References:

Jurafsky and Martin, "Speech and Language Processing".


Hopcroft and Ullman, "Introduction to Automata Theory, Languages and Computation".

Alon Lavie

January 20, 2004

Alon Lavie

Parsing Overview

11-722: Grammar Formalisms
Formal Languages

A mathematical abstraction of real languages: Natural Languages and Computer Languages

Amathematical abstraction of real languages

Given a word \( m \) and a language \( L \), is \( m \in L \)?

Fundamental problem - word membership:

Classes of languages using formal mathematical proofs

Formal Language Theory: The study of properties of the various types and

Languages can be defined declaratively, descriptively or computationally

sentences in a natural language

A language is no more than a set of items called words (the equivalent of

Computer Languages

Formal Languages
Basic Definitions

- Alphabet: A finite (non-empty) set of atomic symbols
- Word: A string of letters from a given alphabet
- Language: A set (finite or infinite) of words constructed from a given alphabet

Important Notation:

\[ \{ T \not= m \mid \exists \exists \not= m \} = T \]

Set Theory and Operations apply to formal languages:

- Union, Intersection, Complementation, Membership

Examples:

- \( \emptyset \)
- \( \{ \} \)

Def: a language \( L \) is a set (finite or infinite) of words of length \( n \) on a given alphabet

- \( 0 \) denotes the empty word
- \( m \) denotes the length of word
- \( u \) denotes a string of letters from a given alphabet
- \( q, a, b, c \) are letters
- \( \sigma \) is a symbol
- The alphabet is a finite (non-empty) set of atomic symbols

Basic Definitions
Language Classes

- Sets of formal languages that can be defined using a particular descriptive definition or abstraction of a computational framework

Examples:
- The set of languages that can be described by Regular Expressions
- The set of languages for which we can construct a Finite State Automaton
- The set of languages that can be defined using a Context-free Grammar

Knowing the class to which a language belongs will allow us to develop efficient algorithms for processing the language or deciding membership in the language.
Deterministic FSA

Formal Definition of a DFSA:
\[ A = (Q, \Sigma, \sigma, q_0, F) \]
where:
- \( Q \) is a finite set of states
- \( \Sigma \) is a finite alphabet
- \( q_0 \in Q \) is an initial (start) state
- \( \sigma : Q \times \Sigma \rightarrow Q \) is the complete transition function
- \( F \subseteq Q \) is a set of final states

The language accepted by a DFSA \( A \) is defined to be:
\[ L(A) = \{ w \in \Sigma^* : (A, q_0, q, w) \in F \} \]

Examples of Regular Languages:

\( \{ \epsilon \} = \{ \epsilon \} \]
\( \epsilon \Sigma^* = \Sigma^* \]

For all DFSA \( A \) such that the language \( L(A) \) is called regular if there exists some \( \Sigma^* \) that is a DFSA.

Definition of Regular Language:
A language \( L \) is called regular if there is a DFSA that accepts it.

Formal definition of \( L(A) \):
\[ L(A) = \{ w \in \Sigma^* : (A, q_0, q, w) \in F \} \]

Grammar Formalisms
Context-Free Grammars

A descriptive generative formalism for specifying the set of words in a language using production rules.

Formal Definition:

A context-free grammar $G = (V, T, P, S)$:

- $V$ is a finite set of variables
- $T$ is a finite set of terminals (similar to $\Sigma$ for FSAs)
- $P$ is a set of context-free production rules, each of the form $A \Rightarrow \alpha \beta$
- $S$ is a start non-terminal

Notations:

- We denote single variables or terminals by $X, Y, Z, \ldots$
- We denote strings over $\Lambda \in L$ by $\alpha, \beta, \gamma, \ldots$
- We denote strings over $L \subseteq \Lambda$ by $w, x, y, \ldots$
- We denote elements of $\Lambda$ by $a, b, c, \ldots$
- We denote elements of $\Lambda \setminus T$ by $q, r, s, \ldots$
- We denote single variables or terminals by $X, Y, Z, \ldots$

Formal Definition:

A context-free grammar $G = (V, T, P, S)$ with:

$\langle \Lambda \not\in S \rangle$ (where $\Lambda \not\in S$)

$L \subseteq N$ is a finite set of terminal symbols (similar to $\Sigma$ for FSAs)

A descriptive generative formalism for specifying the set of words in a language using production rules.
Context-Free Grammars

Example:

\[ L = \{ anb^n j^n | n \geq 1 \} \]

In this case the language \( L(G) \) could be specified in a succinct mathematical form - often this is difficult or not possible.

\[ \begin{align*}
G &: \rightarrow aSb \\
S &\rightarrow ab \\
\end{align*} \]

Example:

\[ \{ I \geq u | uqvd \} = T \]
CFG Derivations

Derivations describe the process of using the context-free rules to derive a string of terminal symbols. A derivation is a sequence of expansion steps in which at each step, the rightmost non-terminal is expanded. A rightmost derivation is one in which at each step, the rightmost non-terminal is expanded. A leftmost derivation is one in which at each step, the leftmost non-terminal is expanded.

### Definition

Let \( G \) be a context-free grammar. A sequence of expansions \( \gamma \) is a derivation if \( \gamma \) induces a sequence of expansions \( \gamma_1 \) such that

\[
\gamma_1 \rightarrow \gamma_2 \rightarrow \cdots \rightarrow \gamma_i \rightarrow \cdots \text{ is a rule in } \mathcal{P}
\]

and \( \mathcal{A} \cap \mathcal{V} = \emptyset \) and \( \gamma_1 \) directly derives \( \gamma \). The sequence \( \gamma_1 \) is denoted by \( \gamma \). The set of all derivations is denoted by \( \mathcal{D} \).

### Derivations

Similarly for a leftmost derivation.
Formal Definition: The language of a CFG $G$ is defined as:

$$L(G) = \{ w \in \Sigma^* \mid T \in \mathcal{G} \text{ and } w \text{ is generated by } T \}$$

A language $L$ is context-free if there exists a grammar $G$ such that $L = L(G)$. The set of all such languages is called the set of context-free languages (CFLs).

Two grammars $G_1$ and $G_2$ are called equivalent if:

$$L(G_1) = L(G_2)$$

Formal Definition: The language of a CFG $G$ is defined as:

$$L(G) = \{ m \in \Sigma^* \mid S \in T \}$$

Context Free Languages (CFLs)
There exist CFLs that are inherently ambiguous.

Two or more different parse trees.

A grammar \( G \) is called ambiguous if there exists a word \( w \in \Sigma^* \) that has

A parse tree constitutes a proof that a given input string is in \( L(G) \).

The tree does not reflect the structure of the input string. What rules were used to
derive the various subtrees of the input.

The tree does not represent the derivation order of the non-terminals.

The leaves (yield) of the tree correspond to a terminal string in \( L(G) \).

A Parse Tree is a graphical representation of a derivation.
Pushdown Automata

An extension of a FSA that is powerful enough to accept CFLs

The FSA is augmented with a memory storing device in the form of a stack

Formal Definition: a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

- $Q$ is a finite set of states
- $\Sigma$ is an input alphabet
- $\Gamma$ is a finite set of stack symbols
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

It can also:
- replace the top element of the stack
- push an element onto the stack
- pop an element from the stack

Note that a PDA is non-deterministic: it can make ε-moves on the input

Grammar Formalisms
there exist a variety of algorithms for parsing CFLs and their variants

possible parse trees (for $m$

Given a grammar $G$ and a word $w$, if $w \in T(G)$, find a parse tree (or all)

the parsing problem

the recognition problem

Recongition and Parsing of CFLs
Languages defined by these types of grammars require more powerful computation devices in order to recognize the languages.

- **Context-Free Grammars (CFGs)** are called context-free because the form of the grammar rules allows them to be used in a derivation regardless of the context in which a non-terminal appears.

- **Unrestricted Grammars** are grammars where the rules are unrestricted in form - they contain one or more grammar symbols, and in order to be applied in a derivation, the entire left-hand side of the rule must match a substring of the current derived string.

- **Context-Sensitive Grammars** are grammars where the rules have the form where contains one or more grammar symbols, and contains zero or more grammar symbols, with the restriction that contains zero or more grammar symbols, and contains one or more terminal symbols.

Context-Sensitive Grammars are grammars where the rules have the form where contains zero or more terminal symbols.

- **Context-Free Grammars (CFGs)** are called context-free because the form of the grammar rules allows them to be used in a derivation regardless of the context in which a non-terminal appears.
Chomsky was one of the pioneers in identifying the correspondence between
the different types of grammars and the formal computational models that are
required to recognize them.
Parsing Algorithms

Cleardistinctioninallgrammarformalisms:

–TheGrammar: a declaration (generally generative) finite description of what structures in the language are grammatical

–TheLanguage: the (possibly infinite) set of all strings that are derivable according to the grammar

–TheParser: an algorithm that given an input decides membership in the language

–TheGrammar Formalisms

Recognition vs. Parsing:

–Recognition - deciding the membership in the language:
For a given grammar $G$, an algorithm $\mathcal{A}(G)$, an algorithm that given an input $w \in \Sigma^*$ decides: is $w \in L(G)$?

–Parsing - Recognition + producing a parse tree for $w$.

Is parsing more “difficult” than recognition (time complexity)?

Ambiguity - a parse for $w$ or all parses for $w$?

–Identifying the “correct” parse

–Ambiguity representation - an input may have exponentially many parses

In many grammar formalisms CFGs are basis for describing the constituent structure of NL sentences

In many grammar formalisms CFGs are basis for describing the constituent structure of English, and determining it's structure according to the grammar

–TheParser: an algorithm that for a given input, decides membership in the language

–TheLanguage: the (possibly infinite) set of all strings that are derivable according to the grammar

–TheGrammar: a declarative (generally generative) finite description of what structures in the language are grammatical

Clear distinction in all grammar formalisms:
### CFL Parsing Algorithms

**Parsing General CFLs vs. Limited Forms**

- **Efficiency:**
  - Deterministic (LR) languages can be parsed in linear time
  - A number of parsing algorithms for general CFLs require \( O(n^3) \) time
  - Asymptotically best parsing algorithm for general CFLs requires \( O(n^{2.376}) \), but is not practical

- **Utility - why parse general grammars and not just CNF?**
  - Grammar intended to reflect actual structure of language
  - Conversion to CNF completely destroys the parse structure

- Parsing Unification-based grammars is quite a different story...
Top-Down vs. Bottom-Up Parsing

Top-Down Parsing:
- Construct the parse tree starting from the root ("S") of the grammar
- Expand a non-terminal using one selected grammar rule
- Match terminal nodes with the input
- Advantages: only constructs partial trees that can be derived from the root "S"
- Problems: efficiency, handling ambiguity, left-recursion

Bottom-Up Parsing:
- Construct a parse starting from the input symbols
- Build constituents from sub-constituents
- When all constituents on the RHS of a rule are matched, create a constituent for the LHS of the rule
- Advantages: only creates constituents that are consistent with the input
- Problems: efficiency, handling ambiguity

16-722 Grammar Formalisms
Various CFG parsing algorithms are a hybrid of Top-Down and Bottom-Up. They attempt to combine the advantages of both.

Ambiguity Packing allows efficient storage of ambiguous analyses, so that they can be shared or memorized.

A Chart allows storing partial analyses, so that they can be shared or memorized.
The Earley Parsing Algorithm

General Principles:
- A clever hybrid Bottom-Up and Top-Down approach
  - Bottom-Up parsing completely guided by Top-Down predictions
  - Maintains sets of ‘dotted’ grammar rules that:
    - Reflect what the parser has ‘seen’ so far
    - Explicitly predict the rules and constituents that will combine into a complete parse
  
- Time Complexity $O(n^3)$, but better on particular sub-classes
- First efficient parsing algorithm for general context-free grammars.
The Earley Parsing Method

Main Data Structure: The "state" (or "item")

We denote the set of states for position \( i \) by \( S_i \)

\[
\begin{align*}
S_0 &= \{ \text{starting from 0} \} \\
A \leftarrow \{ X_1^{\epsilon} \cdots X_m^{\epsilon} \} \\
A \text{ state is a "dotted" rule and starting position:} \\
\text{Main Data Structure: The "state" (or "item")}
\end{align*}
\]
The Earley Parsing Algorithm

Three Main Operations:

Predictor:
If state $[C \cdot \cdot \cdot X_1 \cdot \cdot \cdot X_m \cdot \cdot \cdot j] \in S_i$ then for every rule of the form $C \cdot \cdot \cdot a \cdot \cdot \cdot X \cdot \cdot \cdot l$,
add to $S_i$ the state $[C \cdot \cdot \cdot a \cdot \cdot \cdot X \cdot \cdot \cdot l]$.

Completer:
If state $[B \cdot \cdot \cdot X_1 \cdot \cdot \cdot X_k \cdot \cdot \cdot l] \in S_j$ and the next input word is $x_i+1 = a$,
then add to $S$ the state $[B \cdot \cdot \cdot a \cdot \cdot \cdot X \cdot \cdot \cdot k \cdot \cdot \cdot l]$.

Scanner:
If state $[C \cdot \cdot \cdot a \cdot \cdot \cdot X \cdot \cdot \cdot l] \in S_j$ and the next input word is $x_i+1 = a$,
then add to $S$ the state $[C \cdot \cdot \cdot a \cdot \cdot \cdot X \cdot \cdot \cdot l]$. 

The Earley Parsing Algorithm
The Earley Recognition Algorithm

- Simplified version with no lookaheads and for grammars without epsilon-rules
- Assumes input is string of grammar terminal symbols
- We extend the grammar with a new rule $S' \rightarrow S \; \$\$
- The algorithm sequentially constructs the sets $S_i$ for $0 \leq i \leq n + 1$
- We initialize the set $S_0$ with $S_0 = \{[S' \rightarrow \bullet S \; \$, 0]\}$
The Earley Recognition Algorithm

The Main Algorithm: parsing input

\[ S^0 = \varnothing \]

1. For 0 ≤ i ≤ n do:

\[ \{[0, S \cdot \leftarrow S]\} = \{ \[0, S \cdot \leftarrow S\]\} \]

2. For 0 ≤ i ≤ n do:

\[ S_{i+1} = \{ \} \]

3. If \( S_{i+1} = \{ \} \) then Accept the input

4. If \( i = n \) and \( S_n + 1 = \{\{0, S \cdot \leftarrow S\}\} \) then Accept the input

The Earley Recognition Algorithm
Parsing with an Earley Parser

Weneedtopreservereferencesfromtheback-pointerswhentwocompletionrules

Each item must be extended to have the form

Every rule completion must be kept.

When we complete a rule

The constituents and the pointers can be created during Scanner and

Where the $pt_i$ are pointers to the already found RHS sub-constituents

At the end - reconstruct parse from the back-pointers
Efficient Representation of Ambiguities

- The back-pointers:
  - Unpacking - Producing one or more of the packed parses needs by following exponentially many
  - Allows to efficiently represent a very large number of ambiguities (even
  - Local Ambiguity Packing: create a single item in the chart for $A(p_j; p_k)$, with pointers to the various possible derivations.
  - Multiple items in the constituent chart of the form $X \leftarrow V$
    - What do local ambiguities look like with Early Parsing?

| A | Terminal
---|---
| A | non-terminal
| A | Local Ambiguity - multiple ways to derive the same sub-string from a

Efficient Representation of Ambiguities
Time Complexity of Earley Algorithm

An algorithm iterates for each word of input (i.e., \( n \) iterations)

- How many items can be created and processed in \( S_i \)?
  - Each item in \( S_i \) has the form \([A^*] A \cdots A\]
  - How many items can be created and processed in \( S_i \)?
  - Thus \( n \) items

- Time required for each iteration \((S_i)O\) is thus
  - Time for each processed item \((u)O\) to \( \{A^*\} A \cdots A \)

The Complete operation on an item adds items of form

\([B^*] A \cdots A\]

Thus \( u \) items

- Thus items
- Each item in \( S_i \) has the form \([\ell^*] A \cdots A\]

Algorithm iterates for each word of input (i.e., \( n \) iterations)

Time Complexity of Earley Algorithm
Special Cases:

Completer is the operation that may require \( O(i^2) \) time in iteration \( i \).

- For unambiguous grammars, Earley shows that the completer operation will require at most \( O(i) \) time.

Thus time complexity for unambiguous grammars is \( O(n^2) \).

For bounded-state grammars and include even some ambiguous grammars.

For bounded-state grammars, the number of items in each set \( S_i \) is bounded by a constant.

- Time Complexity of Earley Algorithm
The Grammar:

(1) S
(2) NP
(3) NP!
(4) NP!
(5) NP!
(6) VP
(7) VP!
(8) vNP

The original input: "x = The large can can hold the water"

POS assigned input: "x = art adj n aux in art n "

Parser input: "x = art adj n aux in art n $"

The Grammar:

\[
\begin{align*}
\text{dN} & \leftarrow \text{dA} \quad (6) \\
\text{dA} & \leftarrow \text{dA} \quad (5) \\
n & \leftarrow \text{dN} \quad (4) \\
\text{art} & \leftarrow \text{dN} \quad (3) \\
\text{art adj} & \leftarrow \text{dN} \quad (2) \\
\text{dA dN} & \leftarrow S \quad (1)
\end{align*}
\]
The input: \( x, \text{art adj n aux v art n } \$ \)
The input: \texttt{art adj aux v art n "$\)
The input: "a red art and aux and art in $\$"

Earley Parsing - Example
Earley Parsing - Example

The input: “x = art adj n aux v art n $”

$S_2$: [NP $\rightarrow$ art$_1$ adj$_2$ $\bullet$ n $\bullet$, 0]

$S_3$: [NP$_4$ $\rightarrow$ art$_1$ adj$_2$ n$_3$ $\bullet$, 0]
The input: "The input: art adj in aux v art in S"
The input: "art adj n aux v art n."

Early Parsing - Example
Earley Parsing - Example

The input: art adj\textsubscript{1} aux art\textsubscript{2} in $S$.
Earley Parsing - Example

The input: "x, art adj in aux and art $u$"

The formalisms:

\[(\mathcal{L}_7' \cup \mathcal{L}_5') \cup \mathcal{L}_7' \cup \mathcal{L}_5) \cup \mathcal{L}_7' \cup \mathcal{L}_5 \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_7' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]

\[\mathcal{S}_6' \cup \mathcal{S}_5' \cup \mathcal{S}_7' \cup \mathcal{S}_5' \]
```
[0, ' • $ S \rightarrow \epsilon S] : ^8 S

[0, ' $ • S \rightarrow \epsilon S]

(\ell x) \beta p \wedge \bowtie p n \rightarrow S \ \forall I
(\ell x) \beta p \wedge \bowtie p n \rightarrow d \Lambda \ \forall I
(\ell x) \beta p \wedge \bowtie p n \rightarrow d \Lambda \ \forall I

[0, ' • \beta p \wedge \bowtie p n \rightarrow \epsilon S]
[0, ' • \beta p \wedge \bowtie p n \rightarrow \epsilon \beta p \wedge \bowtie p n]
[\forall ' • \beta p \wedge \bowtie p n \rightarrow \epsilon \beta p \wedge \bowtie p n]
[\forall ' • \beta p \wedge \bowtie p n \rightarrow \epsilon \beta p \wedge \bowtie p n]

The input: art adj n aux \land art n $```

Earley Parsing - Example
Augmenting CFGs with Features

Certain linguistic constraints are not naturally described via CFGs only. CF rules are not practical for describing a large set of such feature constraints using only CF rules. Much more natural to describe via a single feature-augmented CFG rule:

\[
\begin{align*}
\text{NP} & \rightarrow \text{ART} \ N \\
\text{NP} & \rightarrow \text{ART} \rightarrow \text{ART} \rightarrow \text{ART} \rightarrow \text{ART} \rightarrow \text{ART} \\
\text{NP} & \rightarrow \text{ART} \rightarrow \text{ART} \rightarrow \text{ART} \rightarrow \text{ART} \rightarrow \text{ART}
\end{align*}
\]

Possible to describe using refined CF rules:

```
Example: Number Agreement between constituents - "a boy's"
```

Certain linguistic constraints are not naturally described via CFGs.
Feature Structures

1. Complex Feature Structures: Feature values can themselves be features.
   - Verb form features and sub-categorizations
   - Number, Gender, and Person agreement

2. Some basic features for English:

3. Linguistic constraints express rules about how the feature-structure of a constituent is formed from its sub-constituents.

4. Features can be shared between constituents.

5. Assigned values can be viewed as structures (collections) of features that have
Unification of feature structures

The Unification operation allows easy expression of grammatical relationships among constituent feature structures.

The Most General Unifier is the minimal FS $F$ that both $F_1$ and $F_2$ subsume.

The Unification operation allows easy expression of grammatical relationships among constituent feature structures.

Two FSs $F_1$ and $F_2$ unify if there exists a FS $F$ that both $F_1$ and $F_2$ subsume.

$F_1$ subsumes $F_2$ if every feature-value pair in $F_1$ is also in $F_2$.

Key concept: subsumption relationship between two FSs.

For a language via a set of relationships between feature structures of unification grammars (such as HPSG) establish a complete linguistic theory.
Unification of Feature Structures

Example:

\[ F_1 = ((\text{cat} \times \text{v}) \quad F_2 = ((\text{cat} \times \text{v}) \quad F_3 = ((\text{cat} \times \text{v}) \quad (\text{root} \times \text{cry}) \quad (\text{vform} \times \text{pres})) \quad \) \]

\[ F_1 \text{ subsumes } F_2 \]

\[ F_1 \text{ and } F_2 \text{ do not unify:} \]

\[ F_1 = ((\text{cat} \times \text{v}) \quad F_2 = ((\text{cat} \times \text{v}) \quad F_3 = ((\text{cat} \times \text{v}) \quad (\text{root} \times \text{cry}) \quad (\text{vform} \times \text{pres})) \quad (\text{agr} \times \text{3s}) \quad (\text{agr} \times \text{3p})) \]

\[ F_3 \text{ is MCG of } F_1 \text{ and } F_2 \]

\[ F_1 \text{ subsumes } F_2 \]

Unification of Feature Structures
Unification-based Grammars

Grammars can be completely specified using unification.

Example:

\[ ((x_1 = x) \land (x_1 \neq x)) \]

\[ s \rightarrow np \land np \]

Example:

Non-terminal of a CFG rule

If a feature (such as cat) is always specified, it can be associated with the non-terminal.

Example:

\[ ((x_1 = x) \land (x_1 \neq x)) \]

\[ x_1 \land np \land np \]

Example:

Grammar rules can be completely specified using unification.

Unification-based Grammars
Unification-based Grammars

Example:

The grammar rule:

\[ \text{NP} \rightarrow \text{ART} \text{N} \]

\[ (((x_1 \text{agr}) = (x_2 \text{agr})) \]

\[ ((x_0 \text{spec}) = (x_1 \text{spec})) \]

\[ (x_0 = x_2) \]

The Feature Structures:

\[ \text{ART} : ( ( \text{agr}^3, \text{agr}^3 ) \]

\[ \text{NP} : ( ( \text{agr}^3 ) \]

\[ \text{N} : ( ( \text{agr}^3 ) \]

\[ \text{root}^* \text{the} \]

\[ \text{spec}^* \text{def} \]

\[ \text{root}^* \text{boy} \]

\[ \text{spec}^* \text{def} \]

\[ \text{root}^* \text{the} \]

\[ \text{spec}^* \text{def} \]

Example:

Unification-based Grammars
CFG Parsing with Feature Unification

Backbone

CFG is augmented with a functional description that describes unification constraints between grammar constituents. This is called Interleaved Unification. Other approaches are also possible. The FS corresponding to the "root" of the grammar is constructed compositionally during parsing. This is called Interleaved Unification. The FS corresponding to the "root" of the grammar is constructed compositionally during parsing. Backbone CFG is augmented with a functional description that describes...
CFG is augmented with unification equations.

During parse time - the parser maintains an F&S associated with each constituent in the chart.

Whenever COMPLETE applies (for rule i) - the unification operations associated with rule i are applied to the given F&S of the RHS constituents.

If unification succeeds, the F&S associated with the LHS constituents is returned and attached to the new constituent created for the LHS of the rule.

If the unification function fails - the rule completion "fails" - LHS constituent is not created.

Unification Augmented Earley Parsing
In some pure unification grammars, subsumption can replace ambiguity

- Unification can interfere with efficient ambiguity packing
- In unification-augmented CFG parsing with interleaved unification: Pure unification grammars cannot always be parsed efficiently
- Unification creates non-local chains of dependencies
- Complex interaction between ambiguity detection and packing and

Strategies other than interleaved unification are possible:

- Compute packed c-structure first, then solve unification constraints
- Multi-pass strategies for computing c-structure and f-structure can
- Compute packed c-structure first, then solve unification constraints

- Unification creates non-local chains of dependencies

- Unification can interfere with efficient ambiguity packing
- f-structures must also be efficiently represented and packed
- Parsing algorithms can be optimized to achieve maximal ambiguity

- Unification-augmented CFG parsing with interleaved unification:
- Parse f-structures first, then solve unification constraints
- f-structures must also be efficiently represented and packed
- Compute packed c-structure first, then solve unification constraints

- Unification can interfere with efficient ambiguity packing
- Unification-augmented CFG parsing with interleaved unification:
- Pure unification grammars cannot always be parsed efficiently
- Unification creates non-local chains of dependencies
- Complex interaction between ambiguity detection and packing and

Ambiguity Packing and Unification Grammars