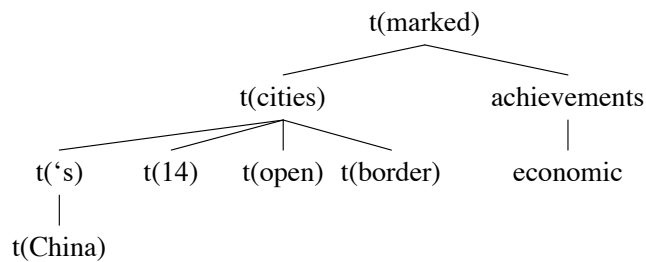
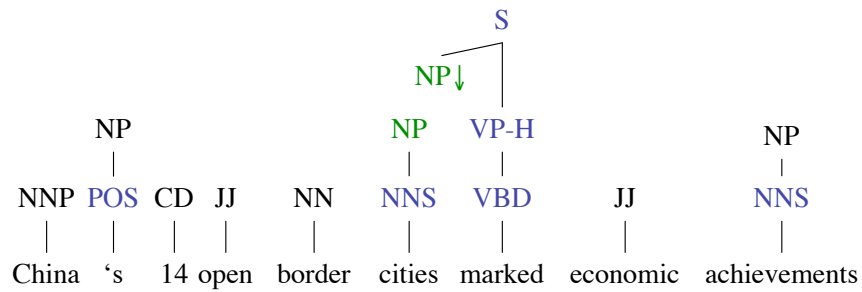
88

LTAG derivations from TreeBanks



89

LTAG derivations from TreeBanks



90

Synchronous TAG

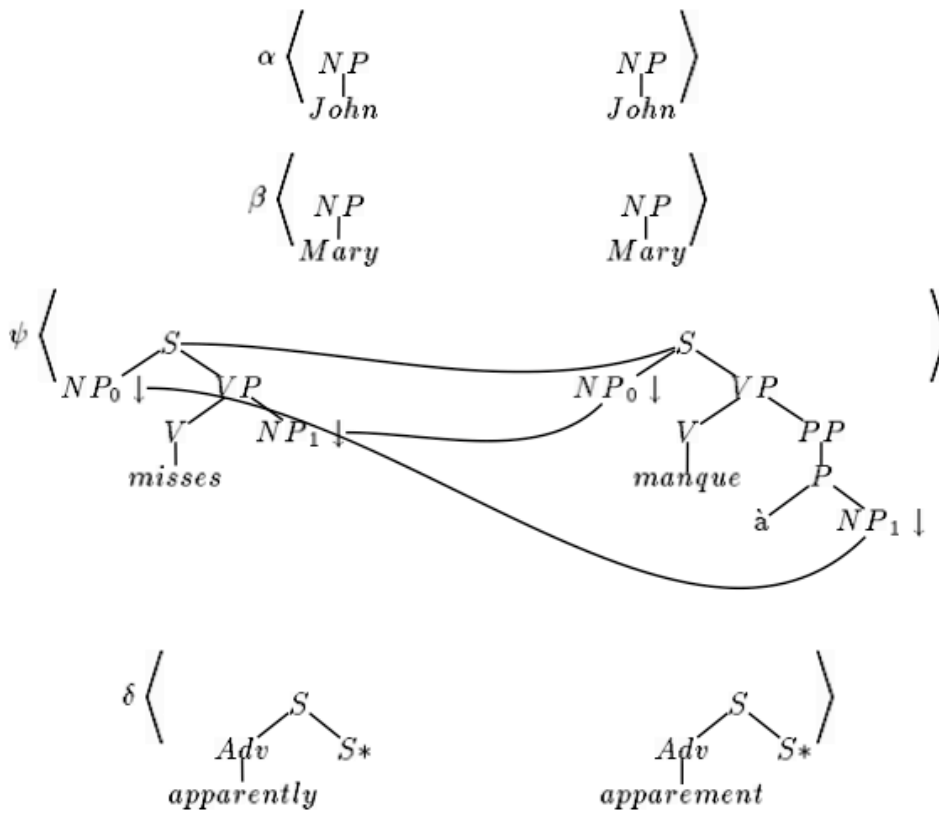
91

Synchronous TAG

(Shieber, 1994)

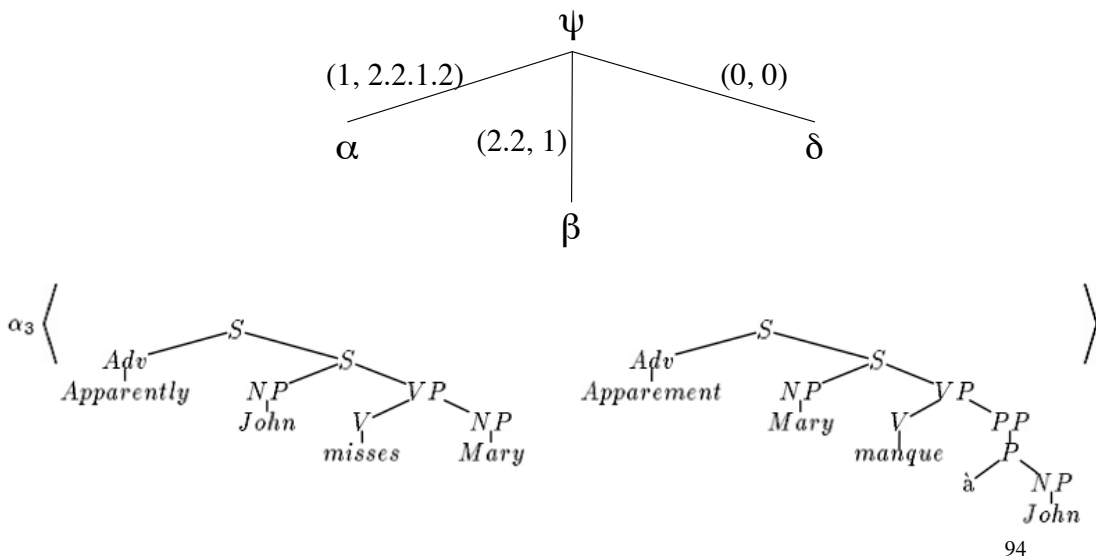
- Just like TAG we have derivation trees
- Except each node in the derivation is not a single elementary tree but rather a pair of trees
- The derivation tree now can be used to build a pair of derived trees
- Synchronous TAG can be used to generate a pair of derived trees or map a source input string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

92



93

Synchronous TAG



94

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, *Tiburón: 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

95

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.

96