TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

TIỂU LUẬN

Môn: SEMINAR CHUYÊN ĐỀ - LỚP: DKP1201
NĂM HỌC: 2024 - 2025

Chuyên đề: LẬP TRÌNH LOGIC VÀ ỨNG DỤNG

Đề tài: Xây dựng công cụ phân tích cú pháp

Sinh viên thực hiện:	Phan Duy Cửu
Mã số sinh viên:	3121410101
Giảng viên hướng dẫn:	PGS. TS Nguyễn Tuấn Đăng

Thành phố Hồ Chí Minh, tháng 11 năm 2024

LỜI NÓI ĐẦU

Trong thời đại công nghệ phát triển nhanh chóng như hiện nay, việc áp dụng logic vào lập trình và phát triển các ứng dụng đã trở thành một yếu tố then chốt trong lĩnh vực khoa học máy tính. Môn học "Lập trình logic và ứng dụng" không chỉ cung cấp cho sinh viên những kiến thức cơ bản và chuyên sâu về logic trong lập trình, mà còn giúp rèn luyện khả năng phân tích, giải quyết vấn đề một cách có hệ thống và chặt chẽ.

Bài tiểu luận này được thực hiện với đề tài "Xây dựng công cụ phân tích cú pháp", nhằm mục đích nghiên cứu và ứng dụng các phương pháp lập trình logic trong việc xây dựng một công cụ có khả năng phân tích cú pháp tự động. Qua đó, người học có thể hiểu rõ hơn về tầm quan trọng của logic trong quá trình xây dựng và phát triển các ứng dụng phần mềm.

Tôi xin chân thành cảm ơn thầy Nguyễn Tuấn Đăng đã tận tình hướng dẫn và cung cấp những kiến thức quý báu trong suốt khóa học. Chính những bài giảng và sự hỗ trợ của thầy đã giúp tôi có động lực và định hướng rõ ràng hơn trong quá trình hoàn thành bài tiểu luận này. Rất mong nhận được những ý kiến đóng góp của thầy để tôi có thể học hỏi và cải thiện hơn nữa trong tương lai.

NHẬN XÉT CỦA GIẢNG VIÊN

MỤC LỤC

I. Giới thiệu

Trong lĩnh vực xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP), phân tích cú pháp là một trong những bước quan trọng nhằm hiểu và biểu diễn cấu trúc ngữ nghĩa của các câu trong ngôn ngữ tự nhiên. Các kỹ thuật phân tích cú pháp giúp xác định mối quan hệ giữa các thành phần của câu như chủ ngữ, động từ, tân ngữ,... từ đó tạo nền tảng cho các ứng dụng phức tạp hơn như dịch máy, phân loại văn bản, trích xuất thông tin và hội thoại tự động. Một trong những cách tiếp cận nổi bật trong phân tích cú pháp là sử dụng văn phạm DCG (Definite Clause Grammar), một dạng văn phạm phi ngữ cảnh giúp biểu diễn các quy tắc cú pháp và ngữ nghĩa dưới dạng các luật logic.

Tiểu luận này sẽ trình bày tổng quan về lịch sử nghiên cứu phân tích cú pháp trong khoa học máy tính, đặc biệt tập trung vào cách tiếp cận dựa trên luật và lập trình logic, cũng như vai trò của DCG trong việc xây dựng các công cụ phân tích cú pháp. Cụ thể, chúng ta sẽ xây dựng một bộ phân tích cú pháp sử dụng văn phạm DCG để phân tích tập ngữ liệu cho trước gồm năm câu tiếng Việt. Công cụ phân tích này sẽ cho phép sinh ra các cây cú pháp cho từng câu, giúp làm rõ cấu trúc ngữ nghĩa của câu.

Nội dung tiểu luận bao gồm:

- Tổng quan lịch sử về các nghiên cứu phân tích cú pháp trong khoa học máy tính.
- Phương pháp phân tích cú pháp dựa trên luật và lập trình logic.
- Quy trình xây dựng công cụ phân tích cú pháp sử dụng văn phạm DCG.
- Thử nghiệm và minh họa cây cú pháp cho các câu trong tập dữ liệu thử nghiệm.

II. Lịch sử nghiên cứu phân tích cú pháp trong khoa học máy tính

a) Khái quát

Khái niệm: Phân tích cú pháp (tiếng Anh: **parsing**, **syntax analysis**, hoặc **syntactic analysis**) là một quá trình phân tích một chuỗi các biểu tượng, sử dụng trong ngôn ngữ tự nhiên, ngôn ngữ máy tính và các cấu trúc dữ liệu, tuân theo các quy tắc của ngữ pháp hình thức (formal grammar). Thuật ngữ *parsing* đến từ từ Latin *pars* (*orationis*), nghĩa là từ loại.

Thuật ngữ này có đôi chút khác biệt trong các chuyên ngành ngôn ngữ học và khoa học máy tính. Phân tích cú pháp một câu truyền thống thông thường được thực thi dưới dạng một phương pháp hiểu ý nghĩa chính xác của một câu hoặc từ, đôi khi với sự trợ giúp của các thiết kế chẳng hạn như các biểu đồ câu. Theo đó,

tầm quan trọng của các bộ phận ngữ pháp như chủ ngữ và vị ngữ được nhấn mạnh.

Với khoa học máy tính, thuật ngữ được dùng để phân tích ngôn ngữ máy tính, nhắc đến cách phân tích cú pháp của mã đầu vào thành các phần thành phần của nó nhằm tạo điều kiện thuận lợi cho việc viết trình biên dịch và trình thông dịch. Thuật ngữ cũng dùng cho việc mô tả sự tách biệt hay phân chia.

b) Lịch sử

- **Thời kỳ sơ khai (những năm 1950)**

  Những năm 1950 chứng kiến sự ra đời của các ngôn ngữ lập trình đầu tiên như **Fortran** và **LISP**, đi kèm với nhu cầu cần có công cụ để dịch mã nguồn sang mã máy.

  Noam Chomsky, một nhà ngôn ngữ học, đã phát triển **lý thuyết về ngữ pháp hình thức** vào cuối thập kỷ 1950. Ông đưa ra hệ phân cấp Chomsky (Chomsky hierarchy), chia các ngữ pháp thành 4 loại: ngữ pháp chính quy, phi ngữ cảnh, nhạy ngữ cảnh, và không giới hạn. Phân loại này trở thành nền tảng để hiểu và xây dựng các công cụ phân tích cú pháp.

- **Sự phát triển của các thuật toán phân tích cú pháp cơ bản (những năm 1960)**

  Trong những năm 1960, nhu cầu về phân tích cú pháp phức tạp hơn tăng cao do các ngôn ngữ lập trình mới và các hệ thống phần mềm lớn bắt đầu xuất hiện.

  Donald Knuth, một nhà khoa học máy tính nổi tiếng, đã giới thiệu **phân tích cú pháp LR(k)** vào năm 1965. Thuật toán LR cho phép xử lý cú pháp các ngôn ngữ phi ngữ cảnh, được thiết kế để đọc mã từ trái sang phải và sử dụng phân tích theo hướng dẫn từ phải sang trái. Các thuật toán phân tích cú pháp như **LL(k)** và **LR(k)** từ đó ra đời và được ứng dụng trong trình biên dịch.

  Một số thuật toán đáng chú ý khác bao gồm:

  Thuật toán CYK (Cocke-Younger-Kasami): Phân tích cú pháp theo hướng từ dưới lên cho các ngữ pháp phi ngữ cảnh, sử dụng cấu trúc bảng để lưu trữ trạng thái và tối ưu hóa thời gian xử lý.

  Thuật toán Earley (1970): Khả năng phân tích các ngữ pháp phi ngữ cảnh chung mà không cần biến đổi ngữ pháp.

- **Bùng nổ công cụ và sự phát triển trong trình biên dịch (những năm 1970-1980)**

  Đến những năm 1970, phân tích cú pháp trở thành một phần quan trọng trong thiết kế trình biên dịch. Các công cụ như **Yacc (Yet Another Compiler-Compiler)** ra đời vào năm 1975 tại AT&T Bell Labs để giúp phát triển trình biên dịch dễ dàng hơn.

  Lex được sử dụng cùng với Yacc để phân tích cú pháp từ vựng và ngữ pháp của mã nguồn.

  Các công cụ như Yacc và Lex, sau này là **Bison** (phiên bản mã nguồn mở của Yacc), giúp các nhà phát triển định nghĩa cú pháp của ngôn ngữ và sinh mã tự động cho trình biên dịch.

  Phân tích cú pháp LALR (Lookahead LR): Một biến thể của phân tích cú pháp LR, được phát triển để xử lý cú pháp hiệu quả hơn cho nhiều ngôn ngữ lập trình. Nó đã trở thành tiêu chuẩn trong các công cụ trình biên dịch vào thời điểm này.

- **Phân tích cú pháp trong xử lý ngôn ngữ tự nhiên (những năm 1990-2000)**

  Với sự phát triển của **xử lý ngôn ngữ tự nhiên (NLP)**, nghiên cứu phân tích cú pháp mở rộng sang lĩnh vực này. Việc phân tích cú pháp trong ngôn ngữ tự nhiên gặp nhiều thách thức vì ngôn ngữ tự nhiên có cấu trúc phức tạp hơn ngôn ngữ lập trình.

  Các kỹ thuật như **Cấu trúc cú pháp phụ thuộc (Dependency Parsing)** và **Cấu trúc cú pháp cụm từ (Constituency Parsing)** bắt đầu được sử dụng để phân tích ngôn ngữ tự nhiên.

  Các phương pháp **dựa trên học máy** (machine learning) bắt đầu được ứng dụng, cho phép xử lý cú pháp linh hoạt hơn trong văn bản tự nhiên. Những năm 2000 chứng kiến sự phát triển của các mô hình thống kê, như **Mô hình Markov ẩn** (Hidden Markov Models - HMMs) và **Mô hình chuỗi điều kiện (Conditional Random Fields - CRFs)**, giúp tự động nhận diện cấu trúc cú pháp dựa trên dữ liệu huấn luyện.

- **Thời kỳ hiện đại và phương pháp dựa trên học sâu (2010 đến nay)**

  Những năm 2010 là thời kỳ bùng nổ của **học sâu (Deep Learning)** trong xử lý ngôn ngữ tự nhiên. Các mô hình học sâu không yêu cầu cấu trúc cú pháp truyền

thống mà dựa vào khả năng của các mạng nơ-ron để học các biểu diễn phức tạp từ dữ liệu văn bản.

Các **mô hình Transformer** như **BERT, GPT** của Google và OpenAI có khả năng xử lý cú pháp văn bản một cách ngữ cảnh hóa cao, mang lại hiệu suất vượt trội so với các phương pháp truyền thống. Các mô hình này sử dụng kiến trúc attention để học cấu trúc ngữ cảnh trong văn bản, thay vì dựa vào cấu trúc ngữ pháp cứng nhắc.

Hiện nay, các công cụ và thư viện NLP như **spaCy, AllenNLP** và các mô hình mã nguồn mở như **Hugging Face Transformers** giúp các nhà phát triển ứng dụng mô hình NLP hiện đại mà không cần phải thực hiện các bước phân tích cú pháp truyền thống.

III. Cách tiếp cận phân tích cú pháp dựa trên luật và lập trình logic

a) Phân tích cú pháp dựa trên luật

- Tổng quát
  - ❖ Phân tích cú pháp dựa trên luật là một cách tiếp cận trong xử lý ngôn ngữ tự nhiên, trong đó cú pháp của câu được phân tích dựa trên một tập hợp các quy tắc xác định, hay còn gọi là văn phạm. Thay vì dựa vào thống kê và xác suất để suy đoán cấu trúc câu, phương pháp này sử dụng các quy tắc ngữ pháp được định nghĩa rõ ràng để phân tích cấu trúc của câu, nhận dạng các thành phần như chủ ngữ, động từ, tân ngữ và các mối quan hệ giữa chúng.
- Quy trình phân tích cú pháp dựa trên luật
  - ❖ Xác định ngữ pháp: Đầu tiên, ta cần xác định ngữ pháp của ngôn ngữ cần phân tích. Ngữ pháp thường được định nghĩa bằng các quy tắc.
  - ❖ Ví dụ:
    - ➢ · S → NP VP (Câu = Chủ ngữ + Vị ngữ)
    - ➢ · NP → Det N (Chủ ngữ = Định từ + Danh từ)
  - ❖ Xây dựng bộ quy tắc: Các quy tắc này sẽ mô tả cách mà các thành phần trong câu kết hợp với nhau. Ví dụ, ta có thể có quy tắc cho danh từ, động từ, và các phần khác trong câu.
  - ❖ Ví dụ:
    - ➢ · Det → "một" , "cái" , "nhiều"
    - ➢ · N → "sách" , "bàn" , "máy tính"
    - ➢ · V → "đọc" , "viết", "xem"

- ❖ Phân tích cú pháp: Sử dụng các thuật toán phân tích cú pháp như LL, LR hoặc các biến thể của chúng để chuyển đổi một chuỗi đầu vào thành một cấu trúc cây cú pháp. Cây cú pháp này giúp biểu diễn cấu trúc ngữ nghĩa của câu.

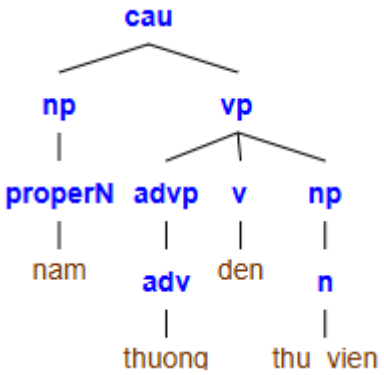
b) Lập trình logic

- Tổng quát
  - ❖ Lập trình logic là cách sử dụng các ngôn ngữ lập trình logic (như Prolog) để biểu diễn các quy tắc ngữ pháp và thực hiện phân tích cú pháp. Các ngôn ngữ này cho phép định nghĩa các quy tắc dưới dạng các mệnh đề logic, giúp biểu diễn các quan hệ ngữ pháp một cách tự nhiên và dễ hiểu. Trong lập trình logic, các quy tắc ngữ pháp có thể được biểu diễn dưới dạng các luật logic, và quá trình phân tích cú pháp sẽ dựa trên việc suy diễn các mệnh đề này.
- Quy trình lập trình logic
  - ❖ **Định nghĩa sự thật và quy tắc**: Trong lập trình logic, ta định nghĩa các sự thật (facts) và quy tắc (rules) bằng ngôn ngữ logic, thường là Prolog.
  - ❖ **Sử dụng truy vấn**: Ta có thể đặt truy vấn để tìm kiếm các sự thật dựa trên các quy tắc đã định nghĩa. Truy vấn này sẽ kích hoạt quá trình suy diễn logic để tìm ra kết quả.
  - ❖ **Suy diễn**: Lập trình logic cho phép ta sử dụng các kỹ thuật suy diễn như suy diễn tiến (forward chaining) và suy diễn lùi (backward chaining) để giải quyết các bài toán phức tạp.

- **Văn phạm DCG (Definite Clause Grammar)** là một ví dụ tiêu biểu của phân tích cú pháp dựa trên luật và lập trình logic. DCG là một loại văn phạm phi ngữ cảnh được biểu diễn dưới dạng các mệnh đề logic trong Prolog, giúp định nghĩa các quy tắc cú pháp của ngôn ngữ tự nhiên một cách linh hoạt. Một bộ phân tích cú pháp sử dụng DCG có thể xác định cấu trúc của câu bằng cách áp dụng tuần tự các quy tắc, từ đó sinh ra cây cú pháp hoặc biểu diễn cấu trúc ngữ nghĩa của câu.
- Ví dụ sử dụng văn phạm DCG trong prolog
  - ❖ Ta có câu “Nam thường đến thư viện”
    Với quy ước :
    properN: danh từ riêng
    np: cụm danh từ
    vp: cụm động từ
    adv: trạng từ

advp: cụm trạng từ
v: động từ
n: danh từ

Ta có thể phân tích cú pháp của câu trên dựa trên luật thành :



Xây dựng DCG trong Prolog:

```
properN --> [nam].
adv --> [thuong].
v --> [den].
n --> [thu,vien].
advp--> adv.
np --> properN.
np --> n.
vp --> advp, v, np.
cau --> np, vp.
```

Truy vấn trong SWI-Prolog :

```
?- phrase(cau, [nam,thuong,den,thu,vien]).
true
```

IV. Xây dựng công cụ phân tích cú pháp dựa trên luật

a) Dữ liệu

- Tập dữ liệu gốc:
  ➔ Nam thường đến thư viện.

➔ Nam rất thích đọc sách ở thư viện.
➔ Nhà của Nam ở gần trường.
➔ Nam mới mua mấy cuốn sách mới.
➔ Nam tặng Lan một cuốn sách rất hay.

- Câu phái sinh:

Các câu phái sinh có thể có của câu : Nam thường đến thư viện.

➔ Nam đến thư viện.

Các câu phái sinh có thể có của câu : Nam rất thích đọc sách ở thư viện.

➔ Nam đọc sách ở thư viện.
➔ Nam rất thích đọc sách.
➔ Nam thích đọc sách ở thư viện.
➔ Nam thích đọc sách.
➔ Nam rất thích sách.
➔ Nam thích sách.

Các câu phái sinh có thể có của câu : Nhà của Nam ở gần trường.

➔ Nam ở gần trường.

Các câu phái sinh có thể có của câu : Nam mới mua mấy cuốn sách mới.

➔ Nam mới mua mấy cuốn sách.
➔ Nam mua mấy cuốn sách mới.

Các câu phái sinh có thể có của câu : Nam tặng Lan một cuốn sách rất hay.

➔ Nam tặng lan một cuốn sách.

Các câu phát sinh kết hợp từ 5 câu trên có thể là:

➔ Trường của Nam ở gần thư viện.
➔ Nam mới mua mấy cuốn sách rất hay.
➔ Nam tặng Lan một cuốn sách rất mới.
➔ Nam tặng Lan một cuốn sách mới.
➔ Nam tặng Lan mấy cuốn sách mới.
➔ Nam tặng Lan mấy cuốn sách rất mới.
➔ Nam thường đọc sách ở thư viện.
➔ Nam thường đọc sách ở nhà của Lan.

➔ Nam thường mua sách ở thư viện.
➔ Lan thích cuốn sách Nam tặng.
➔ Lan đến thư viện gần trường của Nam.
➔ Nam mua sách ở trường gần thư viện.
➔ Nam thường đến trường đọc sách.
➔ Thư viện gần nhà của Nam.
➔ Nam thường đọc sách ở trường.
➔ Lan rất thích mấy cuốn sách của nam.
➔ Nam đến thư viện tặng Lan một cuốn sách.
➔ Nam thường mua sách ở thư viện.
➔ Nam thích mua sách ở trường.
➔ Nam thích đọc sách ở trường.
➔ Nam rất thích Lan
➔ Nam mới mua cuốn sách ở thư viện
➔ Nam mới mua mấy cuốn sách mới ở thư viện
➔ Nam mới mua mấy cuốn sách rất hay ở thư viện
➔ Nam rất thích đọc sách ở nhà Lan.
➔ Nam thích đọc sách ở nhà của Lan.
➔ Nam đến trường đọc sách.
➔ Nam thường tặng sách ở trường.
➔ Nam rất thích mua sách ở thư viện.
➔ Nam rất thích ở thư viện.
➔ Trường của Nam thường mua sách ở thư viện.

b) Văn phạm DCG

- **Xây dựng văn phạm DCG**
  - **Xây dựng các quy tắc :**

    s (sentence) : câu
    np (noun phrase) : cụm danh từ
    n (noun) : danh từ
    propern (proper noun): danh từ riêng
    vp (verb phrase) : cụm động từ
    v (verb) : động từ
    advp (adverb phrase) : cụm trạng từ
    adv (adverb) : trạng từ
    adjp (adjective phrase): cụm tính từ
    adj (adjective) : tính từ
    pp (prepositional phrase): cụm giới từ

p (preposition) ; giới từ
det (determiner) : từ hạn định

- **Xây dựng các luật DCG từ tập dữ liệu gốc**

```
properN(properN(nam)) --> [nam].
properN(properN(lan)) --> [lan].

n(n(thu_vien)) --> [thu,vien].
n(n(sach)) --> [sach].
n(n(truong)) --> [truong].
n(n(nha)) --> [nha].
n(n(cuon)) --> [cuon].

v(v(den)) --> [den].
v(v(thich)) --> [thich].
v(v(doc)) --> [doc].
v(v(tang)) --> [tang].
v(v(mua)) --> [mua].

adv(adv(rat)) --> [rat].
adv(adv(thuong)) --> [thuong].
adv(adv(moi)) --> [moi].
```

```
adj(adj(gan)) --> [gan].

adj(adj(moi)) --> [moi].

adj(adj(hay)) --> [hay].


det(det(may)) --> [may].

det(det(mot)) --> [mot].


p(p(o)) --> [o].

p(p(cua)) --> [cua].


adjp(adjp(X)) --> adj(X).

advp(advp(X)) --> adv(X).


pp(pp(P, NP)) --> p(P), np(NP).


np(np(ProperN)) --> properN(ProperN).

np(np(N)) --> n(N).

np(np(N1, N2)) --> n(N1), n(N2).

np(np(N, PP)) --> n(N), pp(PP).

np(np(AdjP, N)) --> adjp(AdjP), n(N).

np(np(Det, N1, N2, AdjP)) --> det(Det), n(N1), n(N2), adjp(AdjP).
```

```
np(np(Det, N1, N2, AdvP, AdjP)) --> det(Det), n(N1), n(N2), advp(AdvP),
adjp(AdjP).


vp(vp(AdvP, V, NP)) --> advp(AdvP), v(V), np(NP).

vp(vp(AdvP, V, V2, NP)) --> advp(AdvP), v(V),v(V2), np(NP).

vp(vp(V, NP)) --> v(V), np(NP).


s(s(NP, VP)) --> np(NP), vp(VP).

s(s(NP, VP, PP)) --> np(NP), vp(VP), pp(PP).

s(s(NP, PP)) --> np(NP), pp(PP).

s(s(NP, VP, NP2)) --> np(NP), vp(VP), np(NP2).

parse_tree(Sentence, Tree) :-

        phrase(s(Tree), Sentence).
```

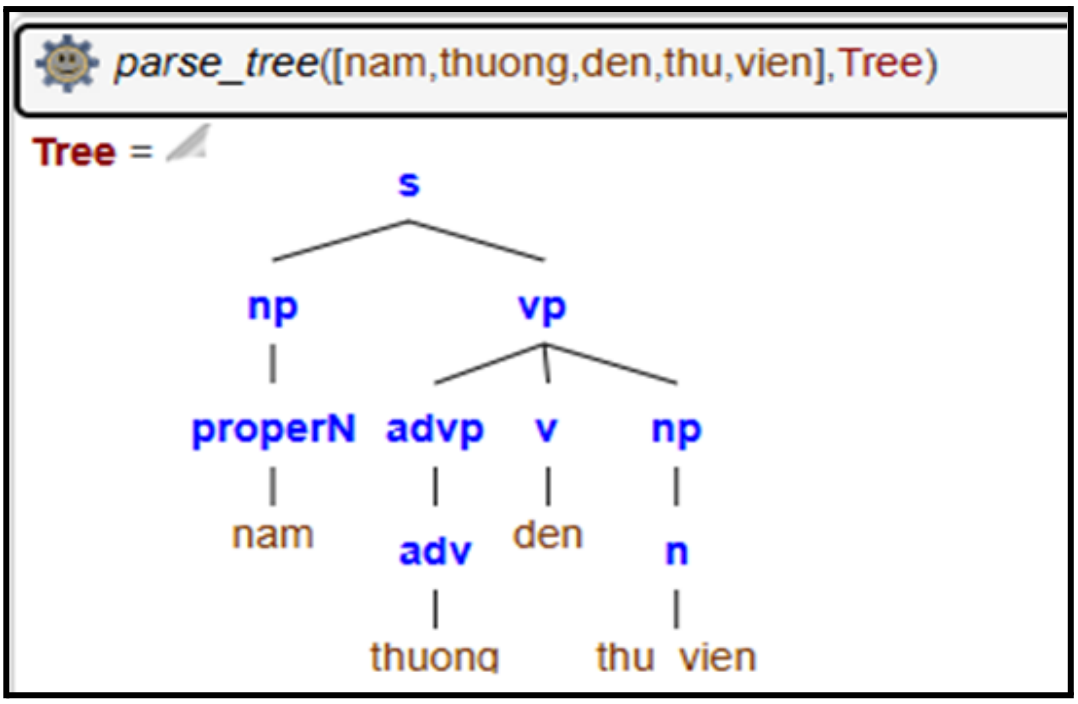
V. Thử nghiệm

a) Môi trường thử nghiệm : SWISH SWI-Prolog

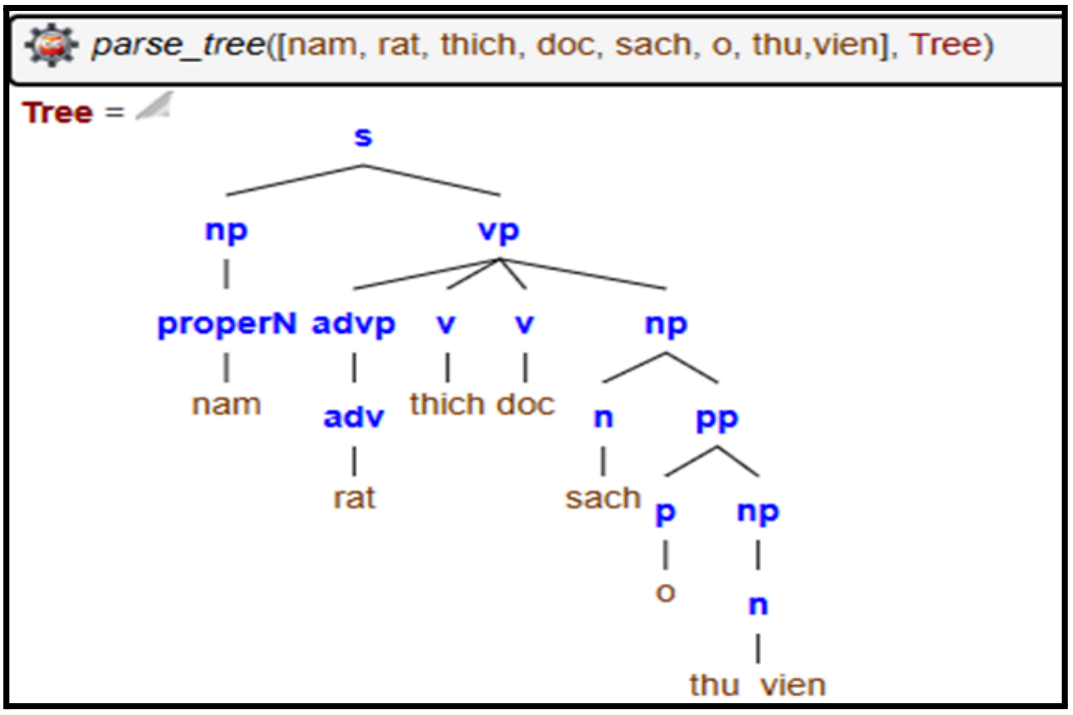
b) Các câu thử nghiệm và kết quả

i. Các câu trong tập dữ liệu gốc:

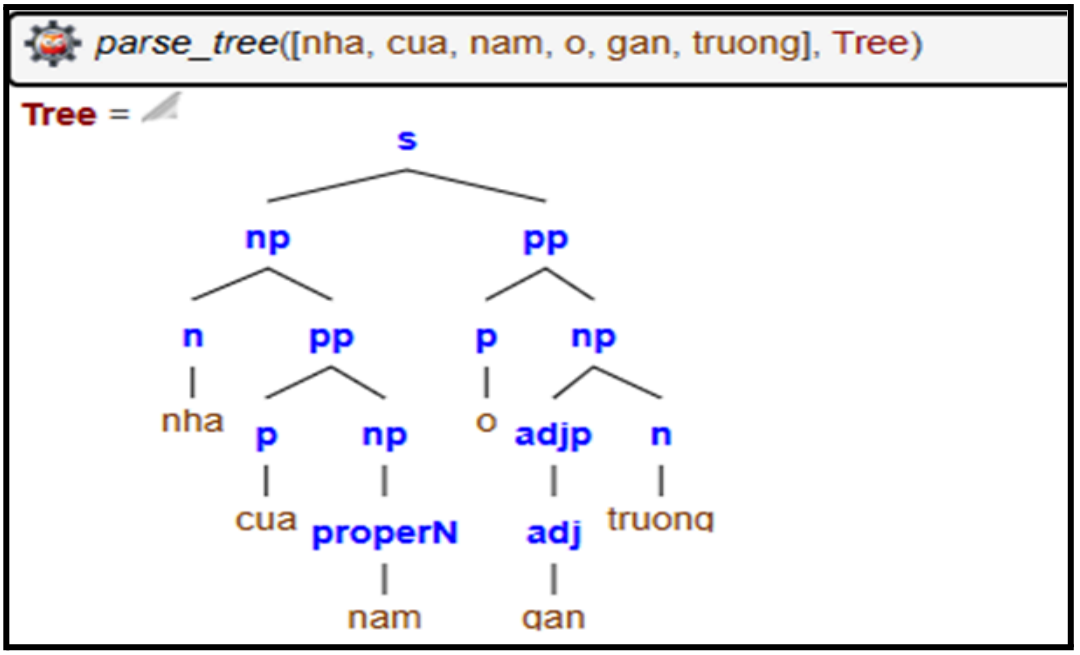
- Nam thường đến thư viện
  Cây cú pháp:



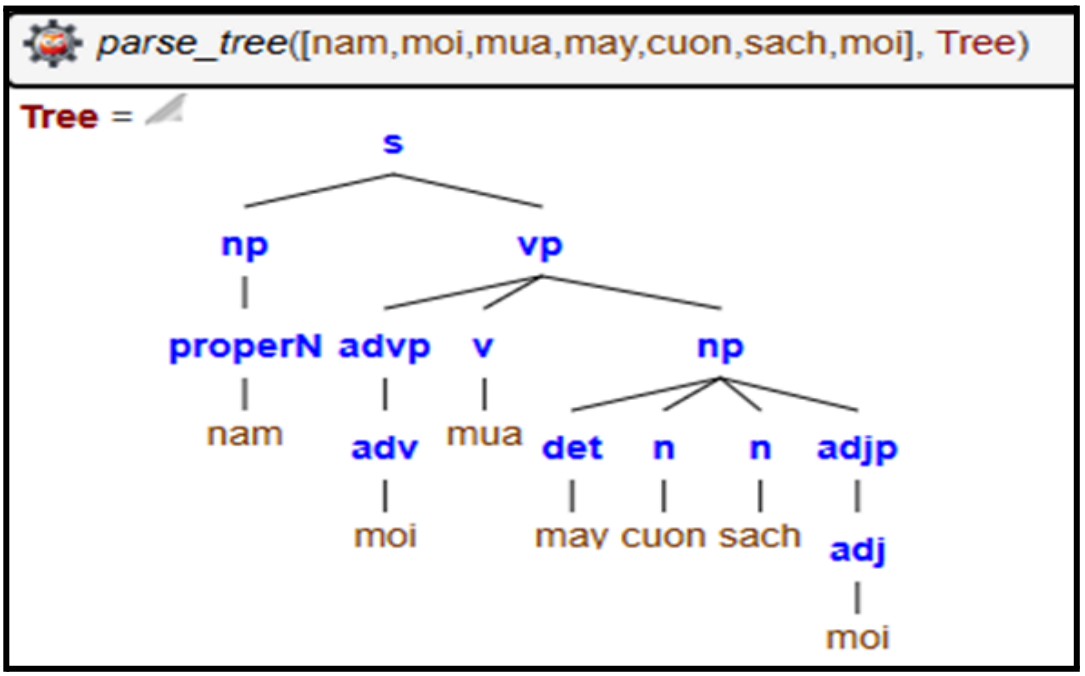
- Nam rất thích đọc sách ở thư viện
  Cây cú pháp:



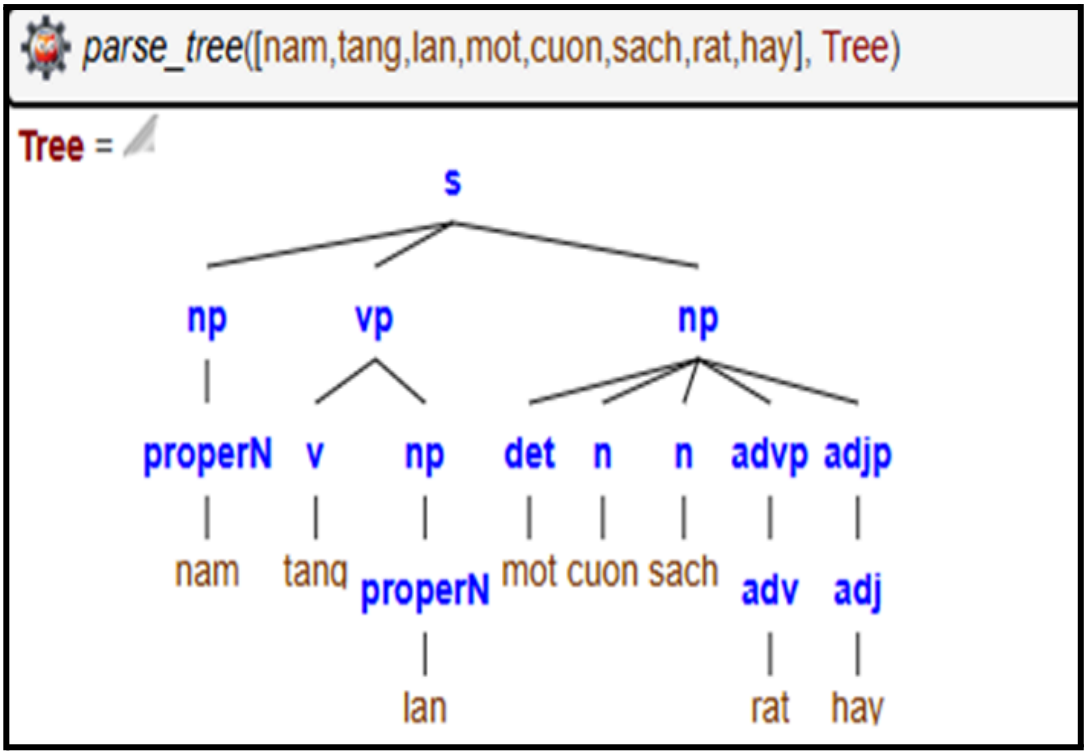
- Nhà của Nam ở gần trường
  Cây cú pháp:



- Nam mới mua mấy cuốn sách mơi
  Cây cú pháp:



- Nam tặng Lan một cuốn sách rất hay
  Cây cú pháp:



ii. Các câu phái sinh:

- Từ câu "Nam thường đến thư viện."

➔ Nam đến thư viện.

```
?- parse_tree([nam,den,thu,vien], Tree).

Tree = s(np(properN(nam)), vp(v(den), np(n(thu_vien)))) .
```

- Từ câu "Nam rất thích đọc sách ở thư viện."

➔ Nam đọc sách ở thư viện.

```
?- parse_tree([nam,doc,sach,o,thu,vien], Tree).

Tree = s(np(properN(nam)), vp(v(doc), np(n(sach), pp(p(o),
np(n(thu_vien))))))) .
```

➔ Nam rất thích đọc sách.

```
?- parse_tree([nam,rat,thich,doc,sach], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(rat)), v(thich), v(doc),
np(n(sach)))) .
```

➔ Nam thích đọc sách ở thư viện.

```
?- parse_tree([nam,thich,doc,sach,o,thu,vien], Tree).

false.
```

➔ Nam thích đọc sách.

```
?- parse_tree([nam,thich,doc,sach], Tree).

false.
```

➔ Nam rất thích sách.

```
?- parse_tree([nam,rat,thich,sach], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(rat)), v(thich), np(n(sach))))
```

➔ Nam thích sách.

```
?- parse_tree([nam,thich,sach], Tree).

Tree = s(np(properN(nam)), vp(v(thich), np(n(sach))))
```

- Từ câu "Nhà của Nam ở gần trường."

➔ Nam ở gần trường.

```
?- parse_tree([nam,o,gan,truong], Tree).

Tree = s(np(properN(nam)), pp(p(o), np(adjp(adj(gan)), n(truong)))) .
```

➔ Nam ở trường.

```
?- parse_tree([nam,o,truong], Tree).
Tree = s(np(properN(nam)), pp(p(o), np(n(truong))))
```

- Từ câu "Nam mới mua mấy cuốn sách mới."

➔ Nam mới mua mấy cuốn sách.

```
?- parse_tree([nam,moi,mua,may,cuon,sach], Tree).

false.
```

➔ Nam mua mấy cuốn sách mới.

```
?- parse_tree([nam,mua,may,cuon,sach,moi], Tree).
```

Tree = s(np(properN(nam)), vp(v(mua), np(det(may), n(cuon), n(sach), adjp(adj(moi))))) .

- Từ câu "Nam tặng Lan một cuốn sách rất hay."

➔ Nam tặng lan một cuốn sách.

?- parse_tree([nam,tang,lan,mot,cuon,sach], Tree).

false.

iii. Các câu phái sinh kết hợp với các câu khác

➔ Trường của Nam ở gần thư viện.

?- parse_tree([truong,cua,nam,o,gan,thu,vien], Tree).

Tree = s(np(n(truong), pp(p(cua), np(properN(nam)))), pp(p(o), np(adjp(adj(gan)), n(thu_vien)))) .

➔ Nam mới mua mấy cuốn sách rất hay.

?- parse_tree([nam,moi,mua,may,cuon,sach,rat,hay], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(moi)), v(mua), np(det(may), n(cuon), n(sach), advp(adv(rat)), adjp(adj(hay))))) .

➔ Nam tặng Lan một cuốn sách rất mới.

```
?- parse_tree([nam,tang,lan,mot,cuon,sach,rat,moi], Tree).

Tree = s(np(properN(nam)), vp(v(tang), np(properN(lan))), np(det(mot),
n(cuon), n(sach), advp(adv(rat)), adjp(adj(moi)))) .
```

➔ Nam tặng Lan một cuốn sách mới.

```
?- parse_tree([nam,tang,lan,mot,cuon,sach,moi], Tree).

Tree = s(np(properN(nam)), vp(v(tang), np(properN(lan))), np(det(mot),
n(cuon), n(sach), adjp(adj(moi)))) .
```

➔ Nam tặng Lan mấy cuốn sách mới.

```
?- parse_tree([nam,tang,lan,may,cuon,sach,moi], Tree).

Tree = s(np(properN(nam)), vp(v(tang), np(properN(lan))), np(det(may),
n(cuon), n(sach), adjp(adj(moi)))) .
```

➔ Nam tặng Lan mấy cuốn sách rất mới.

```
?- parse_tree([nam,tang,lan,may,cuon,sach,rat,moi], Tree).

Tree = s(np(properN(nam)), vp(v(tang), np(properN(lan))), np(det(may),
n(cuon), n(sach), advp(adv(rat)), adjp(adj(moi)))) .
```

➔ Nam thường đọc sách ở thư viện.

```
?- parse_tree([nam,thuong,doc,sach,o,thu,vien], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(thuong)), v(doc), np(n(sach),
pp(p(o), np(n(thu_vien))))))) .
```

➔ Nam thường đọc sách ở nhà của Lan.

```
?- parse_tree([nam,thuong,doc,sach,o,nha,cua,lan], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(thuong)), v(doc), np(n(sach),
pp(p(o), np(n(nha), pp(p(cua), np(properN(lan)))))))))
```

➔ Nam thường mua sách ở thư viện.

```
?- parse_tree([nam,thuong,mua,sach,o,thu,vien], Tree).
false.
```

➔ Lan thích cuốn sách Nam tặng.

```
?- parse_tree([lan,thich,cuon,sach,nam,tang], Tree).
false.
```

➔ Lan đến thư viện gần trường của Nam.

```
?- parse_tree([lan,den,thu,vien,gan,truong,cua,nam], Tree).
```

```
false.
```

➔ Nam mua sách ở trường gần thư viện.

```
?- parse_tree([nam,mua,sach,o,truong,gan,thu,vien], Tree).

Tree = s(np(properN(nam)), vp(v(mua), np(n(sach), pp(p(o),
np(n(truong))))), np(adjp(adj(gan)), n(thu_vien)))
```

➔ Nam thường đến trường đọc sách.

```
?- parse_tree([nam,thuong,den,truong,doc,sach], Tree).

false.
```

➔ Thư viện gần nhà của Nam.

```
parse_tree([thu,vien,gan,nha,cua,nam], Tree).

false.
```

➔ Nam thường đọc sách ở trường.

```
?- parse_tree([nam,thuong,doc,sach,o,truong], Tree).
```

Tree = s(np(properN(nam)), vp(advp(adv(thuong)), v(doc), np(n(sach), pp(p(o), np(n(truong))))))) ;

Tree = s(np(properN(nam)), vp(advp(adv(thuong)), v(doc), np(n(sach))), pp(p(o), np(n(truong)))) ;

➔ Lan rất thích mấy cuốn sách của nam.

?- parse_tree([lan,rat,thich,may,cuon,sach,cua,nam], Tree).

false.

➔ Nam đến thư viện tặng Lan một cuốn sách.

?- parse_tree([nam,den,thu,vien,tang,lan,mot,cuon,sach],Tree).

False.

➔ Nam thường mua sách ở thư viện.

?- parse_tree([nam,thuong,mua,sach,o,thu,vien], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(thuong)), v(mua), np(n(sach), pp(p(o), np(n(thu_vien)))))) ;

Tree = s(np(properN(nam)), vp(advp(adv(thuong)), v(mua), np(n(sach))), pp(p(o), np(n(thu_vien)))) ;

➔ Nam thích mua sách ở trường.

```
?- parse_tree([nam,thich,mua,sach,o,truong], Tree).

false.
```

➔ Nam thích đọc sách ở trường.

```
?- parse_tree([nam,thich,doc,sach,o,truong], Tree).

false.
```

➔ Nam rất thích Lan

```
?- parse_tree([nam,rat,thich,lan], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(rat)), v(thich),
np(properN(lan)))) ;
```

➔ Nam mới mua cuốn sách ở thư viện.

```
?- parse_tree([nam,moi,mua,cuon,sach,o,thu,vien], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(moi)), v(mua), np(n(cuon),
n(sach))), pp(p(o), np(n(thu_vien)))) ;

Tree = s(np(properN(nam)), vp(advp(adv(moi)), v(mua), np(n(cuon))),
np(n(sach), pp(p(o), np(n(thu_vien))))) ;
```

➔ Nam mới mua mấy cuốn sách mới ở thư viện

```
?- parse_tree([nam,moi,mua,may,cuon,sach,moi,o,thu,vien],Tree).

Tree = s(np(properN(nam)), vp(advp(adv(moi)), v(mua), np(det(may),
n(cuon), n(sach), adjp(adj(moi)))), pp(p(o), np(n(thu_vien)))) ;
```

➔ Nam mới mua mấy cuốn sách rất hay ở thư viện

```
?- parse_tree([nam,moi,mua,may,cuon,sach,rat,hay,o,thu,vien], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(moi)), v(mua), np(det(may),
n(cuon), n(sach), advp(adv(rat)), adjp(adj(hay)))), pp(p(o),
np(n(thu_vien)))) ;
```

➔ Nam rất thích đọc sách ở nhà Lan.

```
?- parse_tree([nam,rat,thich,doc,sach,o,nha,lan], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(rat)), v(thich), v(doc),
np(n(sach), pp(p(o), np(n(nha))))), np(properN(lan))) ;
```

➔ Nam đến trường đọc sách.

```
?- parse_tree([nam,den,truong,doc,sach], Tree).

false.
```

➔ Nam thường tặng sách ở trường.

```
?- parse_tree([nam,thuong,tang,sach,o,truong], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(thuong)), v(tang), np(n(sach),
pp(p(o), np(n(truong))))))) ;

Tree = s(np(properN(nam)), vp(advp(adv(thuong)), v(tang), np(n(sach))),
pp(p(o), np(n(truong)))) ;
```

➔ Nam rất thích mua sách ở thư viện.

```
?- parse_tree([nam,rat,thich,mua,sach,o,thu,vien], Tree).

Tree = s(np(properN(nam)), vp(advp(adv(rat)), v(thich), v(mua),
np(n(sach), pp(p(o), np(n(thu_vien)))))) ;

Tree = s(np(properN(nam)), vp(advp(adv(rat)), v(thich), v(mua),
np(n(sach))), pp(p(o), np(n(thu_vien)))) ;
```

➔ Nam rất thích ở thư viện.

```
?- parse_tree([nam,rat,o,thu,vien], Tree).

false.
```

➔ Trường của Nam thường mua sách ở thư viện.

```
?- parse_tree([truong,cua,nam,thuogng,mua,sach,o,thu,vien],Tree).

false.
```

VI. Nhận xét và đánh giá

- **Ưu điểm của Phân tích Cú pháp Dựa trên Luật với DCG**
  - **Biểu diễn cú pháp rõ ràng và dễ hiểu**: Văn phạm DCG cho phép mô tả cú pháp của một ngôn ngữ bằng các quy tắc cụ thể và dễ hiểu. Các quy tắc cú pháp được viết dưới dạng mệnh đề logic, dễ dàng biểu diễn các cấu trúc phức tạp và quan hệ cú pháp giữa các thành phần câu.
  - **Khả năng xử lý ngữ nghĩa**: Với DCG, ta có thể kết hợp các biểu diễn ngữ nghĩa trong các quy tắc cú pháp, giúp xây dựng các hệ thống có thể hiểu không chỉ cấu trúc mà còn ý nghĩa của câu. Điều này là một điểm mạnh trong các hệ thống cần xử lý ngữ nghĩa như hệ thống hỏi-đáp, trợ lý ảo, hay dịch máy.
  - **Tính chính xác và khả năng kiểm soát**: Phương pháp dựa trên luật không phụ thuộc vào các thống kê hay xác suất, nên kết quả phân tích cú pháp ít bị ảnh hưởng bởi dữ liệu huấn luyện. Các nhà phát triển có thể kiểm soát chính xác các quy tắc cú pháp, dễ dàng điều chỉnh để phù hợp với yêu cầu của ứng dụng.

- **Nhược điểm và Hạn chế của Phân tích Cú pháp Dựa trên Luật với DCG**
  - **Đòi hỏi công sức phát triển thủ công**: Để xây dựng một hệ thống phân tích cú pháp chính xác và đầy đủ, cần phải định nghĩa thủ công nhiều quy tắc cú pháp, đặc biệt đối với ngôn ngữ có cấu trúc phức tạp. Công việc này đòi hỏi kiến thức sâu về ngôn ngữ học và khá tốn thời gian.
  - **Thiếu khả năng khái quát**: Vì phương pháp dựa trên các quy tắc tường minh, nên văn phạm DCG thường gặp khó khăn trong việc phân tích các cấu trúc câu lạ hoặc không phổ biến, cũng như các ngữ pháp mới phát sinh trong ngôn ngữ tự nhiên. Điều này làm giảm tính linh hoạt trong môi trường ngôn ngữ luôn thay đổi.

- **Khả năng mở rộng hạn chế với ngôn ngữ phức tạp**: DCG hoạt động tốt với những ngôn ngữ có cấu trúc ngữ pháp chặt chẽ, nhưng với các ngôn ngữ có cấu trúc phức tạp hoặc không cố định như tiếng Anh hoặc tiếng Việt, việc xây dựng và duy trì bộ quy tắc có thể trở nên phức tạp và tốn nhiều công sức.
- **Hiệu suất tính toán**: Khi phân tích các câu dài hoặc phức tạp, phương pháp này có thể trở nên chậm và đòi hỏi nhiều tài nguyên tính toán. Các quy tắc cần được kiểm tra tuần tự, làm cho hiệu suất không tối ưu khi so với các phương pháp phân tích cú pháp hiện đại dựa trên học máy hoặc xác suất.

- **Ứng dụng và Triển vọng của DCG trong Xử lý Ngôn ngữ Tự nhiên**
  - **Các ứng dụng cần tính chính xác cao**: DCG thích hợp cho các ứng dụng đặc thù như xử lý ngôn ngữ trong hệ thống pháp lý, y tế, hoặc những lĩnh vực yêu cầu chính xác cao. Vì các quy tắc được xây dựng thủ công, các hệ thống có thể được tinh chỉnh để phản ánh chính xác ngữ pháp của các lĩnh vực này.
  - **Kết hợp với các phương pháp học máy**: Để khắc phục các hạn chế về khả năng khái quát và hiệu suất, DCG có thể được kết hợp với các phương pháp học máy hiện đại. Các mô hình học sâu có thể xử lý các cấu trúc không phổ biến, trong khi DCG cung cấp các ràng buộc cú pháp chặt chẽ, cải thiện độ chính xác và độ tin cậy của hệ thống.
  - **Giáo dục và nghiên cứu ngôn ngữ học tính toán**: DCG vẫn là công cụ hữu ích trong nghiên cứu và giảng dạy ngôn ngữ học tính toán, vì nó cung cấp cái nhìn trực quan về cấu trúc cú pháp và quy tắc ngữ pháp, giúp các sinh viên và nhà nghiên cứu hiểu rõ hơn về cấu trúc của ngôn ngữ tự nhiên.

VII. Kết luận

- Phân tích cú pháp là một lĩnh vực quan trọng trong khoa học máy tính, không chỉ trong phát triển ngôn ngữ lập trình mà còn trong xử lý ngôn ngữ tự nhiên. Qua các giai đoạn lịch sử, từ những năm 1950 với sự ra đời của ngôn ngữ lập trình đầu tiên cho đến sự phát triển mạnh mẽ của học sâu trong những năm gần đây, chúng ta thấy rõ sự tiến bộ vượt bậc trong các phương pháp và công cụ phân tích cú pháp.

- Việc áp dụng các quy tắc và mô hình ngữ pháp giúp chúng ta xây dựng được các công cụ phân tích cú pháp hiệu quả, phục vụ cho nhiều mục đích khác nhau. Phân tích cú pháp không chỉ hỗ trợ việc biên dịch ngôn ngữ máy tính mà còn mở rộng ra lĩnh vực xử lý ngôn ngữ tự nhiên, giúp máy tính hiểu và xử lý văn bản một cách chính xác hơn.
- Với sự phát triển không ngừng của công nghệ, các phương pháp và công cụ phân tích cú pháp sẽ tiếp tục được cải tiến, mở ra nhiều cơ hội và thách thức mới trong việc phát triển các ứng dụng thông minh. Từ việc sử dụng ngữ pháp dựa trên luật đến áp dụng các mô hình học sâu, nghiên cứu và ứng dụng phân tích cú pháp sẽ góp phần nâng cao khả năng tương tác giữa con người và máy tính trong tương lai.

Tài liệu tham khảo

- **Khái niệm về phân tích cú pháp**
  - ➢ https://en.wikipedia.org/wiki/Parsing
- **Chomsky hierarchy**
  - ➢ https://en.wikipedia.org/wiki/Chomsky_hierarchy
  - ➢ https://thebasics.guide/chomsky-hierarchy/
- **LR(k)**
  - ➢ https://blog.reverberate.org/2013/09/ll-and-lr-in-context-why-parsing-tools.html
- **CYK (Cocke-Younger-Kasami) Algorithm**
  - ➢ https://iscl-parsing2020.github.io/slides/earley.pdf
  - ➢ https://www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/
- **Earley Parser**
  - ➢ https://en.wikipedia.org/wiki/Earley_parser
- **Lex && Yacc**
  - ➢ https://en.wikipedia.org/wiki/Lex_(lexical_analysis_generator)
  - ➢ https://nlp.stanford.edu/software/lex-parser.html
- **ngôn ngữ tự nhiên (NLP)**
  - ➢ https://www.analyticsvidhya.com/blog/2023/09/advanced-natural-language-processing-nlp/
- **CRFs**
  - ➢ https://en.wikipedia.org/wiki/Conditional_random_field