Chomsky Hierarchy

Formal languages

We have seen two formalisms for representing linguistic phenomena:

- Regular expressions and finite state automata (FSA)
- Phrase-structure grammars. In formal language theory, these are called **Context-free grammars (CFG)**

Formal languages

We have seen two formalisms for representing linguistic phenomena:

- Regular expressions and finite state automata (FSA)
- Phrase-structure grammars. In formal language theory, these are called **Context-free grammars (CFG)**

Formal languages are different in

- expressive power
- speed of inference

Formal languages

We have seen two formalisms for representing linguistic phenomena:

- Regular expressions and finite state automata (FSA)
- Phrase-structure grammars. In formal language theory, these are called **Context-free grammars (CFG)**

Formal languages are different in

- expressive power
- speed of inference

In particular,

- CFGs are more expressive than regular expressions/FSAs
- They are also much slower: $\mathcal{O}(n^3)$ vs $\mathcal{O}(n)$

Chomsky Hierarchy

Chomsky (1956) classifies formal languages by the rules they can contain.

Type	Name	Rules allowed	Example
0	Turing complete	$\alpha \to \beta$	LFG, Python
1	Context sensitive	$\alpha A \beta \to \alpha \gamma \beta$	
-	Mildly context sensitive		CCG
2	Context free	$A \to \gamma$	PSG
3	(Right) regular	$A \to aB$ or $A \to a$	FSA, regex

Here,

- a is a terminal symbol
- A and B are non-terminals
- α, β, γ are strings of both terminal and non-terminal symbols

Example CFG

`S`	$\rightarrow$	`NP VP`	*Subject-predicate*
`NP`	$\rightarrow$	`Pronoun | ProperNoun`	*One way to list alternatives*
	|	`Det Nominal`	*Another way*
`Nominal`	$\rightarrow$	`Nominal Noun`	
	|	`Noun`	
`VP`	$\rightarrow$	`Verb`	*Intransitive verb*
	|	`Verb NP`	*Transitive verb*
	|	`Verb PP`	*Oblique / adjunct*
	|	`Verb NP PP`	*Oblique / adjunct*
`PP`	$\rightarrow$	`Preposition NP`	

Regular language and FSAs

There are two types of regular languages, based on the rules allowed:

- **Right regular grammar**
  - $A \to a$
  - $A \to aB$
  - $A \to \epsilon$

Regular language and FSAs

There are two types of regular languages, based on the rules allowed:

- **Right regular grammar**
  - $A \to a$
  - $A \to aB$
  - $A \to \epsilon$
- **Left regular grammar**
  - $A \to a$
  - $A \to Ba$
  - $A \to \epsilon$

Regular language and FSAs

There are two types of regular languages, based on the rules allowed:

- **Right regular grammar**
  - $A \to a$
  - $\boldsymbol{A \to aB}$
  - $A \to \epsilon$
- **Left regular grammar**
  - $A \to a$
  - $\boldsymbol{A \to Ba}$
  - $A \to \epsilon$

Regular language and FSAs

There are two types of regular languages, based on the rules allowed:

- **Right regular grammar**
  - $A \rightarrow a$
  - $\boldsymbol{A \rightarrow aB}$
  - $A \rightarrow \epsilon$
- **Left regular grammar**
  - $A \rightarrow a$
  - $\boldsymbol{A \rightarrow Ba}$
  - $A \rightarrow \epsilon$

The two are completely equivalent, but the two types of rules cannot be mixed.

Regular language and FSAs

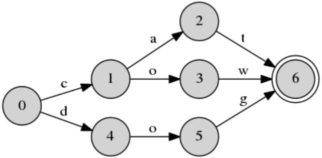
There are two types of regular languages, based on the rules allowed:

- **Right regular grammar**
  - $A \rightarrow a$
  - $\boldsymbol{A \rightarrow aB}$
  - $A \rightarrow \epsilon$
- **Left regular grammar**
  - $A \rightarrow a$
  - $\boldsymbol{A \rightarrow Ba}$
  - $A \rightarrow \epsilon$

The two are completely equivalent, but the two types of rules cannot be mixed.

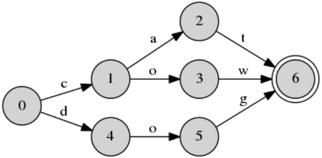
FSAs can simply be transcribed into right-regular grammars by equating FSA states with non-terminals.

Regular language example



Can be converted to:

Regular language example



Can be converted to:

$$
\begin{aligned}
(0) &\rightarrow \texttt{c}\ (1)\\
(0) &\rightarrow \texttt{d}\ (4)\\
(1) &\rightarrow \texttt{a}\ (2)\\
(1) &\rightarrow \texttt{o}\ (3)\\
(2) &\rightarrow \texttt{t}\ (6)\\
(3) &\rightarrow \texttt{w}\ (6)\\
(4) &\rightarrow \texttt{o}\ (5)\\
(5) &\rightarrow \texttt{g}\ (6)\\
(6) &\rightarrow \epsilon
\end{aligned}
$$

The hierarchy again...

Type	Name	Rules allowed	Example
0	Turing complete	$\alpha \to \beta$	LFG, Python
1	Context sensitive	$\alpha A \beta \to \alpha \gamma \beta$	
2	Context free	$A \to \gamma$	PSG
3	(Right) regular	$A \to aB$ or $A \to a$	FSA, regex

The hierarchy again...

Type	Name	Rules allowed	Example
0	Turing complete	$\alpha \to \beta$	LFG, Python
1	Context sensitive	$\alpha A \beta \to \alpha\gamma\beta$	
2	Context free	$A \to \gamma$	PSG
3	(Right) regular	$A \to aB$ or $A \to a$	FSA, regex

Some observations:

The hierarchy again...

Type	Name	Rules allowed	Example
0	Turing complete	$\alpha \rightarrow \beta$	LFG, Python
1	Context sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$	
2	Context free	$A \rightarrow \gamma$	PSG
3	(Right) regular	$A \rightarrow aB$ or $A \rightarrow a$	FSA, regex

Some observations:

- Lower hierarchy levels are subsets of those above; e.g.
  - Regular languages are a subset of context-free languages
  - Right (left) regular rules $A \rightarrow aB$ are valid CFG rules

The hierarchy again...

Type	Name	Rules allowed	Example
0	Turing complete	$\alpha \rightarrow \beta$	LFG, Python
1	Context sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$	
2	Context free	$A \rightarrow \gamma$	PSG
3	(Right) regular	$A \rightarrow aB$ or $A \rightarrow a$	FSA, regex

Some observations:

- Lower hierarchy levels are subsets of those above; e.g.
  - Regular languages are a subset of context-free languages
  - Right (left) regular rules $A \rightarrow aB$ are valid CFG rules
- Turing complete problems are those that can be computed by an algorithm
  - What cannot: e.g. the *Halting problem*: from a program code and its input, tell whether the program will finish running

Formal languages: questions

Q. What is it that cannot be expressed with a regular expression?

A. **Center embedding**:

(1) The cat likes tuna fish.

(2) The cat the dog chased likes tuna fish.

(3) The cat the dog the rat bit chased likes tuna fish.

With regular expressions, we could model it with $\texttt{NP}^{\texttt{i}}$ $\texttt{V}^{\texttt{i}}$ `tuna fish`.
Unfortunately, the closest we can get is $\texttt{NP}^{+}$ $\texttt{V}^{+}$ `tuna fish`.

Formal languages: questions

Q. What is it that cannot be expressed with a regular expression?

A. **Center embedding**:

(1) The cat likes tuna fish.

(2) The cat the dog chased likes tuna fish.

(3) The cat the dog the rat bit chased likes tuna fish.

With regular expressions, we could model it with $\texttt{NP}^{\texttt{i}}\ \texttt{V}^{\texttt{i}}\ \texttt{tuna fish}$.
Unfortunately, the closest we can get is $\texttt{NP}^{+}\ \texttt{V}^{+}\ \texttt{tuna fish}$.

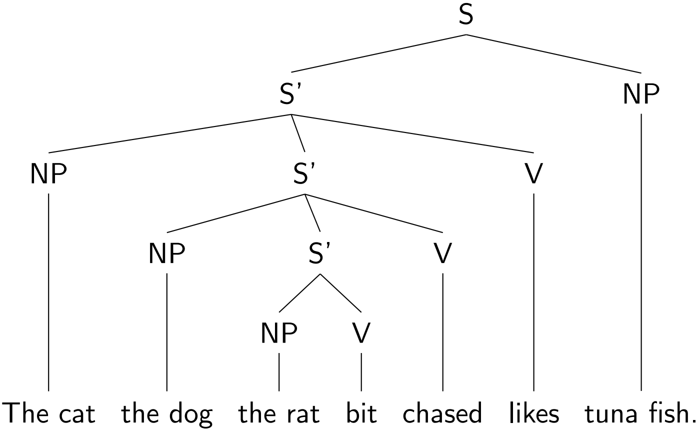
With CFG, it is easy:

$$\begin{array}{lcl} \texttt{S} & \rightarrow & \texttt{S' tuna fish} \\ \texttt{S'} & \rightarrow & \texttt{NP S' V} \mid \epsilon \end{array}$$

Formal languages: questions

Q. Can CFGs express everything in natural grammars?

A. CFG could deal with center embedding, because it could generate the `NP-V` pairs from the inside out:



Formal languages: questions

Q. Can CFGs express everything in natural grammars?

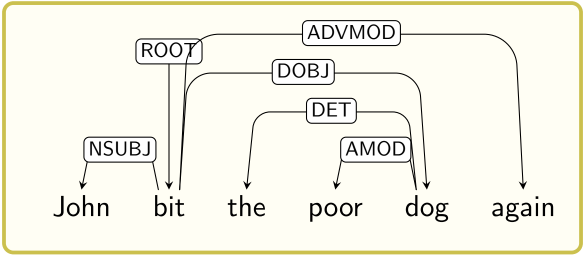
A. In Swiss German however, cross-serial dependencies also occur:

(1) *(Jan säit das) mer¹ d'chind² em Hans³ es*
(Jan says that) we¹ the children/ACC² Hans/DAT³ the
huus⁴ haend wele¹ laa² hälfe³ aastriiche⁴.
house/ACC⁴ have wanted¹ to let² help³ paint⁴.
'(Jan says that) we¹ have wanted¹ to let² the children² help³ Hans³ paint⁴ the house⁴.'

Such phenomena can only be modelled with (mildly) context-sensitive grammars.

Dependency parsing

Dependency graphs – introduction



- represents sentences as directed, acyclic graphs (DAGs) of words, whose edges are labeled with grammatical relations
- reflects *grammatical dependencies*
- does NOT reflect *syntactic constituency*

Dependency graphs – some features

- preferred in ML-based NLP systems
- quasi-universal (see later)
- very fast parsers available
- many treebanks for many languages
- many parsers using various methods

Dependency parsing – introduction

- Given a sentence (usually with POS-tags), find the most probable dependency graph
- A supervised learning task – many annotated treebanks are available
- Language-independent approaches are on the rise (see later)
- Important and also very popular, see e.g. the CoNLL 2017 Shared Task

Dependency parsing – approaches

- Arc-factored parsing
  - train an ML system to score edges of a dependency graph
  - for each new sentence, find the tree with the largest total score
- Transition-based parsing
  - build graphs by adding one word at a time
  - train an ML system to predict the most probable next step for any intermediate configuration

Arc-factored parsing (1)

Objective: learn a scoring function for edges that assigns the best total score to the most likely analysis

- represent edges with features F
  - "the head is a verb"
  - "the dependent is a noun"
  - "the head is a verb and the dependent is a noun"
  - "the arc goes from left to right"
  - "the arc has length 2"

Arc-factored parsing (1)

Objective: learn a scoring function for edges that assigns the best total score to the most likely analysis

- represent edges with features F
  - "the head is a verb"
  - "the dependent is a noun"
  - "the head is a verb and the dependent is a noun"
  - "the arc goes from left to right"
  - "the arc has length 2"
- assign weights to each feature $\rightarrow$ to each parse tree

Arc-factored parsing (2)

Objective: learn a scoring function for edges that assigns the best total score to the most likely analysis

- for each sentence in the training data, compare
  - the top-scoring analysis A
  - with the gold analysis G

Arc-factored parsing (2)

Objective: learn a scoring function for edges that assigns the best total score to the most likely analysis

- for each sentence in the training data, compare
  - the top-scoring analysis A
  - with the gold analysis G
- update weights to increase the relative likelihood of the gold analysis:
  - increase the weight of features in $F(G) \setminus F(A)$
  - decrease the weight of features in $F(A) \setminus F(G)$

The most well-known example is the Eisner algorithm

- based on dynamic programming (like Viterbi, CKY)
- runs in $\mathcal{O}(n^3)$

Transition-based parsing

Transition-based methods are based on a technique called **shift-reduce parsing**.

Transition-based parsing

Transition-based methods are based on a technique called **shift-reduce parsing**. Generally, the parser

- reads the sentence *once* word by word from left to right $\implies \mathcal{O}(n)$
- builds the *parse* incrementally, without backtracking
- uses two data structures:
  - **buffer**: the list of words yet unread – starts with the whole sentence
  - **stack**: stores words that are not part of the parse yet – starts empty

Transition-based parsing

Transition-based methods are based on a technique called **shift-reduce parsing**. Generally, the parser

- reads the sentence *once* word by word from left to right $\implies \mathcal{O}(n)$
- builds the *parse* incrementally, without backtracking
- uses two data structures:
  - **buffer**: the list of words yet unread – starts with the whole sentence
  - **stack**: stores words that are not part of the parse yet – starts empty
- at each step, chooses one of two actions:
  - **shift**: read the next word from the buffer and push it on the stack
  - **reduce**: pop words from the stack and add the construct that covers them to the parse

Transition-based parsing – cont.

Transition based dependency parsing has many variants; we'll now discuss **arc-standard parsing**

- a shift-reduce parser
- it has two *reduce* actions:

  LeftArc add an arc $w_1 \rightarrow w_2$ and pop w_2
  RightArc add an arc $w_2 \rightarrow w_1$ and pop w_1

  (where w_1 and w_2 are the two topmost words on the stack)
- An ML model is trained to predict the most likely next step for any configuration: this is known as the **Guide**

Transition-based parsing – example

Stack	Buffer	Action

Book me the morning flight

Transition-based parsing – example

Stack	Buffer	Action
root	book, me, the, morning, flight	SHIFT

Book me the morning flight

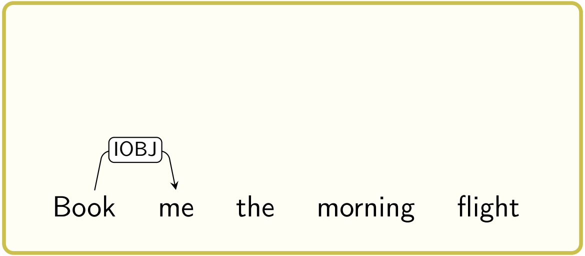
Transition-based parsing – example

Stack	Buffer	Action
root	book, me, the, morning, flight	SHIFT
root, book	me, the, morning, flight	SHIFT

Book me the morning flight

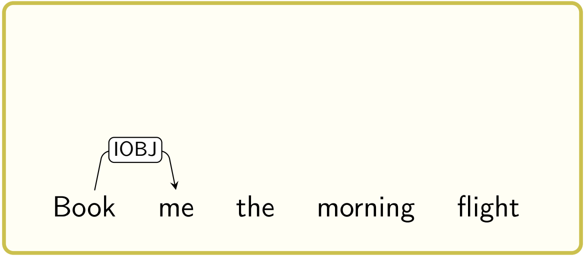
Transition-based parsing – example

Stack	Buffer	Action
root	book, me, the, morning, flight	SHIFT
root, book	me, the, morning, flight	SHIFT
root, book, me	the, morning, flight	RIGHT ARC



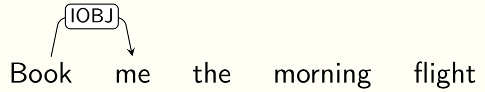
Transition-based parsing – example

Stack	Buffer	Action
root	book, me, the, morning, flight	SHIFT
root, book	me, the, morning, flight	SHIFT
root, book, me	the, morning, flight	RIGHT ARC
root, book	the, morning, flight	SHIFT



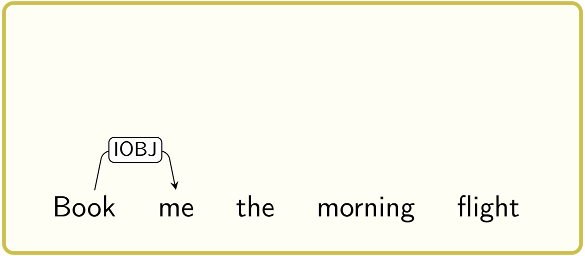
Transition-based parsing – example

Stack	Buffer	Action
root	book, me, the, morning, flight	SHIFT
root, book	me, the, morning, flight	SHIFT
root, book, me	the, morning, flight	RIGHT ARC
root, book	the, morning, flight	SHIFT
root, book, the	morning, flight	SHIFT



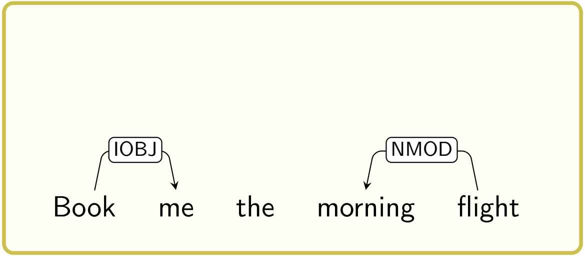
Transition-based parsing – example

Stack	Buffer	Action
root	book, me, the, morning, flight	SHIFT
root, book	me, the, morning, flight	SHIFT
root, book, me	the, morning, flight	RIGHT ARC
root, book	the, morning, flight	SHIFT
root, book, the	morning, flight	SHIFT
root, book, the, morning	flight	SHIFT



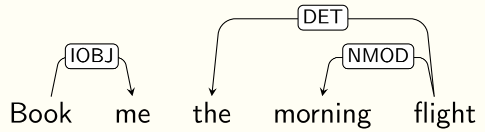
Transition-based parsing – example

Stack	Buffer	Action
root	book, me, the, morning, flight	SHIFT
root, book	me, the, morning, flight	SHIFT
root, book, me	the, morning, flight	RIGHT ARC
root, book	the, morning, flight	SHIFT
root, book, the	morning, flight	SHIFT
root, book, the, morning	flight	SHIFT
root, book, the, morning, flight		LEFT ARC



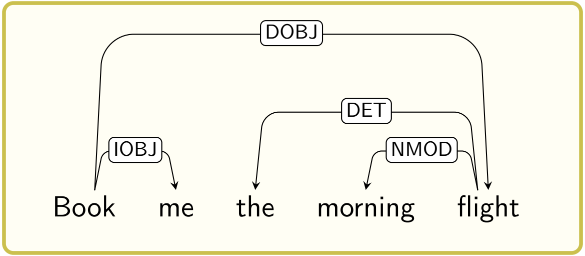
Transition-based parsing – example

Stack	Buffer	Action
root	book, me, the, morning, flight	SHIFT
root, book	me, the, morning, flight	SHIFT
root, book, me	the, morning, flight	RIGHT ARC
root, book	the, morning, flight	SHIFT
root, book, the	morning, flight	SHIFT
root, book, the, morning	flight	SHIFT
root, book, the, morning, flight		LEFT ARC
root, book, the, flight		LEFT ARC



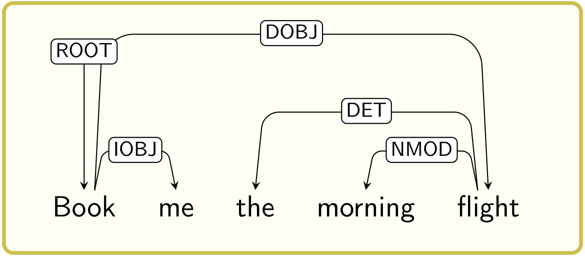
Transition-based parsing – example

Stack	Buffer	Action
root	book, me, the, morning, flight	SHIFT
root, book	me, the, morning, flight	SHIFT
root, book, me	the, morning, flight	RIGHT ARC
root, book	the, morning, flight	SHIFT
root, book, the	morning, flight	SHIFT
root, book, the, morning	flight	SHIFT
root, book, the, morning, flight		LEFT ARC
root, book, the, flight		LEFT ARC
root, book, flight		RIGHT ARC



Transition-based parsing – example

Stack	Buffer	Action
root	book, me, the, morning, flight	SHIFT
root, book	me, the, morning, flight	SHIFT
root, book, me	the, morning, flight	RIGHT ARC
root, book	the, morning, flight	SHIFT
root, book, the	morning, flight	SHIFT
root, book, the, morning	flight	SHIFT
root, book, the, morning, flight		LEFT ARC
root, book, the, flight		LEFT ARC
root, book, flight		RIGHT ARC
root, book		RIGHT ARC



Transition-based parsing – the Guide (1)

A **guide** is a model responsible for selecting the most likely next step in the parsing process

- guides are trained on gold-standard trees
- configurations are represented by features, e.g.
  - word form, POS-tag of words on the stack and/or the last words on the buffer
  - dependency relations of top word on the stack
  - combinations of the above
  - misc: distances, no. of children, ...

Transition-based parsing – the Guide (2)

ML models used to implement guides:

- Preferred models are those that can handle large number of sparse features, e.g.:
  - multinomial logistic regression
  - support vector machines (SVMs)

Transition-based parsing – the Guide (2)

ML models used to implement guides:

- Preferred models are those that can handle large number of sparse features, e.g.:
  - multinomial logistic regression
  - support vector machines (SVMs)
- in recent years: deep learning methods have appeared
  - Multi-layer perceptron (MLP): Chen and Manning (2014)
  - Long Short-term Memory (LSTM): Dyer et al. (2015)

Transition-based parsing – pros and cons

Pros:

- Efficient: time complexity linear in the number of words
- Transparent: guides can be trained with informative features

Cons:

- may not be able to yield the best tree
- risk of error propagation (consider *book* $\xrightarrow{\text{DOBJ}}$ *me* in the sentence *Book me a flight*)

Transition-based parsing – variants

Recall: **arc-standard** parsing is just one way of doing transition-based parsing!
Variants include:

- **arc-eager**
  - shift, reduce, left-arc, right-arc
  - not strictly bottom-up
- **non-projective**
  - shift, swap, left-arc, right-arc
  - allows for non-projective parsing ("crossing lines")
  - quadratic in the worst case, but not in practice

Universal Dependencies

- language-independent dependency relations
- annotations consistent across 60+ languages
- 100+ treebanks publicly available
- Shared tasks in multilingual dpeendency parsing (2017, 2018)
- `http://universaldependencies.org`

Outlook

Other formalisms

Besides PSG and DG, a large number of formal grammars exist. Some of the most interesting formalisms are

- Lexical-Functional Grammar (LFG)
- Combinatory Categorial Grammar (CCG)
- Link Grammar (LG)

All **lexicalized grammars**, which put the brunt of syntax into the lexicon.

Other formalisms

Besides PSG and DG, a large number of formal grammars exist. Some of the most interesting formalisms are

- Lexical-Functional Grammar (LFG)
- Combinatory Categorial Grammar (CCG)
- Link Grammar (LG)

All **lexicalized grammars**, which put the brunt of syntax into the lexicon.

Others not detailed here

- Head-driven Phrase Structure Grammar (HPSG)
- Tree-adjoining Grammar (TAG)
- Constraint Grammar (CG)
- ...

Other formalisms

Besides PSG and DG, a large number of formal grammars exist. Some of the most interesting formalisms are

- Lexical-Functional Grammar (LFG)
- Combinatory Categorial Grammar (CCG)
- Link Grammar (LG)

All **lexicalized grammars**, which put the brunt of syntax into the lexicon.

Others not detailed here

- Head-driven Phrase Structure Grammar (HPSG)
- Tree-adjoining Grammar (TAG)
- Constraint Grammar (CG)
- ...

There isn't really just one "right tool" for the job!

Other formalisms: LFG

Lexical-functional grammar provides a multi-dimensional view on language.

- *c-structure*: the constituent structure
- *f-structure*: the functional (predicate) structure
- Optional structures: semantic, morphologic, etc.
- Relations between words in the lexicon (e.g. active–passive)

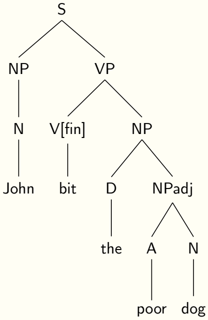
Other formalisms: LFG

Lexical-functional grammar provides a multi-dimensional view on language.

- *c-structure*: the constituent structure
- *f-structure*: the functional (predicate) structure
- Optional structures: semantic, morphologic, etc.
- Relations between words in the lexicon (e.g. active–passive)

Example:

pred	'bite<[7:John], [2:dog]>'
tns-adp	[tense past, prog -, perf -, mood indicative]
subj	[pred 'John', ntype [nsem [proper [proper-type name, name-type first_name]], nsyn proper], pers 3, num sg, human +, gend-sem male, case nom]
obj	...
vtype main,	passive -, clause-type decl



Other formalisms: (C)CG

Categorial grammars are based on the idea of *compositionality*.

- Lexicon: each word has a *lexical category*
  - Basic: N, NP, S
  - Function: `the: NP/N` means: "*the*" is a function that takes a N to produce an NP
  - Argument on left: $S\backslash NP$ ($\equiv$ VP), right: N/N ($\equiv$ Adj)
- Inference rules: language-agnostic
- The most popular formalism is Combinatory Categorial Grammar (CCG), which is based on *combinatory logic*.

Other formalisms: (C)CG

Categorial grammars are based on the idea of *compositionality*.

- Lexicon: each word has a *lexical category*
  - Basic: N, NP, S
  - Function: `the: NP/N` means: "*the*" is a function that takes a N to produce an NP
  - Argument on left: $S\backslash NP$ ($\equiv$ VP), right: N/N ($\equiv$ Adj)
- Inference rules: language-agnostic
- The most popular formalism is Combinatory Categorial Grammar (CCG), which is based on *combinatory logic*.

Example:

$$\dfrac{\dfrac{\dfrac{John}{NP} \qquad \dfrac{bit}{(S\backslash NP)/NP}}{S\backslash NP} \qquad \dfrac{\dfrac{the}{NP/N} \qquad \dfrac{\dfrac{poor}{N/N} \qquad \dfrac{dog}{N}}{N}}{NP}}{S}$$

John	NP
bit	(S\NP)/NP
the	NP/N
poor	N/N
dog	N

Other formalisms: LG

Link Grammar is a theory of syntax and (optionally) morphology.

- Like DG, it builds labelled relations between words
- Unlike DG, it is highly lexicalized:
  - the main resource is the *dictionary*, which lists *words*
  - along with their *linking requirements*
- Each word is like a jigsaw/domino piece, and parsing is akin to assembling a puzzle
- Word order matters: extensions for free word order languages

Other formalisms: LG

Link Grammar is a theory of syntax and (optionally) morphology.

- Like DG, it builds labelled relations between words
- Unlike DG, it is highly lexicalized:
  - the main resource is the *dictionary*, which lists *words*
  - along with their *linking requirements*
- Each word is like a jigsaw/domino piece, and parsing is akin to assembling a puzzle
- Word order matters: extensions for free word order languages

Example:

```
    +----------------Xp---------------+
    |            +-------Os-------+   |
    |            |    +-----Ds----+   |
    +---Wd--+-Ss-+    |     +--A--+   |
    |       |    |    |     |     |   |
LEFT-WALL John bit.v the poor.a dog.n .
```

John	S+ & O-
bit	S- & O+
the	D+
poor	A+
dog	D- & S+ & O-
.	Xp-

Resources

- (P)CFG and DG
  - For English:
    - Online version of the Stanford Parser
    - The Stanford CoreNLP library
  - For Hungarian:
    - `e-magyar.hu`
    - The hunlp-GATE library
- LFG
  - XLE-Web, an online LFG parser
- CCG
  - A primer to CCG (Steedman and Baldridge, 2011)
  - The OpenCCG parser
- Link Grammar
  - Webpage with online parser: http://www.link.cs.cmu.edu/link/
  - Original report (Sleator and Temperley, 1991)

Appendix: bibliography

Chen, Danqi and Christopher D Manning (2014). "A Fast and Accurate Dependency Parser using Neural Networks". In: *EMNLP*, pp. 740–750.

Chomsky, Noam (1956). "Three models for the description of language". In: *IRE Transactions on Information Theory* 2, pp. 113–124.

Dyer, Chris, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith (2015). "Transition-Based Dependency Parsing with Stack Long Short-Term Memory". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 334–343. arXiv: `1505.08075 [cs.CL]`. url: `http://www.aclweb.org/anthology/P15-1033`.