**Natural Language Processing**

**- Nguyễn Trung Hiếu –**

Đại học Sư phạm Kỹ thuật Thành phố Hồ Chí Minh

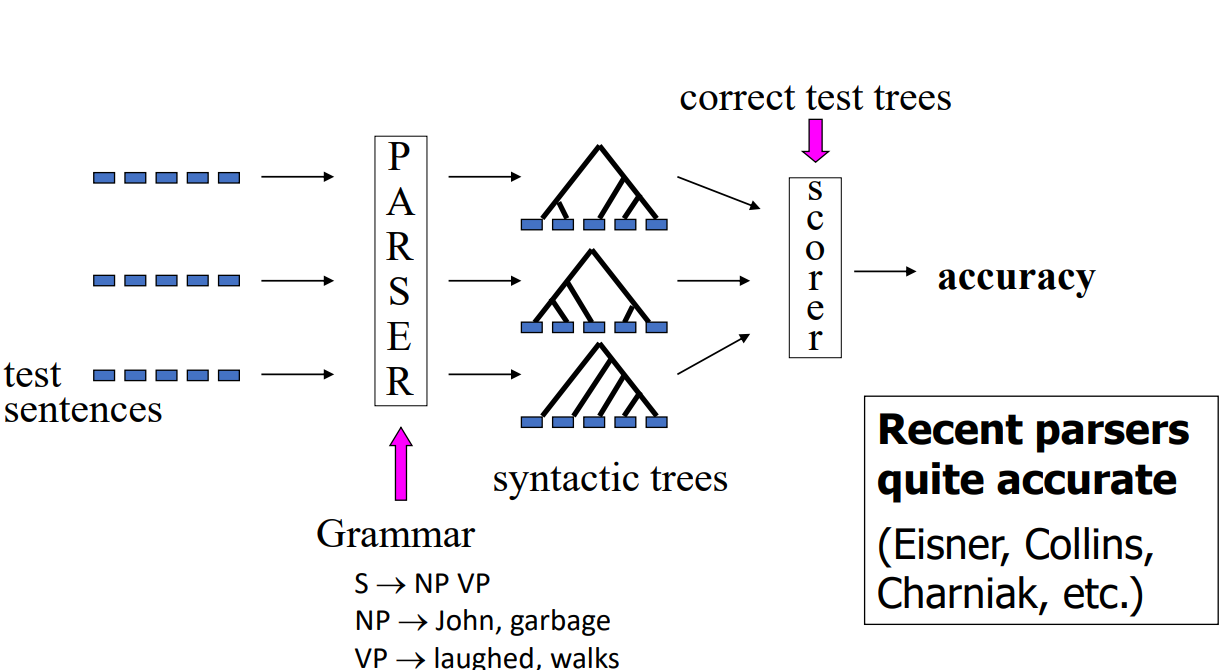
Khoa CNTT

Email: hieuanthonydisward@gmail.com

**Parsing**

**1. Parsing**

**Vấn đề parsing**



**Parsing problem (Vấn đề phân tích cú pháp)**

* **Phân tích cú pháp (Parsing)** là quá trình phân tích cấu trúc cú pháp của một câu theo một **ngữ pháp hình thức (formal grammar)**.
* Mục tiêu là tạo ra **cây cú pháp (syntactic tree)** thể hiện mối quan hệ giữa các từ trong câu.

**2. Parser (Bộ phân tích cú pháp)**

* **Parser** là thuật toán hoặc chương trình để phân tích cú pháp của câu dựa trên một tập hợp quy tắc ngữ pháp.
* Một số **parser nổi tiếng**:
  + **Eisner’s parser**
  + **Collins' parser**
  + **Charniak’s parser**
  + Các parser này được đánh giá là khá chính xác trong việc tạo ra cây cú pháp.

**3. Grammar (Ngữ pháp)**

* Ngữ pháp được sử dụng trong parsing thường là **ngữ pháp phi ngữ cảnh (Context-Free Grammar - CFG)**.
* Ví dụ về quy tắc ngữ pháp CFG trong sơ đồ:
  + **S → NP VP** (Một câu gồm một cụm danh từ NP và một cụm động từ VP)
  + **NP → John, garbage** (Cụm danh từ có thể là "John" hoặc "garbage")
  + **VP → laughed, walks** (Cụm động từ có thể là "laughed" hoặc "walks")

**4. Scorer (Bộ đánh giá)**

* Dùng để đo **độ chính xác (accuracy)** của parser bằng cách so sánh **cây cú pháp dự đoán** với **cây cú pháp đúng (correct test trees)** của tập dữ liệu kiểm tra.
* Điểm số được tính dựa trên mức độ khớp giữa cây cú pháp của parser và cây cú pháp tham chiếu.

**5. Test sentences (Câu kiểm tra)**

* Parser được đánh giá bằng cách kiểm tra nó trên một tập hợp **câu đầu vào (test sentences)**.
* Mục tiêu là xem parser có thể tạo ra **cây cú pháp đúng** cho những câu này không.

**6. Accuracy (Độ chính xác)**

* Parser hiện đại (như của Eisner, Collins, Charniak) có độ chính xác cao.
* Độ chính xác được đo bằng cách so sánh các cây cú pháp đầu ra với các cây cú pháp chuẩn trong tập dữ liệu kiểm tra.

**Tóm tắt**

* **Parsing** là quá trình phân tích cú pháp của một câu theo ngữ pháp hình thức.
* **Parser** là công cụ thực hiện quá trình này, và nó cần được **đánh giá (scorer)** bằng cách so sánh với cây cú pháp đúng.
* **Độ chính xác của parser** hiện nay khá cao nhờ các mô hình tiên tiến.
* **Ngữ pháp CFG** thường được sử dụng trong phân tích cú pháp, với các quy tắc như **S → NP VP**.

**Cấu trúc Cú pháp và Cây Phân tích (Parse Tree)**

**1. Cấu trúc cú pháp là gì?**

Cấu trúc cú pháp (**Syntactic Structure**) mô tả cách các từ trong một câu được tổ chức theo các quy tắc ngữ pháp để tạo thành các thành phần lớn hơn như cụm danh từ (**NP** - Noun Phrase) hay cụm động từ (**VP** - Verb Phrase).

**2. Cây phân tích cú pháp (Parse Tree) là gì?**

Cây phân tích cú pháp (**Parse Tree**) là một biểu đồ thể hiện cấu trúc cú pháp của một câu theo một **ngữ pháp hình thức (formal grammar)**. Nó cho thấy các quan hệ giữa các từ và cách nhóm chúng lại theo từng cấp bậc ngữ pháp.

**Phân tích cây cú pháp của câu:**

**"The plan to swallow Wanda has been thrilling Otto."**

**3. Cấu trúc của cây phân tích cú pháp**

Cây phân tích cú pháp của câu này có thể được biểu diễn như sau:

A line drawing of a bird

Description automatically generated with medium confidence

**4. Giải thích từng thành phần**

**(1) Cấu trúc tổng thể**

* **S (Sentence - Câu chính)**: Là toàn bộ câu, bao gồm một **NP (Cụm danh từ)** và một **VP (Cụm động từ)**.
* **NP (Cụm danh từ)**: "The plan to swallow Wanda"
* **VP (Cụm động từ)**: "has been thrilling Otto"

**(2) Phân tích cụ thể từng phần**

**Cụm danh từ (NP - Noun Phrase)**

* **Det (Mạo từ)**: "The"
* **N (Danh từ chính)**: "plan"
* **to VP (Cụm động từ vô hướng - infinitive phrase)**: "to swallow Wanda"
  + **V (Động từ)**: "swallow"
  + **NP (Cụm danh từ)**: "Wanda"

**Cụm động từ (VP - Verb Phrase)**

* **V (Động từ chính)**: "has"
* **V (Phụ động từ)**: "been"
* **V (Động từ chính của câu)**: "thrilling"
* **NP (Cụm danh từ - Object của động từ)**: "Otto"

**5. Ý nghĩa của cây phân tích cú pháp**

* **Xác định quan hệ cú pháp** giữa các từ trong câu.
* **Giúp máy tính hiểu cú pháp** khi xử lý ngôn ngữ tự nhiên (NLP).
* **Hỗ trợ dịch máy, phân tích ngữ nghĩa, sinh văn bản tự động**.

**Ứng dụng**

Cây phân tích cú pháp (Parse Tree) có nhiều ứng dụng quan trọng trong xử lý ngôn ngữ tự nhiên (NLP) và các lĩnh vực liên quan. Dưới đây là một số ứng dụng chính:

**1. Xử lý Ngôn ngữ Tự Nhiên (NLP - Natural Language Processing)**

* **Phân tích cú pháp tự động**: Giúp máy tính hiểu cấu trúc của câu để xử lý thông tin chính xác hơn.
* **Nhận diện thực thể có tên (NER - Named Entity Recognition)**: Xác định các thực thể như tên người, địa điểm, tổ chức trong văn bản.
* **Tóm tắt văn bản (Text Summarization)**: Giúp hệ thống xác định các phần quan trọng của văn bản.
* **Trả lời câu hỏi (Question Answering)**: Giúp hệ thống AI hiểu câu hỏi và tìm câu trả lời chính xác.

Ví dụ: Trợ lý ảo như Siri, Google Assistant sử dụng phân tích cú pháp để hiểu câu lệnh của người dùng.

**2. Dịch Máy (Machine Translation - MT)**

* Dịch tự động từ ngôn ngữ này sang ngôn ngữ khác bằng cách phân tích cú pháp của câu nguồn và ánh xạ sang cấu trúc của câu đích.
* Ví dụ: Google Translate sử dụng mô hình phân tích cú pháp để tạo bản dịch tự nhiên hơn.

**3. Nhận diện và sửa lỗi ngữ pháp (Grammar Checking)**

* Các công cụ như Grammarly hoặc tính năng kiểm tra ngữ pháp của Microsoft Word sử dụng cây phân tích cú pháp để xác định lỗi ngữ pháp và đề xuất chỉnh sửa.
* Ví dụ: Nếu một câu thiếu chủ ngữ hoặc có lỗi thì hệ thống có thể phát hiện bằng cách kiểm tra cấu trúc cú pháp của câu.

**4. Phân tích ngữ nghĩa (Semantic Analysis)**

* Sau khi phân tích cú pháp, hệ thống có thể suy luận về **ngữ nghĩa (meaning)** của câu.
* Hỗ trợ chatbot và trợ lý ảo hiểu đúng ý định của người dùng.

Ví dụ: Khi bạn hỏi **"Who is the president of France?"**, hệ thống có thể xác định chủ ngữ ("Who"), động từ ("is"), và tân ngữ ("the president of France") để tìm kiếm câu trả lời chính xác.

**5. Trích xuất thông tin (Information Extraction - IE)**

* Tìm kiếm thông tin quan trọng từ văn bản dựa trên cấu trúc cú pháp.
* Ví dụ: Từ câu **"Apple Inc. acquired Beats Electronics in 2014"**, hệ thống có thể trích xuất:
  + **Công ty mua lại**: Apple Inc.
  + **Công ty bị mua**: Beats Electronics
  + **Năm**: 2014

**6. Xử lý Giọng Nói (Speech Processing)**

* Cây phân tích cú pháp giúp hệ thống chuyển giọng nói thành văn bản và hiểu nội dung câu nói.
* Ứng dụng trong trợ lý giọng nói (Alexa, Google Assistant).

**7. Tìm kiếm và Trích xuất Thông tin trên Web (Web Search & Information Retrieval)**

* Công cụ tìm kiếm như Google sử dụng phân tích cú pháp để hiểu ý định của người tìm kiếm và cung cấp kết quả phù hợp hơn.

Ví dụ: Khi bạn tìm kiếm **"best restaurants in New York"**, hệ thống phân tích cú pháp để xác định:

* **Chủ đề**: "restaurants"
* **Vị trí**: "New York"  
  => Sau đó trả về kết quả phù hợp.

**Khái niệm quan trọng trong phân tích cú pháp (parsing)**

**1. Ngữ pháp (Grammar)**

* **Định nghĩa**: Ngữ pháp là một tập hợp các quy tắc chính thức mô tả các cấu trúc hợp lệ trong một ngôn ngữ.
* **Ý nghĩa**:
  + Xác định cách các từ và cụm từ có thể được sắp xếp để tạo thành câu đúng.
  + Trong NLP, ngữ pháp có thể là **ngữ pháp phi ngữ cảnh (Context-Free Grammar - CFG)**, được dùng phổ biến trong phân tích cú pháp.
* **Ví dụ** về một bộ quy tắc ngữ pháp phi ngữ cảnh:

mathematica

CopyEdit

S → NP VP

NP → Det N | N

VP → V NP | V

Det → the | a

N → dog | cat

V → sees | eats

* + Với bộ quy tắc này, câu **"The dog sees a cat."** có thể được tạo ra theo các quy tắc đã định.

**2. Thuật toán phân tích cú pháp (Parsing Algorithm)**

* **Định nghĩa**: Thuật toán phân tích cú pháp là phương pháp xác định cấu trúc của một câu theo một ngữ pháp đã cho.
* **Ý nghĩa**:
  + Giúp xác định cây cú pháp (parse tree) của câu.
  + Có nhiều loại thuật toán phân tích cú pháp, tùy theo mục đích sử dụng.
* **Một số thuật toán phổ biến**:
  + **Top-down parsing**: Bắt đầu từ ký hiệu gốc (start symbol) và cố gắng phát triển nó theo quy tắc ngữ pháp.
  + **Bottom-up parsing**: Bắt đầu từ các từ trong câu và dần dần kết hợp chúng thành các thành phần lớn hơn theo ngữ pháp.
  + **Earley parser**: Một thuật toán parsing tổng quát cho ngữ pháp phi ngữ cảnh.
  + **CYK (Cocke–Younger–Kasami) parser**: Hiệu quả với ngữ pháp dạng Chomsky (CNF).

**3. Bộ phân tích cú pháp (Parser)**

* **Định nghĩa**: Parser là một chương trình phần mềm có nhiệm vụ xác định cấu trúc cú pháp của câu dựa trên một ngữ pháp cụ thể.
* **Vai trò**:
  + Kiểm tra xem một câu có hợp lệ theo ngữ pháp không.
  + Tạo ra cây phân tích cú pháp thể hiện quan hệ giữa các thành phần trong câu.
* **Ví dụ về Parser**:
  + **Stanford Parser**: Một công cụ phân tích cú pháp mạnh mẽ trong NLP.
  + **SpaCy, NLTK (Python)**: Các thư viện xử lý ngôn ngữ tự nhiên có tích hợp parser.
  + **Các trình biên dịch (compilers)** như GCC sử dụng parser để dịch mã nguồn thành mã máy.

**Ngữ pháp Đơn Giản (Simple Grammar)**

**1. Ngữ pháp là gì?**

* **Ngữ pháp (Grammar)** được mô tả dưới dạng **các quy tắc thay thế (rewrite rules)**.
* Đây là một tập hợp các quy tắc giúp xác định **cấu trúc hợp lệ của một ngôn ngữ**.

**2. Các loại ký hiệu trong ngữ pháp**

* **Ký hiệu đầu cuối (Terminal Symbols)**:
  + Là những ký hiệu **không thể phân tách** thêm nữa.
  + Ví dụ: "John", "garbage", "laughed", "walks".
* **Ký hiệu không đầu cuối (Nonterminal Symbols)**:
  + Là những ký hiệu **có thể phân tách** thành các phần nhỏ hơn theo quy tắc ngữ pháp.
  + Ví dụ: S, NP, VP.

**3. Xét một ngữ pháp đơn giản G**

Cho ngữ pháp G với các quy tắc:

S → NP VP

NP → John | garbage

VP → laughed | walks

* **S (Sentence - Câu)** gồm một **NP (Cụm danh từ)** và một **VP (Cụm động từ)**.
* **NP (Noun Phrase - Cụm danh từ)** có thể là "John" hoặc "garbage".
* **VP (Verb Phrase - Cụm động từ)** có thể là "laughed" hoặc "walks".

**4. Các câu có thể tạo ra từ ngữ pháp này**

Từ quy tắc trên, chúng ta có thể sinh ra các câu hợp lệ:

1. **John laughed.**
   * S → NP VP
   * NP → John
   * VP → laughed
2. **John walks.**
   * S → NP VP
   * NP → John
   * VP → walks
3. **Garbage laughed.**
   * S → NP VP
   * NP → garbage
   * VP → laughed
4. **Garbage walks.**
   * S → NP VP
   * NP → garbage
   * VP → walks

**5. Ứng dụng của Ngữ pháp Đơn Giản**

* **Phân tích cú pháp (Parsing)**: Xác định xem một câu có hợp lệ không.
* **Sinh văn bản tự động (Text Generation)**: Tạo ra các câu đúng ngữ pháp từ một tập quy tắc.
* **Dịch máy (Machine Translation)**: Xây dựng quy tắc chuyển đổi giữa các ngôn ngữ.

**Ngữ pháp phi ngữ cảnh (Context-Free Grammar - CFG)**

**1. Định nghĩa CFG**

Ngữ pháp phi ngữ cảnh (CFG) còn được gọi là **ngữ pháp cấu trúc cụm từ (Phrase Structure Grammar)**. Một CFG được biểu diễn dưới dạng **G = (T, N, P, S, R)**, trong đó:

* **T (Terminals)**: Tập hợp các ký hiệu đầu cuối (không thể phân tách tiếp).
* **N (Nonterminals)**: Tập hợp các ký hiệu không đầu cuối (có thể được thay thế bằng các quy tắc).
* **P (Preterminals)**: Là tập hợp con của N, luôn được thay thế bằng ký hiệu đầu cuối.
* **S (Start Symbol)**: Ký hiệu bắt đầu.
* **R (Rules)**: Tập hợp các quy tắc của ngữ pháp, có dạng X → γ, trong đó X là một **ký hiệu không đầu cuối**, còn γ là một **chuỗi chứa cả ký hiệu đầu cuối và không đầu cuối** (có thể rỗng).

**2. Một ngữ pháp G sinh ra một ngôn ngữ L**

* **G là một tập hợp các quy tắc sinh ra chuỗi ký hiệu**.
* **L là tập hợp tất cả các chuỗi hợp lệ có thể được sinh ra từ G**.

**3. Bộ nhận diện (Recognizer) và Bộ phân tích cú pháp (Parser)**

* **Bộ nhận diện (Recognizer)**: Xác định xem một chuỗi có thuộc ngôn ngữ L(G) hay không (trả về "yes" hoặc "no").
* **Bộ phân tích cú pháp (Parser)**: Xác định tất cả các cây phân tích cú pháp có thể có của một chuỗi theo G.

**Ví dụ 1: Ngữ pháp G1**

Cho ngữ pháp:

G1 = ({a,b}, {X}, X, {X → ε, X → aXb})

**Giải thích:**

* T = {a, b} là tập ký hiệu đầu cuối.
* N = {X} là tập ký hiệu không đầu cuối.
* S = X là ký hiệu bắt đầu.
* R gồm hai quy tắc:
  + X → ε (X có thể sinh ra chuỗi rỗng).
  + X → aXb (X có thể sinh ra aXb).

**Xác định ngôn ngữ L(G1):**  
Bắt đầu từ X, áp dụng các quy tắc sinh dần chuỗi:

* X → ε → **""** (chuỗi rỗng).
* X → aXb → ab.
* X → aXb → aaXbb → aabb.
* X → aXb → aaXbb → aaaXbbb → aaabbb.
* Tiếp tục tạo ra các chuỗi có dạng aⁿbⁿ (**số lượng "a" bằng số lượng "b"**).

**Ngôn ngữ sinh ra:**

L(G1) = {ε, ab, aabb, aaabbb, ...} = { aⁿbⁿ | n ≥ 0 }

G1 sinh ra **các chuỗi đối xứng dạng "a...b" với số lượng a và b bằng nhau**.

**Ví dụ 2: Ngữ pháp G2**

Cho ngữ pháp:

G2 = ({a,b}, {X}, X, {X→ε, X→aXb, X→XX})

**Giải thích:**

* T = {a, b} là tập ký hiệu đầu cuối.
* N = {X} là tập ký hiệu không đầu cuối.
* S = X là ký hiệu bắt đầu.
* R gồm ba quy tắc:
  + X → ε (X có thể sinh ra chuỗi rỗng).
  + X → aXb (X có thể sinh ra aXb).
  + X → XX (X có thể ghép hai X lại với nhau).

**Xác định ngôn ngữ L(G2):**  
Bắt đầu từ X, áp dụng các quy tắc sinh dần chuỗi:

* X → ε → **""** (chuỗi rỗng).
* X → aXb → ab.
* X → aXb → aaXbb → aabb.
* X → aXb → aaXbb → aaaXbbb → aaabbb.
* X → XX → aabbXX → aabbcc.

Vì có quy tắc X → XX, ta có thể ghép bất kỳ hai chuỗi hợp lệ nào sinh ra từ X.

**Ngôn ngữ sinh ra:**

CopyEdit

L(G2) = {ε, ab, aabb, aaabbb, aabbab, abaabb, ...}

* Không chỉ tạo ra các chuỗi đối xứng aⁿbⁿ, mà còn có thể ghép nhiều chuỗi kiểu này lại với nhau, tạo thành dạng **aⁿbⁿaᵐbᵐ**.
* Đây là **ngôn ngữ không chính quy**, có thể chứa nhiều đoạn aⁿbⁿ nối nhau.

**So sánh L(G1) và L(G2)**

| **Ngữ pháp** | **Ngôn ngữ sinh ra (L(G))** | **Đặc điểm** |
| --- | --- | --- |
| **G1** | {ε, ab, aabb, aaabbb, ...} | Chỉ có chuỗi đối xứng aⁿbⁿ (n ≥ 0), số lượng "a" phải bằng số lượng "b". |
| **G2** | {ε, ab, aabb, aaabbb, abaabb, aabbab, ...} | Có thể chứa nhiều đoạn aⁿbⁿ ghép lại, cho phép các cấu trúc phức tạp hơn. |

**Ứng dụng của Quy tắc Viết lại trong Ngữ pháp (Grammar Rewrite Rules)**

**1. Giới thiệu Quy tắc Viết lại (Rewrite Rules)**

Ngữ pháp có thể được biểu diễn bằng **các quy tắc viết lại**, trong đó **một ký hiệu không đầu cuối** có thể được thay thế bằng một **chuỗi ký hiệu đầu cuối và không đầu cuối**.

Ví dụ:

S → NP VP (Một câu gồm Cụm Danh từ + Cụm Động từ)

NP → DT NNS (Cụm danh từ gồm Mạo từ + Danh từ số nhiều)

VP → VBD (Cụm động từ là một động từ quá khứ)

**2. Ví dụ 1: Câu "The children slept"**

Áp dụng các quy tắc từng bước:

1. **S → NP VP**
2. **NP → DT NNS** (Cụm danh từ = Mạo từ + Danh từ số nhiều)
3. **VP → VBD** (Cụm động từ = Động từ quá khứ)
4. **DT → The, NNS → children, VBD → slept**

Kết quả:

S

├── NP

│ ├── DT → The

│ ├── NNS → children

└── VP

└── VBD → slept

Câu hoàn chỉnh: **"The children slept"**

**3. Ví dụ 2: Câu "The children ate the cake"**

Áp dụng các quy tắc từng bước:

1. **S → NP VP**
2. **NP → DT NNS** (Cụm danh từ = Mạo từ + Danh từ số nhiều)
3. **VP → VBD NP** (Cụm động từ = Động từ quá khứ + Cụm danh từ)
4. **NP → DT NN** (Cụm danh từ = Mạo từ + Danh từ số ít)
5. **DT → The, NNS → children, VBD → ate, NN → cake**

Cây phân tích cú pháp:

S

├── NP

│ ├── DT → The

│ ├── NNS → children

├── VP

│ ├── VBD → ate

│ ├── NP

│ │ ├── DT → The

│ │ ├── NN → cake

Câu hoàn chỉnh: **"The children ate the cake"**

**4. Ứng dụng của Quy tắc Viết lại**

Quy tắc viết lại trong ngữ pháp có nhiều ứng dụng quan trọng, bao gồm:

1. **Phân tích cú pháp (Parsing)**
   * Kiểm tra xem một câu có đúng ngữ pháp không.
   * Sử dụng trong trình biên dịch (compilers), xử lý ngôn ngữ tự nhiên (NLP).
2. **Sinh văn bản tự động (Text Generation)**
   * Hỗ trợ hệ thống chatbot, AI tạo nội dung tự động.
3. **Dịch máy (Machine Translation)**
   * Giúp phân tích và chuyển đổi câu từ ngôn ngữ này sang ngôn ngữ khác.
4. **Nhận diện thực thể (Named Entity Recognition - NER)**
   * Xác định danh từ, động từ, chủ ngữ, vị ngữ trong câu.

**Ngữ pháp Ngôn ngữ Tự nhiên có Tính Mơ hồ (Ambiguity in Natural Language Grammars)**

**1. Mơ hồ trong Ngôn ngữ Tự nhiên**

Ngữ pháp ngôn ngữ tự nhiên thường **mơ hồ**, nghĩa là một câu có thể có nhiều **cách phân tích cú pháp khác nhau**, dẫn đến nhiều nghĩa khác nhau.

Ví dụ: **"John saw snow on the campus."**  
Câu này có thể có hai cách hiểu, tùy vào việc **cụm giới từ (PP - Prepositional Phrase) "on the campus"** gắn vào đâu:

1. **VP Attachment (Gắn vào cụm động từ - Verb Phrase):**
   * Nghĩa: John **nhìn thấy tuyết khi ở trong khuôn viên trường**.
2. **NP Attachment (Gắn vào cụm danh từ - Noun Phrase):**
   * Nghĩa: John **nhìn thấy loại tuyết thuộc về khuôn viên trường**.

**2. Cây Phân tích Cú pháp (Parse Trees)**

**(1) PP gắn vào VP (John nhìn thấy tuyết khi ở trong khuôn viên trường)**

S

├── NP (John)

└── VP

├── V (saw)

├── NP (snow)

└── PP (on the campus)

**(2) PP gắn vào NP (John nhìn thấy loại tuyết thuộc về khuôn viên trường)**

S

├── NP

│ ├── NP (snow)

│ ├── PP (on the campus)

└── VP

├── V (saw)

**3. Nguyên nhân gây Mơ hồ**

* **Giới từ có thể gắn vào nhiều vị trí khác nhau.**
* **Các quy tắc viết lại trong ngữ pháp có thể dẫn đến nhiều cây phân tích cú pháp khác nhau.**
* **Cấu trúc câu không rõ ràng trong tiếng nói và văn bản.**

**4. Cách Giải quyết Mơ hồ**

1. **Dùng Ngữ cảnh (Context):**
   * Con người thường dựa vào **bối cảnh** để hiểu câu theo nghĩa phù hợp.
2. **Xây dựng Parser có Ưu tiên Gắn kết (Attachment Preference):**
   * Một số mô hình ngôn ngữ có thể ưu tiên **VP-attachment hoặc NP-attachment** dựa trên thống kê dữ liệu.
3. **Sử dụng Cây Xác suất (Probabilistic Parsing):**
   * Xác suất cao hơn có thể quyết định cách phân tích phù hợp hơn.

**Top-Down Parsing (Phân tích cú pháp từ trên xuống)**

**1. Tổng quan về Top-Down Parsing**

Phân tích cú pháp từ trên xuống (**Top-Down Parsing**) là phương pháp **bắt đầu từ ký hiệu bắt đầu (S) của ngữ pháp** và **dần mở rộng theo quy tắc** để cố gắng tạo ra câu đầu vào.

**2. Cách hoạt động của Top-Down Parsing**

1. **Bắt đầu với danh sách mục tiêu cần xây dựng**:
   * Ví dụ: {S} → {NP VP}
2. **So khớp ký hiệu đầu tiên với LHS (vế trái) của một quy tắc ngữ pháp.**
3. **Thay thế bằng RHS (vế phải) của quy tắc ngữ pháp.**
4. **Tiếp tục mở rộng và cố gắng khớp với câu đầu vào.**
5. **Nếu có nhiều quy tắc có thể áp dụng, cần chọn quy tắc phù hợp (vấn đề tìm kiếm).**
6. **Dừng lại khi toàn bộ chuỗi sinh ra khớp với câu đầu vào.**

**3. Ví dụ minh họa**

Giả sử có ngữ pháp:

S → NP VP

NP → DT N

VP → V NP

DT → the

N → cat

V → sees

Câu đầu vào: **"the cat sees"**

**Quá trình phân tích từ trên xuống:**

1. **Bắt đầu từ S**

S

1. **Áp dụng quy tắc S → NP VP**

S

├── NP

└── VP

1. **Áp dụng NP → DT N và VP → V NP**

S

├── NP

│ ├── DT

│ └── N

└── VP

├── V

└── NP

1. **Thay thế tiền tố (Preterminal) bằng từ cụ thể**

S

├── NP

│ ├── DT → the

│ └── N → cat

└── VP

├── V → sees

└── NP (thiếu phần mở rộng)

1. **Nếu câu đầu vào là "the cat sees", parser sẽ dừng tại đây.**

**4. Vấn đề của Top-Down Parsing**

Dù phương pháp này có thể hoạt động, nhưng nó có một số vấn đề lớn:

**(1) Quy tắc đệ quy trái (Left Recursive Rules)**

* Nếu có quy tắc **đệ quy trái** như:

S → S VP

Parser sẽ **lặp vô hạn**, vì S cứ tiếp tục mở rộng mà không tạo ra đầu cuối.

**(2) Lựa chọn quy tắc (Search Problem)**

* Nếu có nhiều quy tắc cho cùng một vế trái (LHS), parser phải **tìm kiếm** quy tắc phù hợp.
* Ví dụ:

NP → DT N

NP → Pronoun

Nếu có hai quy tắc, parser cần biết khi nào chọn DT N và khi nào chọn Pronoun.

**(3) Công việc vô ích (Useless Work)**

* Top-down parser mở rộng **cấu trúc không cần thiết** và sau đó **bị buộc phải quay lui (backtracking)** nếu không khớp với câu đầu vào.

**(4) Không phù hợp với phân tích từ vựng (Lexical Lookup)**

* Trong thực tế, việc **thay thế tiền tố (preterminals) bằng từ vựng (terminals)** **luôn được thực hiện theo hướng bottom-up**, vì top-down khó xử lý phần này.

**(5) Công việc lặp lại (Repeated Work)**

* Nếu một cấu trúc con chung xuất hiện nhiều lần trong quá trình phân tích, top-down parser có thể phải **phân tích lại nhiều lần thay vì lưu trữ kết quả**.

**5. Khi nào Top-Down Parsing hoạt động tốt?**

* Khi ngữ pháp có **cấu trúc kiểm soát chặt chẽ** (Grammar-driven control).
* Khi không có quy tắc **đệ quy trái**.
* Khi có cách **ưu tiên các quy tắc** để giảm tìm kiếm vô ích.

**6. Giải pháp cải thiện Top-Down Parsing**

1. **Loại bỏ đệ quy trái (Left Recursion Elimination)**
2. **Sử dụng dự đoán trước (Predictive Parsing) để tránh quay lui (Backtracking)**
3. **Kết hợp với Bottom-Up Parsing (Hybrid Parsing)** để xử lý từ vựng tốt hơn.
4. **Dùng phân tích cú pháp theo bảng (Chart Parsing) để lưu kết quả phân tích con và tránh làm lại.**

**Bottom-Up Parsing: Phân tích cú pháp từ dưới lên**

**1. Tổng quan về Bottom-Up Parsing**

Phương pháp **Bottom-Up Parsing** bắt đầu từ các từ đầu vào (**terminals**) và dần xây dựng lên đến ký hiệu bắt đầu (**start symbol, thường là S**).

* Đây là phương pháp **dựa trên dữ liệu (data-directed)**, trái ngược với **Top-Down Parsing (hướng theo mục tiêu, goal-directed)**.
* Nếu một đoạn trong danh sách mục tiêu (**goal list**) **khớp với vế phải (RHS) của một quy tắc**, ta thay thế nó bằng **vế trái (LHS)** của quy tắc đó.
* Kết thúc khi **goal list chỉ còn S**.

**2. Cách hoạt động của Bottom-Up Parsing**

Giả sử ta có ngữ pháp:

S → NP VP

VP → V NP

NP → DT N

DT → the

N → cat

V → sees

Và câu đầu vào: **"the cat sees"**

**Quá trình Bottom-Up Parsing**

1. **Bắt đầu với các từ đầu vào**

[the, cat, sees]

1. **Thay thế the → DT theo quy tắc DT → the**

[DT, cat, sees]

1. **Thay thế cat → N theo quy tắc N → cat**

[DT, N, sees]

1. **Thay thế DT N → NP theo quy tắc NP → DT N**

[NP, sees]

1. **Thay thế sees → V theo quy tắc V → sees**

[NP, V]

1. **Thay thế V NP → VP theo quy tắc VP → V NP**

[NP, VP]

1. **Thay thế NP VP → S theo quy tắc S → NP VP**

[S]

✅ **Hoàn thành!** Câu có thể được tạo bởi ngữ pháp.

**3. Các vấn đề của Bottom-Up Parsing**

**(1) Không hiệu quả khi có nhiều từ mơ hồ (Lexical Ambiguity)**

* Nếu một từ có nhiều cách phân loại (**ví dụ: "flies" có thể là danh từ hoặc động từ**), parser có thể tạo ra **nhiều cây cú pháp không cần thiết**.
* Điều này làm tăng số lượng bước xử lý, gây **tốn kém thời gian và bộ nhớ**.

**(2) Lặp lại công việc (Repeated Work)**

* Nếu một cấu trúc con xuất hiện nhiều lần trong câu, parser có thể **phải phân tích lại thay vì tận dụng kết quả cũ**.
* Ví dụ: Nếu parser gặp **"the cat sees the dog"**, thì phần "the cat" và "the dog" có cùng cấu trúc, nhưng parser có thể phải **phân tích lại từ đầu**.

**(3) Vấn đề tìm kiếm (Search Problem)**

* Nếu có nhiều quy tắc phù hợp cùng một lúc, parser **phải chọn quy tắc nào để áp dụng trước**.
* Có thể cần **tìm kiếm theo chiều sâu (DFS) hoặc theo chiều rộng (BFS)** để tìm cách phân tích đúng.

**(4) Độ phức tạp tính toán cao**

* Cả **Top-Down (LL) và Bottom-Up (LR)** **thường có độ phức tạp theo hàm mũ (exponential) đối với độ dài câu**, nhất là khi có nhiều quy tắc hoặc từ mơ hồ.

**4. Khi nào Bottom-Up Parsing hiệu quả?**

Bottom-Up Parsing hoạt động tốt khi:  
✔ Ngữ pháp **có ít lựa chọn quy tắc** (giảm tìm kiếm).  
✔ Câu đầu vào **ít chứa từ mơ hồ**.  
✔ **Sử dụng các thuật toán tối ưu hóa**, như **LR Parsing hoặc Chart Parsing**, để tránh làm lại công việc.

**5. So sánh Top-Down vs Bottom-Up Parsing**

| **Đặc điểm** | **Top-Down Parsing** | **Bottom-Up Parsing** |
| --- | --- | --- |
| **Cách tiếp cận** | Bắt đầu từ **S**, mở rộng dần xuống | Bắt đầu từ **từ đầu vào**, mở rộng dần lên |
| **Ưu điểm** | Điều hướng theo cấu trúc ngữ pháp | Không mở rộng các quy tắc không cần thiết |
| **Nhược điểm** | Khó xử lý từ vựng (terminals) | Dễ bị lặp lại công việc |
| **Vấn đề lớn** | Đệ quy trái (Left Recursion) | Mơ hồ từ vựng (Lexical Ambiguity) |
| **Cải tiến phổ biến** | Predictive Parsing (LL) | LR Parsing, CKY Algorithm |

**CKY Algorithm (Cocke-Kasami-Younger Algorithm)**

**1. Tổng quan về CKY Algorithm**

**CKY Algorithm** là một thuật toán **bottom-up** để kiểm tra xem một câu có thể được tạo ra bởi một ngữ pháp không. Đây là một thuật toán **lập trình động (Dynamic Programming)** giúp **giảm thiểu việc lặp lại công việc**, giúp phân tích cú pháp nhanh hơn.

📌 **Đặc điểm quan trọng:**  
✔ **Input:** Một chuỗi **n** từ và một tập hợp quy tắc ngữ pháp ở dạng **Chomsky Normal Form (CNF)**.  
✔ **Output:** **Yes/No** – xác định xem câu có thể được sinh ra từ ngữ pháp không.  
✔ **Dữ liệu:** **Bảng n × n (chart table)** để lưu trữ các thành phần cú pháp giữa các vị trí trong câu.

**2. Chomsky Normal Form (CNF) là gì?**

CNF là một dạng chuẩn của **Ngữ pháp phi ngữ cảnh (Context-Free Grammar - CFG)** với các quy tắc có **dạng chuẩn sau:**

* **A → BC** (Hai non-terminal sinh ra một non-terminal khác).
* **A → a** (Một non-terminal sinh ra một terminal).

📌 **Yêu cầu:**  
✔ Mọi quy tắc phải theo dạng **A → BC hoặc A → a**.  
✔ Không có **quy tắc đệ quy đơn A → B**.  
✔ Không có **ε (chuỗi rỗng) hoặc unit production (A → B)**.

**3. Ví dụ chuyển CFG sang CNF**

**Cho ngữ pháp G**

S → bA | aB | CA

A → bAA | aS | a

B → aBB | bS | b

**Chuyển sang dạng CNF**

1. **Thêm ký hiệu trung gian nếu cần**

S → X1A | X2B | CA

A → X1AA | X2S | a

B → X2BB | X1S | b

Trong đó:

* + X1 → b
  + X2 → a

1. **Đảm bảo tất cả quy tắc có dạng A → BC hoặc A → a**

S → X1A | X2B | CA

A → X1D | X2S | a

D → AA

B → X2E | X1S | b

E → BB

📌 **Bây giờ, tất cả các quy tắc đều ở dạng CNF!**

**4. Cách hoạt động của CKY Algorithm**

**Bước 1: Khởi tạo bảng CKY**

* Xây dựng **bảng n × n** (n là số từ trong câu).
* Mỗi ô **(i, j)** chứa **các thành phần cú pháp có thể tạo thành chuỗi con từ vị trí i đến j**.

**Bước 2: Điền bảng từ dưới lên (Bottom-Up)**

1. **Duyệt từng từ trong câu, tìm các non-terminals có thể tạo ra từ đó.**
2. **Kết hợp các thành phần nhỏ thành các thành phần lớn hơn theo quy tắc CNF.**
3. **Nếu ô (0, n) chứa ký hiệu bắt đầu (S), câu hợp lệ.**

**5. Ví dụ CKY Algorithm**

Cho **ngữ pháp CNF**:

S → NP VP

VP → V NP

NP → DT N

DT → "the"

N → "cat" | "dog"

V → "sees"

Và câu: **"the cat sees"**

**Bước 1: Khởi tạo bảng trống (3 × 3)**

| 1 | 2 | 3

------------------

0 | | |

1 | | |

2 | | |

**Bước 2: Xử lý từng từ**

* the → DT
* cat → N
* sees → V

Cập nhật bảng:

| 1 | 2 | 3

----------------------

0 | DT | N | V

1 | | NP | VP

2 | | | S

✅ **Kết quả:** Ô (0,3) chứa S → Câu hợp lệ!

**6. Độ phức tạp của CKY Algorithm**

* **O(n³)** với n là số từ trong câu.
* **Hiệu quả hơn các phương pháp khác** do sử dụng **lập trình động**.

**Bước 1:** **Khởi tạo bảng n × n**, mỗi ô chứa một set() để lưu trữ các non-terminals.

**Bước 2:** **Xử lý từng từ trong câu**, tìm các non-terminals có thể sinh ra từ đó.

**Bước 3:** **Duyệt từng phần của câu từ nhỏ đến lớn**, kiểm tra các cách kết hợp theo quy tắc CNF.

**Bước 4:** **Kiểm tra xem S có xuất hiện ở góc trên bên phải của bảng không**.

**Ngữ pháp phi ngữ cảnh (Context-Free Grammar - CFG)**

**1. Giải thích các quy tắc ngữ pháp**

Ngữ pháp này có các quy tắc như sau:

**Cấu trúc chính**

1. S → NP VP
   * Một câu (S) bao gồm **Cụm danh từ (NP) + Cụm động từ (VP)**.

**Cấu trúc Cụm danh từ (NP - Noun Phrase)**

1. NP → Det Noun
   * Cụm danh từ có thể gồm **Mạo từ (Det) + Danh từ (Noun)**.
2. NP → Name
   * Một cụm danh từ có thể chỉ là một **Tên riêng (Name)**.
3. NP → Name PP
   * Một **Tên riêng** có thể đi kèm một **Cụm giới từ (PP)**.

**Cấu trúc Cụm giới từ (PP - Prepositional Phrase)**

1. PP → Prep NP
   * Một **Cụm giới từ** gồm **Giới từ (Prep) + Cụm danh từ (NP)**.

**Cấu trúc Cụm động từ (VP - Verb Phrase)**

1. VP → V NP
   * Một cụm động từ gồm **Động từ (V) + Cụm danh từ (NP)**.
2. VP → V NP PP
   * Một cụm động từ có thể gồm **Động từ (V) + Cụm danh từ (NP) + Cụm giới từ (PP)**.

**Từ vựng (Lexical Rules)**

* V → ate (Động từ)
* Name → John | ice-cream | snow (Tên riêng)
* Noun → ice-cream | pizza | table | guy | campus (Danh từ)
* Det → the (Mạo từ)
* Prep → on (Giới từ)

**2. Ví dụ sinh câu từ ngữ pháp này**

**Ví dụ 1: "John ate pizza"**

**Bước phân tích cú pháp:**

S

├── NP (John)

└── VP

├── V (ate)

└── NP (pizza)

✅ **Câu hợp lệ!**

**Ví dụ 2: "John ate ice-cream on the campus"**

**Bước phân tích cú pháp:**

S

├── NP (John)

└── VP

├── V (ate)

├── NP (ice-cream)

└── PP

├── Prep (on)

└── NP

├── Det (the)

└── Noun (campus)

✅ **Câu hợp lệ!**

**3. Ứng dụng của Ngữ pháp phi ngữ cảnh (CFG)**

🔹 **Xử lý ngôn ngữ tự nhiên (NLP):** Parsing, Machine Translation, Speech Recognition.  
🔹 **Phân tích cú pháp trong trình biên dịch:** Xác định cú pháp mã nguồn trong lập trình.  
🔹 **Hệ thống Chatbot & AI:** Phân tích cấu trúc câu đầu vào để tạo phản hồi phù hợp.

**Quy tắc Kết hợp trong CKY Parsing**

Trong thuật toán CKY **(Cocke-Kasami-Younger)**, một ô **Cell[i, j]** chứa **non-terminal X** nếu:

✅ **Tồn tại một quy tắc**:

X→YZX → YZX→YZ

✅ **Có thể phân chia câu thành hai phần phù hợp**:

* Cell[i, k] chứa **Y**
* Cell[k, j] chứa **Z**
* **Với k nằm giữa i và j**

**Ví dụ minh họa**

**Câu: "the cat"**

**Ngữ pháp (CNF)**

S → NP VP

NP → DT NN

DT → the

NN → cat

**Bước phân tích trong CKY Table**

| **i/j** | **0** | **1** | **2** |
| --- | --- | --- | --- |
| 0 | {} | DT | NP |
| 1 | {} | {} | NN |
| 2 | {} | {} | {} |

* Cell[0,1] = {DT} (the → DT)
* Cell[1,2] = {NN} (cat → NN)
* Cell[0,2] = {NP} vì có quy tắc **NP → DT NN**

✅ **Kết quả:** Cụm danh từ "the cat" hợp lệ! 🚀

**Ứng dụng trong CKY Parsing**

✔ **Xây dựng bảng parsing tự động**  
✔ **Kiểm tra cú pháp câu dựa trên quy tắc ngữ pháp CNF**  
✔ **Ứng dụng trong xử lý ngôn ngữ tự nhiên (NLP)**

A screenshot of a computer

Description automatically generated

d