# Tree-Adjoining Grammar

**Weiwei Sun**

Institute of Computer Science and Technology
Peking University

February 27, 2019

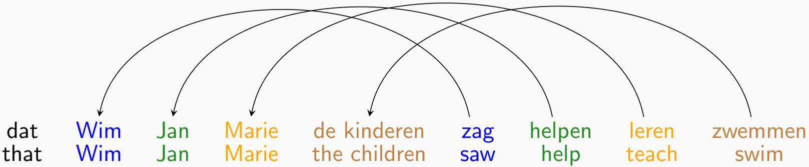
# Expressivity

## Question

Is CFG powerful enough to generate all sentences of a particular language?

## Theorem

*The copy language* $\{ww|w \in \{a,b\}^*\}$ *is not context-free.*



| dat | Wim | Jan | Marie | de kinderen | zag | helpen | leren | zwemmen |
|---|---|---|---|---|---|---|---|---|
| that | Wim | Jan | Marie | the children | saw | help | teach | swim |

We need extensions of CFG!

# Outline

Tree Substitution Grammar

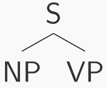
Tree-Adjoining Grammar

Grammar Extraction

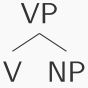
# Tree decomposition

- Elements of a CFG represent very small syntactic trees.

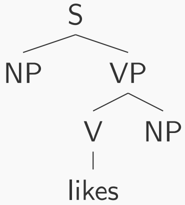
S→NP VP



VP→V NP



- We would rather have entire constructions as elementary building blocks.



This small tree encodes a subcategory of “*like*.”

# Tree combination

**If we've got a set of finite syntactic trees which have:**

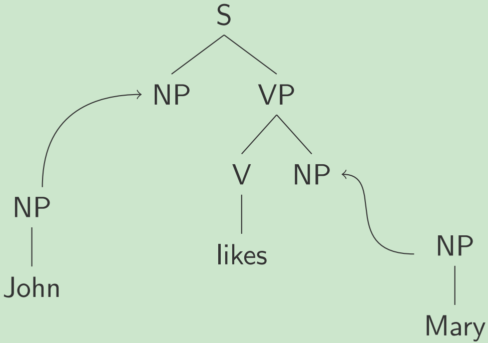
- internal nodes labeled with non-terminals, and
- leaves labeled either with terminals or non-terminals.

**Then we build larger trees by substitution:**

- Pick a non-terminal leaf (substitution node)
- Replace it with a tree the root node of which has the same label.

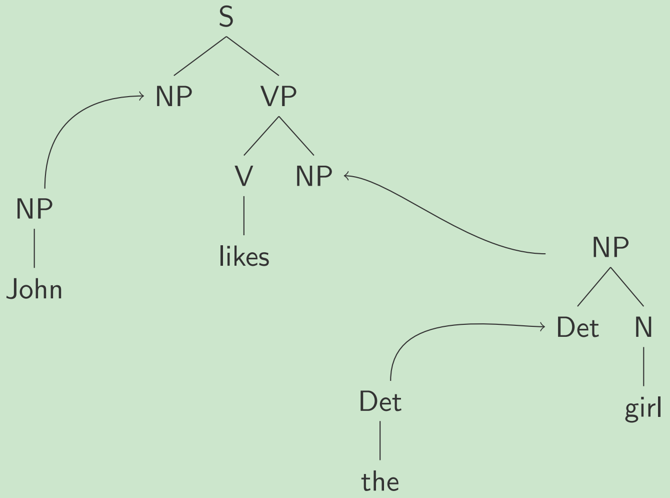
# Examples

## Example



# Examples

## Example



# TSG: Definition

## Definition (Tree substitution grammar)

A Tree Substitution Grammar is a tuple $G = \langle N, T, S, I \rangle$ where

- $N$: a set of non-terminals (syntactic categories)
- $T$: a set of terminals (words)
- $S \in N$: start symbol
- $I$: a finite set of syntactic trees with labels from $N$ and $T$.

Every tree in $I$ is called an elementary tree. $G$ is called lexicalized if every tree in $I$ has at least one leaf with a label from $T$.

## Derivation

- Select a node with a non-terminal label $A$.
- Pick an elementary tree with root label $A$ from the grammar.
- Substitute the node for the new tree.

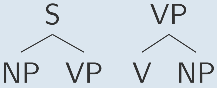
# TSG vs. CFG

CFG and TSG are weakly equivalent.

## CFG ⇒ TSG

Every CFG can be immediately written as a TSG with every production being understood as a tree with a single root and a daughter for every righthand side symbol.

- S→NP VP
- VP→V NP

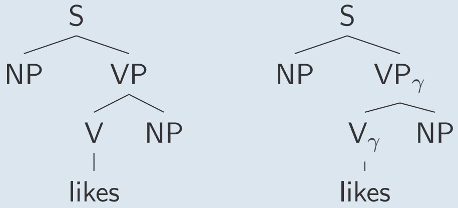


# TSG vs. CFG

CFG and TSG are weakly equivalent.

## TSG ⇒ CFG

In order to construct an equivalent CFG for a given TSG, we have to encode the dependencies between nodes from the same tree within the non-terminal symbols.



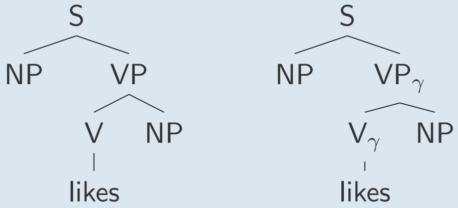
- $S \rightarrow NP\ VP_\gamma$
- $VP_\gamma \rightarrow V_\gamma\ NP$
- $V_\gamma \rightarrow likes$

# TSG vs. CFG

CFG and TSG are weakly equivalent.

## TSG ⇒ CFG

In order to construct an equivalent CFG for a given TSG, we have to encode the dependencies between nodes from the same tree within the non-terminal symbols.



- $S \rightarrow NP\ VP_\gamma$
- $VP_\gamma \rightarrow V_\gamma\ NP$
- $V_\gamma \rightarrow$ likes

But TSGs capture more generalizations than CFGs!

- Lexicalization, subcategorization, dependencies, ...

# Outline

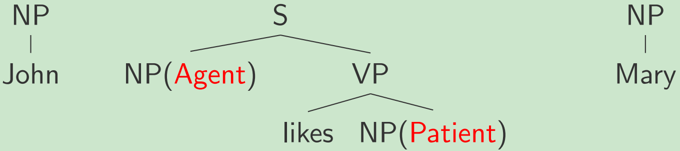
Tree Substitution Grammar

Tree-Adjoining Grammar

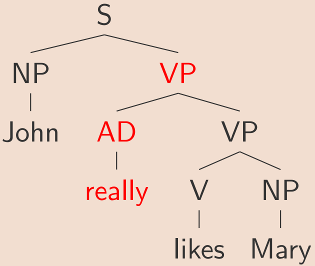
Grammar Extraction

# Problem

## Arguments



## How about adjuncts?

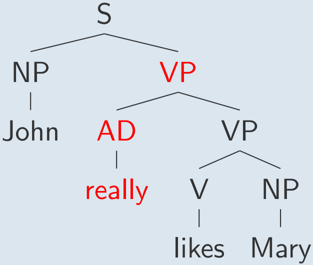


# Basic idea (1)

## Key idea of TSGs

Formalizing words/phrases as "**trees**."

## Extending the idea

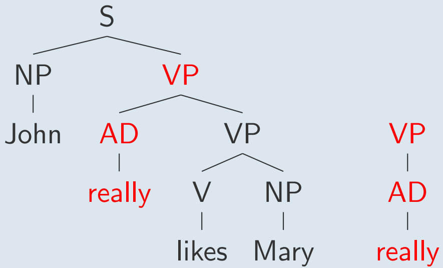


# Basic idea (1)

## Key idea of TSGs

Formalizing words/phrases as "**trees**."

## Extending the idea

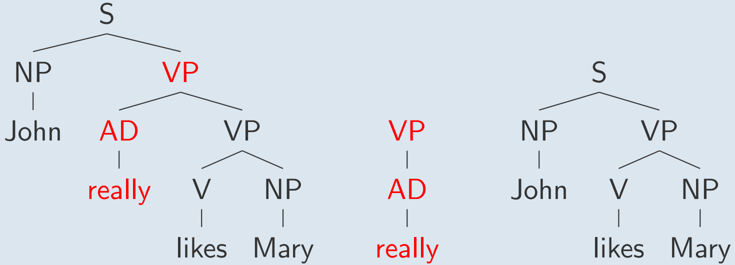


# Basic idea (1)

## Key idea of TSGs

Formalizing words/phrases as "**trees**."

## Extending the idea

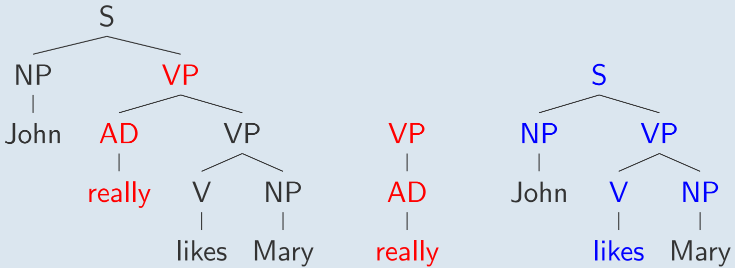


# Basic idea (1)

## Key idea of TSGs

Formalizing words/phrases as "**trees**."

## Extending the idea



# Basic idea (2)

## Key idea of TSGs

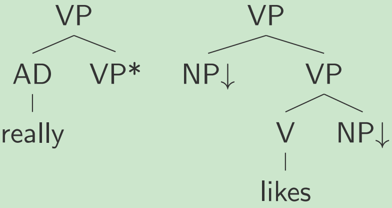
Formalizing words/phrases as “**trees**.”

## Extending the idea

A Tree Adjoining Grammar (TAG) is a set of elementary trees:

- a finite set of initial trees
- a finite set of auxiliary trees

## Example

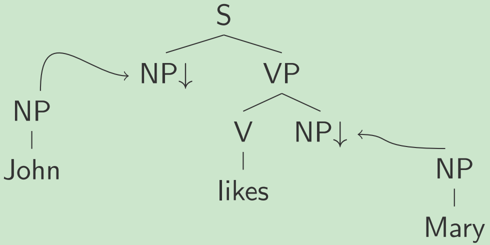


# Basic idea (3)

## Combinatorial operations

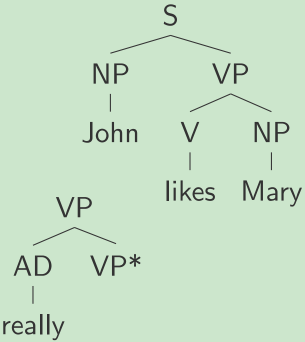
- Substitution: replacing a non-terminal leaf with an initial tree
- Adjunction: replacing an internal node with an auxiliary tree

## Substitution



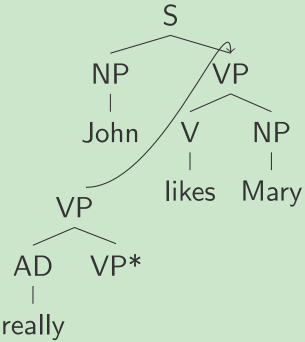
# Basic idea (4)

## Adjoining



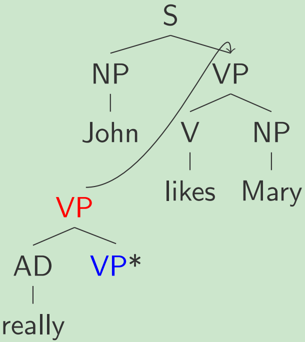
# Basic idea (4)

## Adjoining



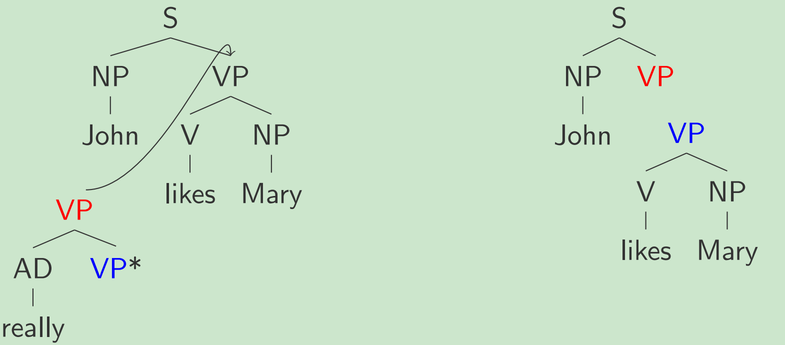
# Basic idea (4)

## Adjoining



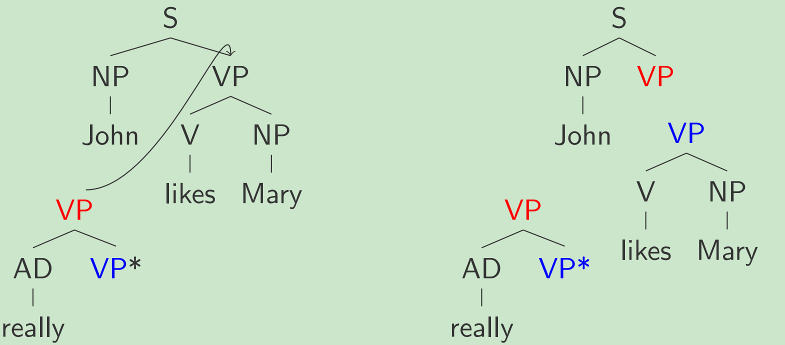
# Basic idea (4)

## Adjoining



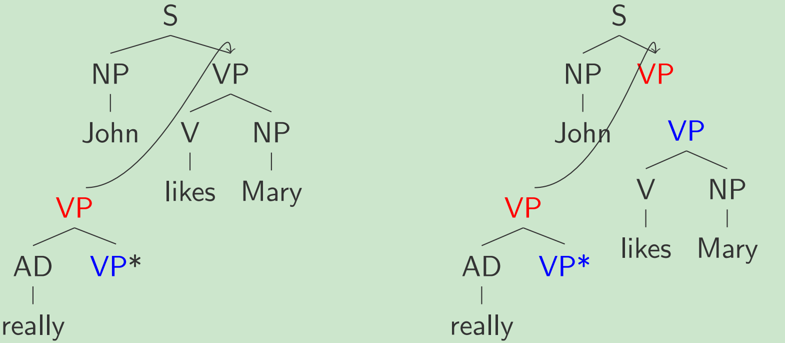
# Basic idea (4)

## Adjoining



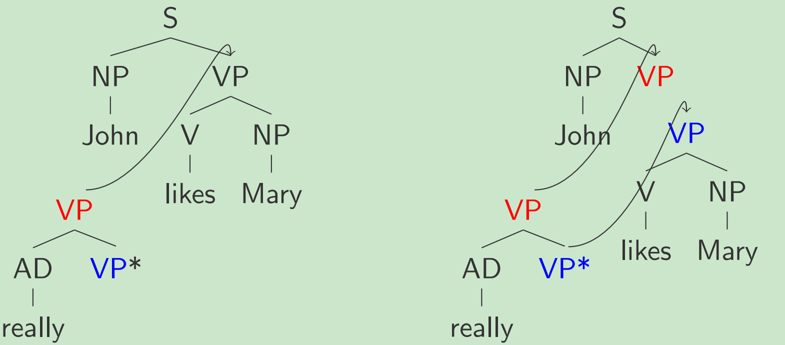
# Basic idea (4)

## Adjoining



# Basic idea (4)

## Adjoining



# TAG: Definition

## Definition

A Tree Adjoining Grammar (TAG) is a tuple $G = \langle N, T, S, I, A \rangle$ such that

- $N$: a set of non-terminals (syntactic categories)
- $T$: a set of terminals (words)
- $S \in N$: start symbol
- $I$: a finite set of initial trees
- $A$: a finite set of auxiliary trees

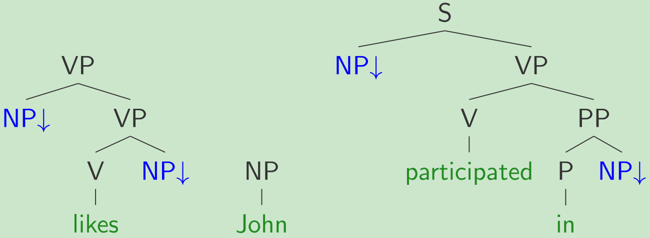
The trees in $I \cup A$ are called elementary trees.
$G$ is called lexicalized if each elementary tree has at least one leaf with a terminal label.

# Elementary trees (1)

## Initial tree

- **Interior nodes**: non-terminal symbols
- **Frontier nodes**: terminals or non-terminals
- Non-terminal nodes on the frontier: marked for substitution
    - we annotate nodes to be substitued with ↓.

## Example

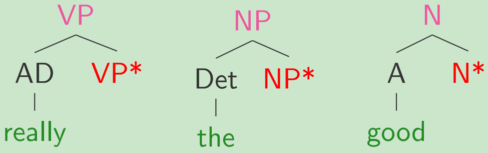


# Elementary trees (2)

## Auxiliary tree

- **Interior nodes**: non-terminal symbols
- **Frontier nodes**: terminals or non-terminals
- Non-terminal nodes on the frontier: marked for substitution except for one node, called the foot node
  - we annotate the foot node with $*$
  - labels of the foot node and the root node must be identical.

## Example



# Operations

## Adjoining (adjunction) operation

Adjoining (or adjunction) builds a new tree from an auxiliary tree $\beta$ and a tree $\alpha$ (initial, auxiliary or derived tree) by cutting $\alpha$ into two parts and inserting $\beta$ in between

- The node of the root of $\beta$ is identified with the node $Z$
- The node of the foot of $\beta$ is identified with the root of the excised tree
- $Z$ is not annotated for substitution/adjoining

It is convenient for linguistic description to have more precision for specifying which auxiliary trees can be adjoined at a given node.

# Derived tree & derivation tree in TAG

## Derived tree

Derived tree is the result of the derivations and represents the phrase structure

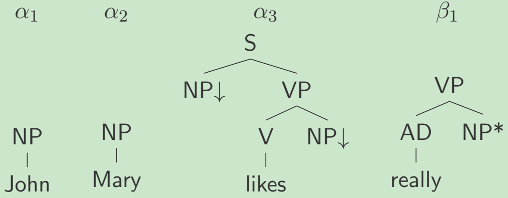
## Derivation tree

Derivation tree specifies how a derived tree was constructed.

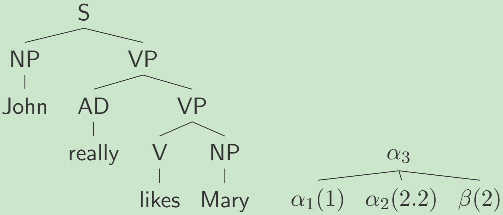
- The root is labeled by an S-type initial tree
- All other nodes are labeled by initial trees in the cases of substitutions, and auxiliary trees in the cases of adjoining
- A tree address is associated with each node to denote the node in the parent tree to which the derivation operation has been performed

# Example

## An example grammar



## An example sentence



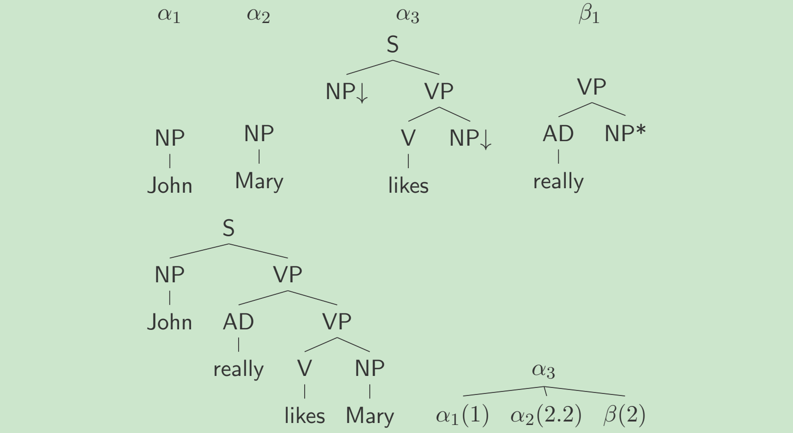
# TAG: Blah blah blah

TAG = TSG + adjunction + adjunction constrints.

- The definition of TAG goes back to
  - Joshi, Levy, and Takahashi. 1975. Tree adjunct grammars.
- TAG is among the most frequently used grammar formalisms in computational linguistics.
- TAG is a tree-rewriting formalism:
  - TAG defines operations (substitution, adjunction) on trees.
  - The elementary objects in TAG are trees.
    - CFG defines operations on non-terminals.
    - The elementary objects in CFG are strings.
- TAG is mildly context-sensitive:
  - TAG can capture Dutch cross-serial dependencies
- TAG is interesting both for its computational properties and for its linguistic applications.

# TAG can capture dependency relations

TAG captures the phrase structure and the dependency structure in the same time.

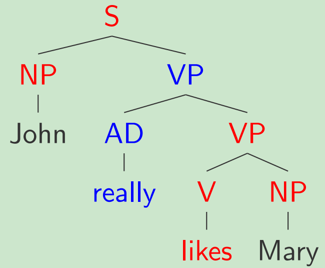


# Extended domain of locality (1)

TAG comes with an **extended domain of locality**

- CFG: the domain of locality is confined to a single rule.
  - Each local tree (rewrite rule) is independent.
- TAG: uses larger tree fragments as elementary building blocks.

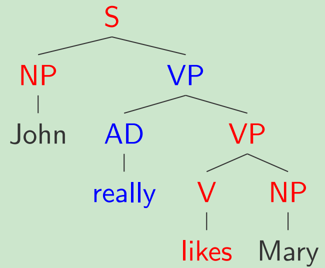
  ⇒ This makes non-local dependencies local.



# Extended domain of locality (2)

Recusion can be factored away by means of adjunction. Adjunction separates the specification of grammatical constraints from the recursive processes.

- TAG localizes the grammatical constraints within small pieces of phrase structure, i.e. elementary trees.
- Recursive structures are treated as auxiliary trees, which adjoin in to produce non-local dependencies.



# Extended domain of locality (3)

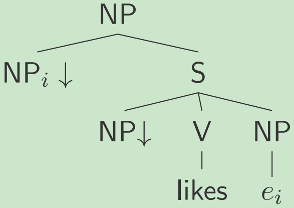
### Extended domain of locality (EDL)

Elementary trees can be of arbitrary size, so the domain of locality is increased

### Factoring recursion from the domain of dependency (FRD)

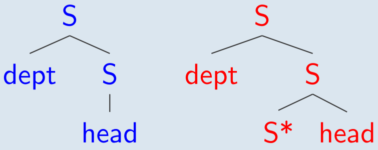
Small initial trees can have multiple adjunctions inserted within them, so what are normally considered non-local phenomena are treated locally

### **Extraction:** *likes* **in** *the girl John likes*

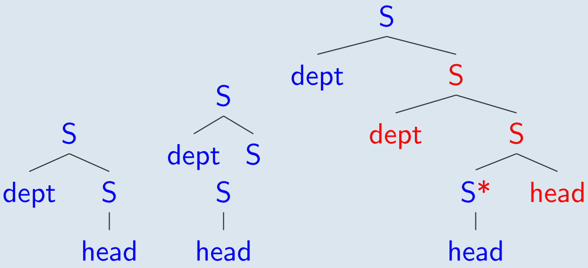


# Cross-serial dependencies (1)

## Elementary trees

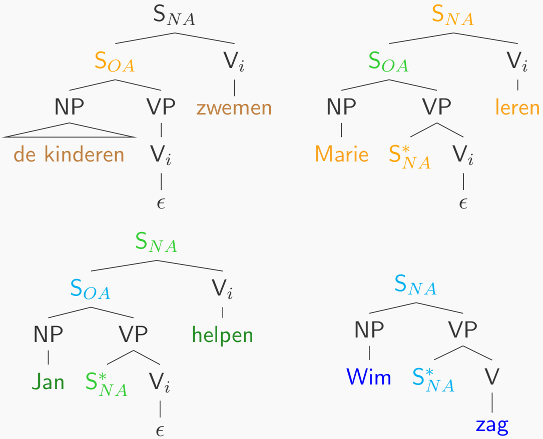


## Deriving "**dept** **dept** **head** **head**"



# Cross-serial dependencies (2)

... dat Wim Jan Marie de kinderen zag helpen leren zwemmen
... that Wim Jan Marie the children saw help teach swim
... that Wim saw Jan help Marie teach the children to swim



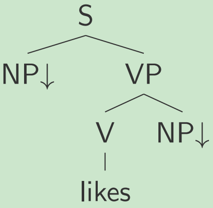
# Quote

## D. Dowty

*As is frequently pointed out but cannot be overemphasized, an important goal of formalization in linguistics is to enable subsequent researchers to see the defects of an analysis as clearly as its merits; only then can progress be made efficiently.*

## Example

S→NP VP
VP→V NP
V→likes
...



# TAG vs. CFG

## The CFG way

- Define simple elementary objects (e.g. words)
- Define various operations to combine these objects.
- Introduce new operations to deal with more complex structures.

## The TAG way

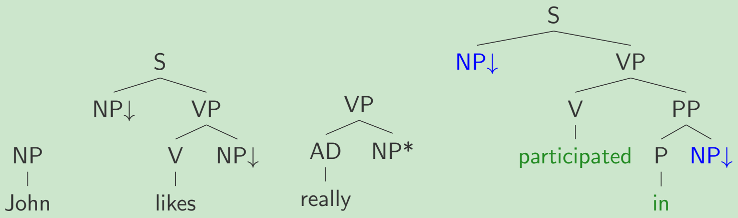
- Define complex elementary objects (e.g. trees) that capture crucial linguistic properties.
- Define simple, general operations to combine these objects.
- Similar to `CG`.

# LTAG: Lexicalized TAG

## Lexicalization

- Each elementary tree is anchored to one (or more) terminal (word)
- The elementary tree contains all arguments of the anchor.
- LTAG requires a linguistic theory which specifies the shape of these elementary trees.

## Example



# Outline

Tree Substitution Grammar

Tree-Adjoining Grammar

Grammar Extraction

# Linguistic motivation

## Some linguists favor

- **Lexicalization**: Each elementary tree has at least one non-empty lexical item. Elementary trees can even have more than one anchor.
- **Predicate argument co-occurrence**: each predicate contains in the elementary tree associated with it argument slots
    - substitution nodes or foot nodes for each of its arguments
- **Elementary tree minimality**: an elementary tree contain argument slots only for the arguments of its lexical anchor, and for nothing else.
- **Compositionality principle**: an elementary tree corresponds to a single semantic unit.

# Where can I find a LTAG grammar?

- Hand-crafting a grammar
- Translating existing (usually shallow) resources/treebanks into this formalisms
  - ⇐ Treebanks for many languages are available: English, Chinese, German, Czech, Arabic, ...

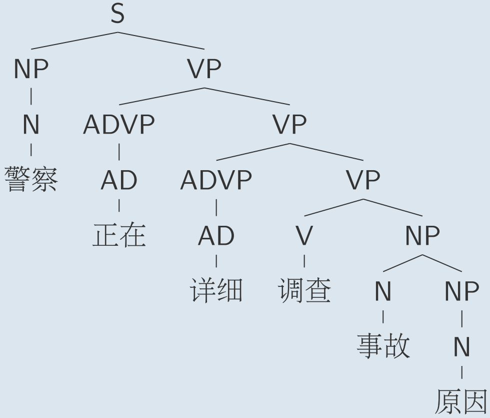
## Treebank-driven approach

- F. Xia. 2001. Automatic Grammar Generation From Two Different Perspectives. PhD thesis, UPenn.
- J. Chen, S. Bangalore, K. Vijay-Shanker. 2005. Automated Extraction of Tree-Adjoining Grammars from Treebanks. *Natural Language Engineering*.

Grammar extraction in other formalisms (HPSG, CCG, LFG) use similar methods, and yield similar results.

# Extracting a LTAG from the Penn Treebank

## Input: a PTB tree = the TAG derived tree

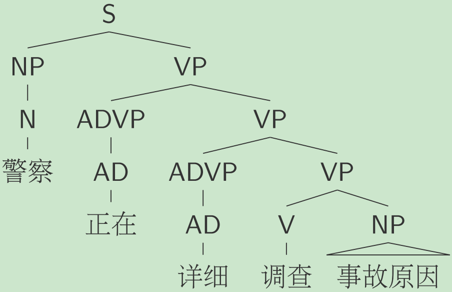


## Output: a set of elementary trees = the TAG lexicon

???

# Extracting a TAG: Head

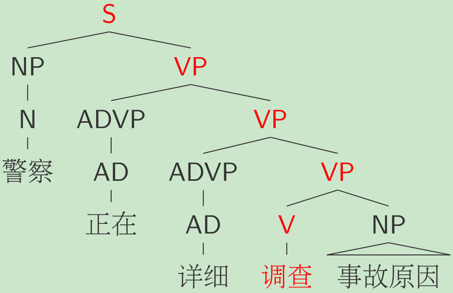
## Example



# Extracting a TAG: Head

- Identify the head path

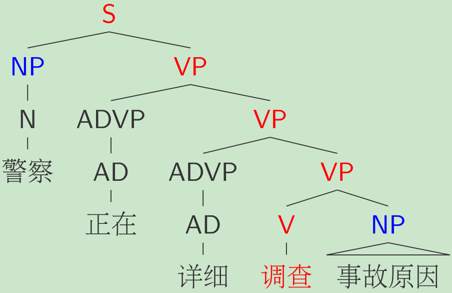
## Example



# Extracting a TAG: Head

- Identify the head path
- Find the arguments of the head

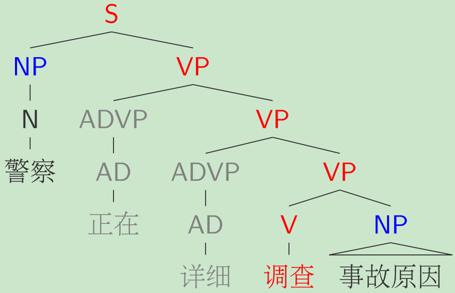
## Example



# Extracting a TAG: Head

- Identify the head path
- Find the arguments of the head
- Ignore modifiers

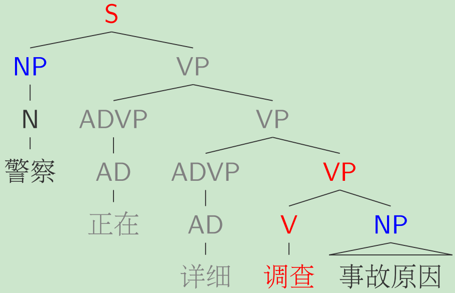
## Example



# Extracting a TAG: Head

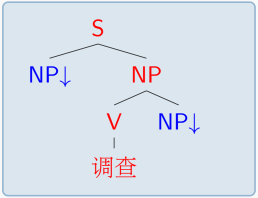
- Identify the head path
- Find the arguments of the head
- Ignore modifiers
- Merge unary productions (VP→VP)

**Example**

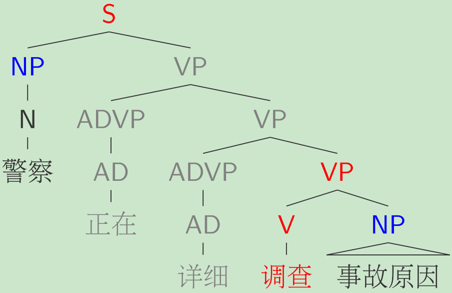


# Extracting a TAG: Head

- Identify the head path
- Find the arguments of the head
- Ignore modifiers
- Merge unary productions (VP→VP)



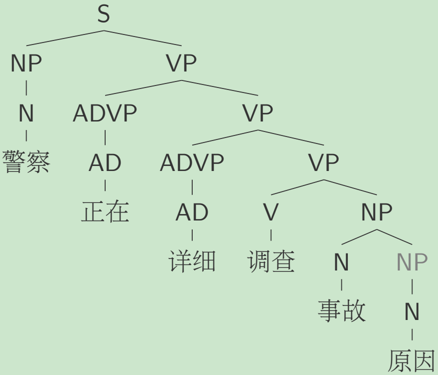
## Example



# Extracting a TAG: Argument

- Arguments are combined via substitution

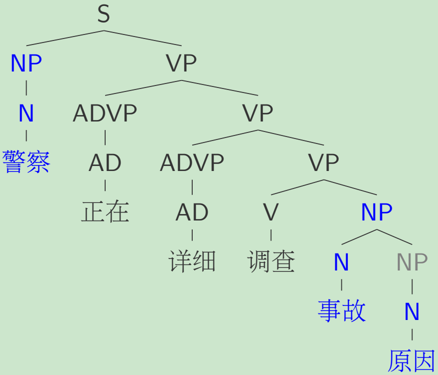
## Example



# Extracting a TAG: Argument

- Arguments are combined via substitution
- Recurse on the arguments

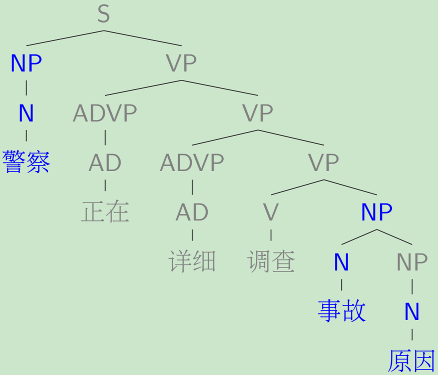
## Example



# Extracting a TAG: Argument

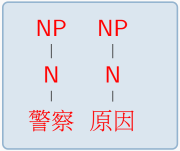
- Arguments are combined via substitution
- Recurse on the arguments

## Example

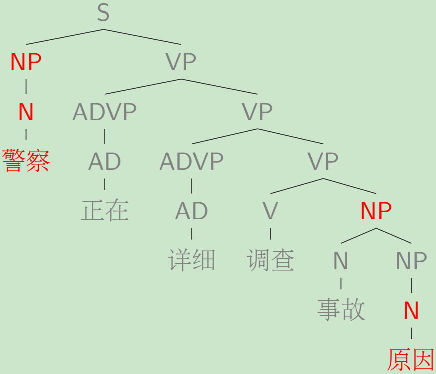


# Extracting a TAG: Argument

- Arguments are combined via substitution
- Recurse on the arguments

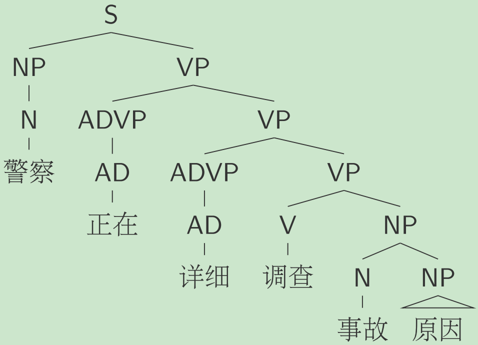


## Example



# Extracting a TAG: Adjunct

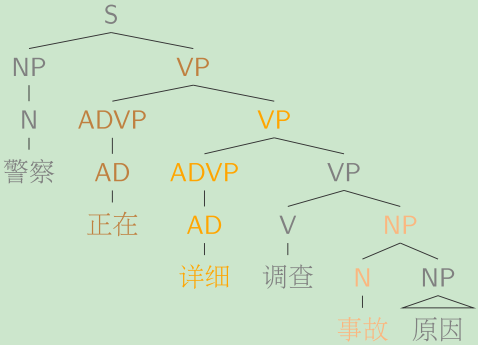
## Example



# Extracting a TAG: Adjunct

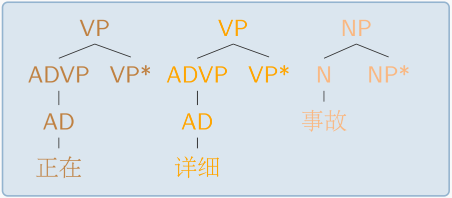
- Adjuncts require auxiliary trees

**Example**

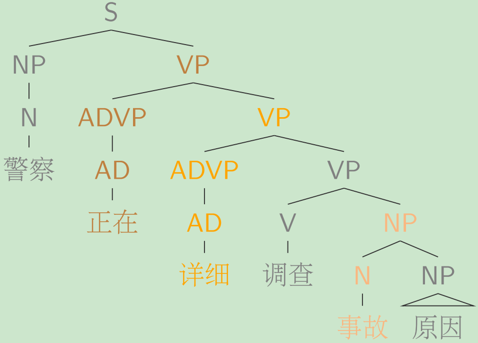


# Extracting a TAG: Adjunct

- Adjuncts require auxiliary trees
- Auxiliary trees require a foot node

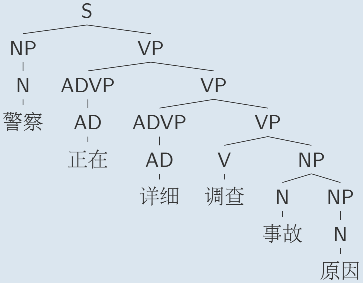


## Example

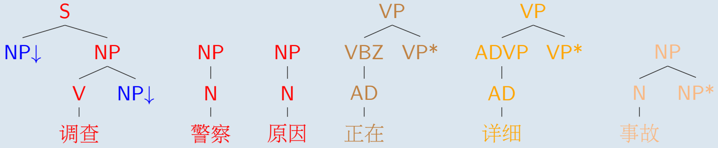


# Extracting a TAG from PTB

## Input: a PTB tree = the TAG derived tree



## Output: a set of elementary trees = the TAG lexicon



# Reading & homework

- Chapter 12. *Grammatical theory: From transformational grammar to constraint-based approaches*
- Chapter 2. *Phrase Structure Composition and Syntactic Dependencies*

## Homework

- 自选10个汉语句子，试用TAG进行分析，并谈谈你对使用TAG分析汉语的感想。
- 作业必须用latex完成，提交pdf版本。文件命名方式为：<学号> + hw01.pdf

  画树推荐使用tikz-qtree
- 作业在3月20日之前发送到我的邮箱，邮件命名方式为：形式语法导论2019作业01 + <学号>