TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

TIỂU LUẬN

Môn: SEMINAR CHUYÊN ĐỀ - LỚP: DKP1201 (Nhóm 02)
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: Trần Quang Trường
Mã số sinh viên: 3121410544
Giảng viên hướng dẫn: PGS. TS Nguyễn Tuấn Đăng

Thành phố Hồ Chí Minh, tháng 11 năm 2024

LỜI CẢM ƠN

Lời đầu tiên, em xin gửi lời cảm ơn chân thành đến các thầy cô trong khoa Công nghệ thông tin, trường Đại học Sài Gòn, những người đã tận tâm giảng dạy và truyền đạt kiến thức cũng như phương pháp trong suốt những năm qua. Những điều này đã trở thành nền tảng vững chắc và hành trang quý báu, giúp em tự tin bước vào con đường sự nghiệp trong tương lai.

Đặc biệt, em xin bày tỏ lòng biết ơn sâu sắc đến thầy Nguyễn Tuấn Đăng, người đã tận tình hỗ trợ, định hướng và đưa ra kế hoạch hướng dẫn để em hoàn thành bài tiểu luận môn Seminar một cách hiệu quả nhất.

Em cũng xin cảm ơn gia đình và bạn bè đã luôn động viên, chia sẻ, hỗ trợ nhiệt tình và đóng góp nhiều ý kiến quý báu giúp em hoàn thành đồ án ngành.

Trong quá trình thực hiện bài tiểu luận, do kinh nghiệm thực tế còn hạn chế và chỉ dựa vào lý thuyết đã học cùng với thời gian có hạn, bài viết chắc chắn không tránh khỏi sai sót. Em mong nhận được sự góp ý và nhận xét từ các thầy cô để có thể hoàn thiện kiến thức, tích lũy thêm kinh nghiệm bổ ích và áp dụng vào thực tiễn một cách hiệu quả trong tương lai.

Em xin chân thành cảm ơn!

NHẬN XÉT CỦA GIẢNG VIÊN

……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……...
……..
……..

MỤC LỤC

I. Giới thiệu

1. Mục tiêu

Mục tiêu của đề tài tiểu luận Xây dựng công cụ phân tích cú pháp tự động dựa trên văn phạm DCG (Definite Clause Grammar), nhằm phân tích các câu có cấu trúc cú pháp hợp lệ và có ý nghĩa từ một tập ngữ liệu cho đã cho trước. Bộ phân tích cú pháp này sẽ có khả năng nhận diện, xử lý và kiểm tra các quy tắc cú pháp để xác định xem câu có tuân theo ngữ pháp đã định nghĩa hay không.

2. Phạm vi

Phạm vi của đề tài bao gồm việc nghiên cứu, thiết kế và triển khai một công cụ phân tích cú pháp tự động sử dụng ngôn ngữ lập trình Prolog để thực hiện việc phân tích câu dựa trên văn phạm DCG. Đề tài sẽ áp dụng cho một tập hợp ngữ liệu có giới hạn, bao gồm các câu đơn giản đến phức tạp trong tiếng Việt hoặc tiếng Anh. Ngoài ra, công cụ sẽ được thiết kế với mục tiêu có thể mở rộng để xử lý các ngữ liệu lớn và phức tạp hơn trong tương lai.

3. Lý do

Lý do lựa chọn đề tài này là do việc phân tích cú pháp đóng vai trò quan trọng trong các ứng dụng xử lý ngôn ngữ tự nhiên (NLP), như các hệ thống dịch tự động, trợ lý ảo và các công cụ tìm kiếm thông minh. Văn phạm DCG là một phương pháp mạnh mẽ để xây dựng các công cụ phân tích cú pháp vì tính linh hoạt và khả năng mở rộng. Việc xây dựng công cụ phân tích cú pháp tự động sẽ giúp hỗ trợ việc nghiên cứu và phát triển trong lĩnh vực NLP và giúp tiết kiệm thời gian trong việc phân tích cú pháp thủ công.

4. Đối tượng nghiên cứu

➔ Ngôn ngữ Prolog.
➔ Văn phạm DCG, cây cú pháp
➔ Phương pháp phân tích cú pháp một câu tiếng Việt.

II. Nội dung

1. Lịch sử nghiên cứu phân tích cú pháp trong khóa học máy tính

1.1 Khóa học máy tính

Khoa học máy tính (Computer science) là ngành nghiên cứu các cơ sở lý thuyết về thông tin và tính toán cùng sự thực hiện và ứng dụng của chúng trong các hệ thống máy tính. Khoa học máy tính là cách tiếp cận khoa học và thực tiễn để tính toán và các ứng dụng của nó và nghiên cứu có hệ thống về tính khả thi, cấu trúc, biểu hiện và cơ giới hóa các thủ tục (hoặc các thuật toán) cơ bản làm cơ sở cho việc thu thập, đại diện, xử lý, lưu trữ, truyền thông và truy cập thông tin.

Khoa học máy tính phát triển từ giữa thế kỷ 20, dựa trên các nền tảng từ toán học, logic và điện tử học:

- Năm 1936: Alan Turing giới thiệu "Máy Turing" trong bài báo "On Computable Numbers," một mô hình trừu tượng về máy tính và thuật toán.
- Thập niên 1940: John von Neumann phát triển kiến trúc máy tính hiện đại, gọi là "kiến trúc Von Neumann," sử dụng bộ nhớ để lưu trữ cả dữ liệu và chương trình.
- Thập niên 1950: Các máy tính điện tử đầu tiên xuất hiện như ENIAC và UNIVAC.
- Thập niên 1960: Sự phát triển của ngôn ngữ lập trình như FORTRAN, LISP và hệ điều hành đầu tiên như Unix.

1.2 Phân tích cú pháp

Phân tích cú pháp (Parsing) là quá trình phân tích cấu trúc của một chuỗi ký tự, trong ngôn ngữ tự nhiên, trong ngôn ngữ máy tính hoặc cấu trúc dữ liệu theo các quy tắc của một ngôn ngữ hình thức. Quá trình này giúp xác

định cách các thành phần của chuỗi liên kết với nhau và tạo ra một cấu trúc biểu diễn hợp lệ.

Phân tích cú pháp được sử dụng rộng rãi trong lập trình để dịch mã nguồn từ các ngôn ngữ lập trình cấp cao thành mã máy mà máy tính có thể thực thi. Các hệ thống dịch ngôn ngữ như trình biên dịch (compiler) dựa vào phân tích cú pháp để xác minh mã nguồn tuântheo cú pháp và quy tắc của ngôn ngữ đó.

- Thập niên 1950: Các lý thuyết cơ bản về phân tích cú pháp bắt đầu xuất hiện cùng với sự phát triển của ngôn ngữ lập trình.

- Năm 1956: Noam Chomsky phát minh ra ngữ pháp phi ngữ cảnh, đóng vai trò quan trọng trong lý thuyết phân tích cú pháp.

- Năm 1960: John Backus và Peter Naur phát triển ký pháp Backus-Naur (BNF) để mô tả cú pháp của ngôn ngữ lập trình ALGOL.

- Năm 1965: Donald Knuth giới thiệu phân tích cú pháp LALR (Look-Ahead LR) trong bài báo "On the Translation of Languages from Left to Right," một phương pháp phân tích cú pháp hiệu quả cho các ngôn ngữ lập trình phức tạp

1.3 Phân tích cú pháp trong khóa học máy tính

Phân tích cú pháp trong khoa học máy tính là một quá trình quan trọng trong việc thiết kế và xây dựng các trình biên dịch và thông dịch. Trong khoa học máy tính, nó không chỉ là một công cụ cho việc dịch mã lệnh, mà còn là công cụ giúp xử lý ngôn ngữ tự nhiên và các dạng dữ liệu có cấu trúc khác.

1.4 Lịch sử nghiên cứu phân tích cú pháp trong khoa học máy tính:

Thập kỷ 1950 – 1960:

Những năm 1950 đánh dấu 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. Trong thời gian này, phân tích cú pháp trong khoa học máy tính được phát triển nhằm hỗ trợ việc xây dựng các trình biên dịch (compiler), giúp hiểu cú pháp của mã nguồn để chuyển đổi nó thành mã máy tính có thể thực thi.

Năm 1956, Noam Chomsky, một nhà ngôn ngữ học nổi tiếng, đã đóng góp quan trọng với lý thuyết ngữ pháp phi ngữ cảnh. Vào cuối thập kỷ 1950, ông phát triển hệ phân cấp Chomsky (Chomsky hierarchy), phân loại các ngữ pháp thành bốn loại:

- **Ngữ pháp không giới hạn (Type 0)**: Mô tả bất kỳ ngôn ngữ nào mà một máy Turing có thể nhận diện.
- **Ngữ pháp ngữ cảnh (Type 1)**: Chỉ cho phép thay đổi các ký hiệu không đầu cuối trong một ngữ cảnh cụ thể.
- **Ngữ pháp phi ngữ cảnh (Type 2)**: Được sử dụng rộng rãi trong các ngôn ngữ lập trình, cho phép thay thế các ký hiệu không đầu cuối mà không cần quan tâm đến ngữ cảnh.
- **Ngữ pháp chính quy (Type 3)**: Đơn giản nhất, dùng cho các ngôn ngữ mà máy trạng thái hữu hạn có thể xử lý, như các mẫu biểu thức chính quy.

Cũng vào cuối thập kỷ 1950, John Backus và Peter Naur đã giới thiệu Backus-Naur Form (BNF). Đây là một cách biểu diễn cú pháp ngôn ngữ lập trình một cách hình thức, giúp định nghĩa rõ ràng cách các thành phần trong một ngôn ngữ lập trình có thể được sắp xếp và kết hợp. BNF trở thành bước tiến quan trọng trong sự phát triển của ngôn ngữ lập trình ALGOL, một trong những ngôn ngữ có ảnh hưởng lớn trong lịch sử.

Thập kỷ 1960 – 1970:

Trong những năm 1960, nhu cầu về phân tích cú pháp phức tạp hơn tăng cao do sự xuất hiện của các ngôn ngữ lập trình mới và các hệ thống phần mềm lớn. Để đáp ứng nhu cầu này, nhiều phương pháp và thuật toán phân tích cú pháp tiên tiến đã được phát triển.

- 1960: John Backus và Peter Naur đã phát triển ký pháp Backus-Naur Form (BNF) để mô tả cú pháp của ngôn ngữ lập trình ALGOL, giúp định nghĩa và chuẩn hóa cú pháp ngôn ngữ lập trình một cách rõ ràng.
- 1965: Donald Knuth, một nhà khoa học máy tính nổi tiếng, đã giới thiệu phân tích cú pháp LR (Left-to-right, Rightmost derivation), còn gọi là phân tích cú pháp từ dưới lên (bottom-up parsing). LR parsing đọc và phân tích mã từ trái sang phải, tìm kiếm sự dẫn xuất từ phải sang trái. Phương pháp này được sử dụng rộng rãi nhờ khả năng phân tích một phạm vi lớn các ngữ pháp phi ngữ cảnh với hiệu suất cao.
- 1968: Alfred Aho và Jeffrey Ullman đã phát triển phương pháp LL parsing (Left-to-right, Leftmost derivation), một phương pháp phân tích cú pháp từ trên xuống (top-down parsing). LL parser cũng phân tích mã từ trái sang phải nhưng thực hiện dẫn xuất từ trái nhất (leftmost derivation). LL parsing dễ cài đặt hơn LR parsing nhưng có khả năng xử lý ít ngữ pháp phi ngữ cảnh hơn.
- Các thuật toán khác:
    - Thuật toán CYK (Cocke-Younger-Kasami): Phương pháp phân tích cú pháp 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): Một thuật toán có khả năng phân tích các ngữ pháp phi ngữ cảnh chung mà không cần biến đổi ngữ pháp.

Những phát triển này đã đặt nền móng cho các trình biên dịch hiện đại và các công cụ phân tích cú pháp hiệu quả.

Thập kỷ 1970 – 1990:

Trong giai đoạn từ thập kỷ 1970 đến 1990, phân tích cú pháp đã phát triển mạnh mẽ và trở thành một phần thiết yếu trong thiết kế và phát triển trình biên dịch cũng như xử lý ngôn ngữ tự nhiên. Dưới đây là những sự kiện và công cụ nổi bật:

- 1975: Sự ra đời của Yacc (Yet Another Compiler-Compiler) tại AT&T Bell Labs giúp phát triển trình biên dịch dễ dàng hơn. Yacc cho phép các nhà phát triển định nghĩa cú pháp của ngôn ngữ và tự động sinh mã cho trình biên dịch.
- Lex: Được sử dụng cùng với Yacc để phân tích từ vựng của mã nguồn. Lex giúp nhận diện các mẫu từ vựng trước khi được xử lý bởi Yacc.
- Bison: Một phiên bản mã nguồn mở của Yacc, được phát triển sau này để cung cấp các tính năng tương tự nhưng với khả năng mở rộng và tự do sử dụng trong các dự án mã nguồn mở.
- Phân tích cú pháp LALR (Lookahead LR): Một biến thể của phân tích cú pháp LR, giúp xử lý cú pháp hiệu quả hơn cho nhiều ngôn ngữ lập trình. LALR đã trở thành tiêu chuẩn trong các công cụ trình biên dịch vào thời kỳ này.
- Thập niên 1980: Các công cụ như Yacc tiếp tục hỗ trợ các lập trình viên phát triển parser tùy chỉnh. Thời kỳ này cũng chứng kiến sự phát triển của lĩnh vực xử lý ngôn ngữ tự nhiên (NLP), với việc áp dụng các kỹ thuật phân tích cú pháp vào việc phân tích và hiểu ngôn ngữ tự nhiên.
- Phân tích cú pháp biểu đồ (Chart Parsing): Một phương pháp phổ biến trong NLP vào thập kỷ 1980, giúp tối ưu hóa việc phân tích các câu phức tạp.
- Thuật toán Earley (1970): Là một thuật toán phân tích cú pháp phi ngữ cảnh hiệu quả, có khả năng xử lý bất kỳ loại ngữ pháp phi ngữ cảnh nào. Điều này đặc biệt hữu ích cho các ngôn ngữ tự nhiên với cấu trúc câu phức tạp.
- Thuật toán CYK (Cocke-Younger-Kasami): Được phát minh vào thập niên 1960 nhưng tiếp tục được sử dụng trong thập niên 1970–1980. Nó là một phương pháp phân tích cú pháp từ dưới lên, hiệu quả cho các ngữ pháp phi ngữ cảnh đã được chuyển đổi về dạng chuẩn Chomsky (Chomsky Normal Form - CNF).

Những công cụ và thuật toán trong giai đoạn này đã đặt nền tảng cho rất nhiều hệ thống trình biên dịch và hệ thống xử lý ngôn ngữ tự nhiên hiện đại.

Thập kỷ 2000 đến nay:

Học máy và AI trong phân tích cú pháp: Sự phát triển mạnh mẽ của học máy và trí tuệ nhân tạo đã cải thiện đáng kể khả năng phân tích cú pháp, đặc biệt trong xử lý ngôn ngữ tự nhiên (NLP). Các mạng nơ-ron hồi tiếp (RNN) và mô hình mã hóa – giải mã (encoder-decoder) giúp tạo ra công cụ phân tích cú pháp tự động.

Học sâu (Deep Learning) và NLP: Những năm 2010 chứng kiến sự bùng nổ của học sâu trong NLP. Các mô hình như **Transformer**, **BERT**, và **GPT** (do Google và OpenAI phát triển) đã cải thiện hiệu suất xử lý cú pháp văn bản với khả năng ngữ cảnh hóa cao. Các mô hình này sử dụng kiến trúc attention để học cấu trúc ngữ cảnh trong văn bản.

Ứng dụng của các mô hình hiện đại: Các công cụ như spaCy, AllenNLP, và các thư viện mã nguồn mở như **Hugging Face Transformers** đã giúp nhà phát triển tích hợp mô hình NLP tiên tiến mà không cần thực hiện các bước phân tích cú pháp truyền thống.

Phương pháp phân tích cú pháp truyền thống:

- **Phân tích cú pháp dựa trên cây (Parse Tree-based Parsing)**: Tạo cây cú pháp để biểu diễn cấu trúc câu, hiệu quả nhưng tốn tài nguyên cho ngữ pháp phức tạp.
- **Phương pháp dựa trên bảng (Table-driven Parsing)**: Sử dụng bảng phân tích cú pháp, bao gồm các kỹ thuật như LL Parsing (phân tích từ trên xuống) và LR Parsing (phân tích từ dưới lên). Các phương pháp này hiệu quả cho ngữ pháp rõ ràng nhưng khó áp dụng với ngữ pháp phức tạp.
- **Phân tích cú pháp dựa trên luật (Rule-based Parsing)**: Áp dụng các quy tắc ngữ pháp để phân tích cú pháp. Phương pháp này linh hoạt

và phù hợp với các ứng dụng NLP nhưng hiệu năng có thể thấp với tập dữ liệu lớn.

Sự phát triển công cụ phân tích cú pháp: Các công cụ như **Yacc, ANTLR, và Bison** tiếp tục được phát triển, tự động tạo ra bộ phân tích cú pháp từ mô tả ngữ pháp, giúp giảm thời gian phát triển trình biên dịch và thông dịch.

Từ năm 2000 đến nay, các phương pháp học máy, học sâu và sự phát triển của các công cụ NLP hiện đại đã giúp phân tích cú pháp đạt được những bước tiến lớn, không chỉ trong xử lý ngôn ngữ tự nhiên mà còn mở rộng ra nhiều lĩnh vực của trí tuệ nhân tạo.

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

2.1 Khái niệm phân tích cú pháp dựa trên luật và lập trình logic:

Phân tích cú pháp dựa trên luật là phương pháp sử dụng các quy tắc ngữ pháp (rules) để xác định và phân loại cấu trúc của một câu văn bản. Các quy tắc này được định nghĩa dưới dạng các mẫu ngữ pháp, cho phép phân tích các thành phần trong câu như cụm danh từ (NP), cụm động từ (VP), và các mệnh đề phụ thuộc. Cách tiếp cận dựa trên luật thường yêu cầu các nhà ngôn ngữ học và lập trình viên tạo ra các quy tắc cụ thể dựa trên ngữ pháp ngôn ngữ tự nhiên.

Lập trình logic là một phương pháp lập trình dựa trên các mệnh đề và quan hệ logic để giải quyết các vấn đề. Ngôn ngữ Prolog là một ví dụ tiêu biểu, trong đó các quy tắc ngữ pháp có thể được biểu diễn dưới dạng Definite Clause Grammar (DCG). DCG cho phép mô hình hóa các cấu trúc ngữ pháp phức tạp và giúp tạo ra cây phân tích cú pháp (parse tree) từ câu đầu vào.

2.2 Văn phạm DCG (Definite Clause Grammar) trong phân tích cú pháp:

Văn phạm DCG (Definite Clause Grammar) là một hình thức ngữ pháp được thiết kế đặc biệt cho ngôn ngữ Prolog, cho phép tạo ra các quy tắc ngữ pháp theo định dạng dễ hiểu và dễ quản lý. DCG giúp xác định cấu trúc câu trong ngôn ngữ bằng cách sử dụng các mẫu ngữ pháp đã được định nghĩa trước. Các quy tắc này được biểu diễn dưới dạng các mệnh đề Prolog, cho phép Prolog tự động kiểm tra và khớp các thành phần của câu với các quy tắc ngữ pháp.

DCG là một biến thể của văn phạm CFG (Context-Free Grammar), thường được sử dụng để mô tả ngữ pháp của các ngôn ngữ tự nhiên. DCG rất phổ biến trong lĩnh vực xử lý ngôn ngữ tự nhiên và lập trình logic.

Các thành phần trong DCG

Trong DCG, ngữ pháp được phân chia thành các thành phần cơ bản như sau:

- **Từ (Words)**: Các ký hiệu đầu cuối đại diện cho từ vựng, ví dụ: danh từ, động từ, v.v.
- **Cụm từ (Phrases)**: Các nhóm từ tạo thành các thành phần lớn hơn trong câu, ví dụ: cụm danh từ (NP), cụm động từ (VP), v.v.
- **Câu (Sentences)**: Sự kết hợp của các cụm từ theo các quy tắc ngữ pháp để tạo thành câu hoàn chỉnh.

Ví dụ minh họa văn phạm DCG đơn giản:

```
% Quy tắc chính
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP).

%---------------- Noun phrase ----------------------------
noun_phrase(np(N)) --> noun(N).
noun_phrase(np(PN))--> proper_noun(PN).
noun_phrase(np(N, PP)) --> noun(N), prep_phrase(PP).
noun_phrase(np(ADVP, N)) --> adverb_phrase(ADVP), noun(N).
```

```
%--------------- Verb phrase ---------------------------
verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP).

%--------------- Prep phrase ---------------------------
prep_phrase(pp(PP, NP)) --> prep(PP), noun_phrase(NP).

%---------------- Adverb phrase ----------------------------
adverb_phrase(advp(ADV)) --> adverb(ADV).
%---------------- Lexicon ----------------------------
proper_noun(propn(nam)) --> [nam].

noun(n(sach)) --> [sach].
noun(n(nha)) --> [nha].

verb(v(doc)) --> [doc].

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

adverb(adv(gan)) --> [gan].
```

Các quy tắc này giúp xác định cấu trúc câu đơn giản như "Nam đọc sách ở gần nhà" bằng cách phân tách các thành phần theo ngữ pháp.

2.3 Quá trình phân tích cú pháp bằng DCG:

Các bước triển khai phân tích cú pháp dựa trên luật trong Prolog:

1. **Tiền xử lý văn bản**: Quá trình tiền xử lý bao gồm việc tách từ (tokenization) và chuẩn hóa văn bản. Mục đích của bước này là đảm bảo rằng câu đầu vào có thể dễ dàng so khớp với các quy tắc DCG. Các ký hiệu như dấu câu và chữ in hoa sẽ được xử lý để giảm thiểu sai lệch trong phân tích cú pháp.
2. **Xây dựng quy tắc ngữ pháp với DCG**: Đây là bước quan trọng nhất, trong đó các quy tắc ngữ pháp cho từng loại cụm từ và thành phần ngữ pháp sẽ được định nghĩa. Các quy tắc này cần mô tả cách mà các từ và cụm từ có thể kết hợp với nhau để hình thành các câu hợp lệ. Việc sử

dụng DCG trong Prolog cho phép mô tả ngữ pháp một cách dễ dàng từ các câu đơn giản đến phức tạp.

3. **Thực thi phân tích cú pháp**: Sau khi đã định nghĩa các quy tắc, bước tiếp theo là sử dụng các câu truy vấn trong Prolog để thực thi phân tích cú pháp. Prolog sẽ áp dụng các quy tắc DCG đã định nghĩa và tìm kiếm cách phân tích hợp lý cho câu đầu vào.
4. **Sinh cây phân tích cú pháp**: Kết quả của quá trình phân tích cú pháp có thể được biểu diễn dưới dạng cây phân tích cú pháp, giúp trực quan hóa cách mà các thành phần câu kết hợp với nhau. Với cách tiếp cận lập trình logic, ta có thể yêu cầu Prolog trả về cấu trúc cây thể hiện mối quan hệ giữa các từ và cụm từ trong câu.

Ví dụ minh họa quá trình phân tích cú pháp một câu đơn giản:

- Sử dụng ví dụ DCG ở trên, phân tích câu "Nam đọc sách ở gần nhà":

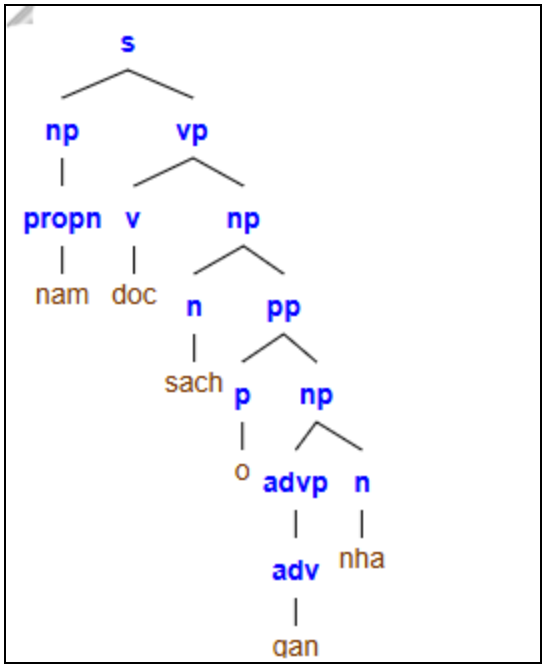
1.Tiền xử lý văn bản:

➔ Câu “Nam đọc sách ở gần nhà” được loại bỏ dấu câu và chuyển chữ hoa thành chữ thường để giảm thiểu sai lệch, sau đó tách câu thành danh sách từ: [nam, doc, sach,o, gan,nha].

➔ Với Quy ước là

- s: câu
- propn: danh từ riêng
- np: cụm danh từ
- n: danh từ
- vp: cụm động từ
- v: động từ
- pp: cụm giới từ
- p: giới từ
- advp: cụm trạng từ
- adv: trạng từ

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



2.Xây dựng quy tắc ngữ pháp với DCG trong Prolog:

```
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP).

noun_phrase(np(N)) --> noun(N).
noun_phrase(np(PN))--> proper_noun(PN).
noun_phrase(np(N, PP)) --> noun(N), prep_phrase(PP).
noun_phrase(np(ADVP, N)) --> adverb_phrase(ADVP), noun(N).

verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP).
prep_phrase(pp(PP, NP)) --> prep(PP), noun_phrase(NP).

adverb_phrase(advp(ADV)) --> adverb(ADV).

proper_noun(propn(nam)) --> [nam].
noun(n(sach)) --> [sach].
noun(n(nha)) --> [nha].
verb(v(doc)) --> [doc].
prep(p(o)) --> [o].
adverb(adv(gan)) --> [gan].
```

3.Thực thi phân tích cú pháp:

- Sau khi xây dựng xong các quy tắc, thêm hàm phân tích cú pháp vào

```
parse_tree(Sentence, Tree) :-
   phrase(sentence(Tree), Sentence).
```

- Thực thi phân tích cú pháp bằng truy vấn sau trong SWI-Prolog:

```
?- parse_tree([nam,doc,sach,o,gan,nha], Tree)
```

- Kết quả

```
Tree = s(np(propn(nam)), vp(v(doc), np(n(sach), pp(p(o),
np(advp(adv(gan)), n(nha))))))
```

Nếu câu không khớp với bất kỳ quy tắc nào, Prolog sẽ trả về false.

3. Xây dựng công cụ phân tích cú pháp dựa trên luận

3.1 Dữ liệu

3.1.1 Tập dữ liệu gốc

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

3.1.2 Các câu phái sinh từ các câu trong tập dữ liệu gốc

- **Từ câu "Nam thường đến thư viện"**
  ➔ Nam đến thư viện.

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

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

- **Từ câu "Nhà của Nam ở gần trường"**
  ➔ Nhà Nam ở gần trường.
  ➔ Nhà Nam ở trường.
  ➔ Nhà của Nam ở trường.
  ➔ Nhà của Nam gần trường.
  ➔ Nhà Nam gần trường.

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

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

➔ Nam tặng Lan cuốn sách hay.
➔ Nam tặng Lan sách rất hay.
➔ Nam tặng Lan sách hay.
➔ Nam tặng Lan sách.
➔ Nam tặng Lan.

3.1.3 Các câu phái sinh kết hợp từ các câu trong dữ liệu gốc

- **Từ câu "Nam thường đến thư viện " với những câu còn lại:**
  ➔ Nam thường đến trường.
  ➔ Nam đến thư viện ở gần trường.
  ➔ Nam thường đến thư viện ở gần nhà.
  ➔ Nam thích đến thư viện ở gần nhà.
  ➔ Nam thích đến thư viện gần nhà Lan.
  ➔ Nam rất thích đến thư viện ở gần trường.
  ➔ Nam đến nhà Lan.
  ➔ Nam thường đến thư viện đọc mấy cuốn sách mới.

- **Từ câu "Nam rất thích đọc sách ở thư viện " với những câu còn lại:**
  ➔ Nam rất thích đọc sách ở thư viện gần trường.
  ➔ Nam thích đọc sách ở thư viện của trường.
  ➔ Nam thường đọc sách ở thư viện gần trường.
  ➔ Nam rất thích đọc sách ở nhà Lan.
  ➔ Nam đọc sách ở thư viện ở gần nhà.
  ➔ Nam rất thích sách ở thư viện gần trường.
  ➔ Nam thích sách ở thư viện ở gần nhà.
  ➔ Nam thích Lan đọc sách.
  ➔ Nam thích Lan.
  ➔ Nam thích sách của Lan.
  ➔ Nam thích Lan tặng sách ở thư viện gần nhà.

- **Từ câu "Nhà của Nam ở gần trường" với những câu còn lại:**
  ➔ Nhà của Nam ở gần nhà Lan.
  ➔ Nhà Nam ở gần thư viện.

➔ Nhà Nam ở nhà Lan.
➔ Nhà của Nam ở trường.
➔ Nhà Nam rất gần trường.
➔ Nhà Nam gần Lan.
➔ Nhà của Nam mới mua tặng Lan.

- **Từ câu "Nam mới mua mấy cuốn sách mới" với những câu còn lại:**
  ➔ Nam mới mua mấy cuốn sách mới ở thư viện.
  ➔ Nam mới mua mấy cuốn sách mới của thư viện gần nhà Lan.
  ➔ Nam mới mua mấy cuốn sách mới của thư viện gần nhà.
  ➔ Nam mới mua mấy cuốn sách mới ở gần thư viện.
  ➔ Nam mua mấy cuốn sách mới ở thư viện.
  ➔ Nam mua mấy cuốn sách mới của thư viện.
  ➔ Nam mới mua sách mới.
  ➔ Nam mua sách mới ở gần thư viện.
  ➔ Nam mua sách mới ở trường.
  ➔ Nam mua sách mới của Lan.
  ➔ Nam mua sách mới tặng Lan.
  ➔ Nam mới mua nhà ở gần trường.

- **Từ câu "Nam tặng Lan một cuốn sách rất hay" với những câu còn lại:**
  ➔ Nam mới tặng Lan một cuốn sách rất hay.
  ➔ Nam tặng Lan mấy cuốn sách rất hay.
  ➔ Nam thường tặng Lan một cuốn sách rất hay.
  ➔ Nam thích tặng Lan một cuốn sách rất hay.
  ➔ Nam mới tặng Lan mấy cuốn sách hay.
  ➔ Nam mới tặng Lan một cuốn sách hay.
  ➔ Nam thích tặng Lan một cuốn sách hay.
  ➔ Nam mới tặng Lan một cuốn sách.
  ➔ Nam rất thích tặng Lan cuốn sách hay.
  ➔ Nam thích tặng Lan cuốn sách hay.
  ➔ Nam thích tặng Lan sách rất hay.

➔ Nam rất thích tặng Lan sách hay.
➔ Nam thích tặng Lan sách.
➔ Nam tặng thư viện một cuốn sách rất hay.

3.2 Văn phạm DCG (Definite Clause Grammar)

3.2.1 Văn phạm DCG của tất cả các câu

```
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP
VP
%---------- Noun phrase ----------
noun_phrase(np(N)) --> noun(N).  %NP -> N
noun_phrase(np(N1, N2)) --> noun(N1), noun(N2). % NP -> NN
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN
noun_phrase(np(N, PN)) --> noun(N), proper_noun(PN). % NP -> N, PN
noun_phrase(np(ADVP, N)) --> adverb_phrase(ADVP), noun(N). % NP
-> ADVP N
noun_phrase(np(N, PP)) --> noun(N), prep_phrase(PP). % NP -> N, P
noun_phrase(np(NM)) --> noun_modifier(NM). % NP -> NM
noun_phrase(np(DET, NM, ADJP)) --> determiner(DET),
noun_modifier(NM), adjective_phrase(ADJP). % NP --> DET, NM,
ADJP
noun_phrase(np(DET, NM)) --> determiner(DET), noun_modifier(NM).
% NP -> DET, NM
noun_phrase(np(N, ADJP)) --> noun(N), adjective_phrase(ADJP). %
NP->N, ADJP

%---------- Verb phrase ----------
verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP). % VP -> V, NP
verb_phrase(vp(V, VP)) --> verb(V), verb_phrase(VP). % VP -> V, VP
verb_phrase(vp(V, PP)) --> verb(V), prep_phrase(PP). % VP-> V, PP
verb_phrase(vp(ADVP, VP)) --> adverb_phrase(ADVP),
verb_phrase(VP). % VP -> ADVP, VP
verb_phrase(vp(ADVP, NP)) --> adverb_phrase(ADVP),
```

```
noun_phrase(NP). % VP -> ADVP, NP
verb_phrase(vp(V, NP, PP)) --> verb(V), noun_phrase(NP),
prep_phrase(PP). % VP -> NP, PP
verb_phrase(vp(V, NP1, NP2)) --> verb(V), noun_phrase(NP1),
noun_phrase(NP2). % VP -> V, NP1, NP2
verb_phrase(vp(V, NP1, NP2, PP)) --> verb(V), noun_phrase(NP1),
noun_phrase(NP2), prep_phrase(PP). % VP - V NP PP

%---------- Prep phrase ----------
prep_phrase(pp(PP, NP)) --> prep(PP), noun_phrase(NP). % PP -> P, NP

%---------- Adverb phrase ----------
adverb_phrase(advp(ADV)) --> adverb(ADV). % ADVP -> ADV

%---------- Noun modifier ----------
noun_modifier(nm(UN, N)) --> unit_noun(UN), noun(N). % NM -> UN,
N
%---------- Determiner ----------
determiner(det(Q)) --> quantity(Q). % DET -> NUM
%---------- Adjective phrase ----------
adjective_phrase(adjp(ADV, ADJ)) --> adverb(ADV), adjective(ADJ). %
ADJP - ADV, ADJ
adjective_phrase(adjp(ADJ)) --> adjective(ADJ). % ADJP -> ADJ
%---------- Lexicon ----------
proper_noun(propn(nam)) --> [nam].
proper_noun(propn(lan)) --> [lan].

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

verb(v(thich)) --> [thich].
```

```
verb(v(o)) --> [o].
verb(v(mua)) --> [mua].

verb(v(tang)) --> [tang].
verb(v(doc)) --> [doc].
verb(v(den)) --> [den].

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

adverb(adv(gan)) --> [gan].
adverb(adv(thuong)) --> [thuong].
adverb(adv(rat)) --> [rat].
adverb(adv(moi)) --> [moi].

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

unit_noun(un(cuon)) --> [cuon].

quantity(q(mot)) --> [mot].
quantity(q(may)) --> [may].

% Hàm phân tích cú pháp
parse_tree(Sentence, Tree) :-
   phrase(sentence(Tree), Sentence).
```

```
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP VP
%--------------- Noun phrase ----------------------------
noun_phrase(np(N)) --> noun(N).  %NP -> N
noun_phrase(np(N1, N2)) --> noun(N1), noun(N2). % NP -> NN
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN
noun_phrase(np(N, PN)) --> noun(N), proper_noun(PN). % NP -> N, PN
noun_phrase(np(ADVP, N)) --> adverb_phrase(ADVP), noun(N). % NP -> ADVP N
noun_phrase(np(N, PP)) --> noun(N), prep_phrase(PP). % NP -> N, P
noun_phrase(np(NM)) --> noun_modifier(NM). % NP -> NM
noun_phrase(np(DET, NM, ADJP)) --> determiner(DET), noun_modifier(NM), adjective_phrase(ADJP). % NP --> DET, NM, ADJP
noun_phrase(np(DET, NM)) --> determiner(DET), noun_modifier(NM). % NP -> DET, NM
noun_phrase(np(N, ADJP)) --> noun(N), adjective_phrase(ADJP). % NP->N, ADJP

%--------------- Verb phrase ----------------------------
verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP). % VP -> V, NP
verb_phrase(vp(V, VP)) --> verb(V), verb_phrase(VP). % VP -> V, VP
verb_phrase(vp(V, PP)) --> verb(V), prep_phrase(PP). % VP-> V, PP
verb_phrase(vp(ADVP, VP)) --> adverb_phrase(ADVP), verb_phrase(VP). % VP -> ADVP, VP
verb_phrase(vp(ADVP, NP)) --> adverb_phrase(ADVP), noun_phrase(NP). % VP -> ADVP, NP
verb_phrase(vp(V, NP, PP)) --> verb(V), noun_phrase(NP), prep_phrase(PP). % VP -> NP, PP
verb_phrase(vp(V, NP1, NP2)) --> verb(V), noun_phrase(NP1), noun_phrase(NP2). % VP -> V, NP1, NP2
verb_phrase(vp(V, NP1, NP2, PP)) --> verb(V), noun_phrase(NP1), noun_phrase(NP2), prep_phrase(PP). % VP - V NP PP

%--------------- Prep phrase ----------------------------
prep_phrase(pp(PP, NP)) --> prep(PP), noun_phrase(NP). % PP -> P, NP

%--------------- Adverb phrase ----------------------------
adverb_phrase(advp(ADV)) --> adverb(ADV). % ADVP -> ADV

%--------------- Noun modifier ----------------------------
noun_modifier(nm(UN, N)) --> unit_noun(UN), noun(N). % NM -> UN, N
%--------------- Determiner ----------------------------
determiner(det(Q)) --> quantity(Q). % DET -> NUM
%--------------- Adjective phrase ----------------------------
adjective_phrase(adjp(ADV, ADJ)) --> adverb(ADV), adjective(ADJ). % ADJP - ADV, ADJ
adjective_phrase(adjp(ADJ)) --> adjective(ADJ). % ADJP -> ADJ
```

```
%---------------- Lexicon -----------------------------
proper_noun(propn(nam)) --> [nam].
proper_noun(propn(lan)) --> [lan].

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

verb(v(thich)) --> [thich].
verb(v(o)) --> [o].
verb(v(mua)) --> [mua].

verb(v(tang)) --> [tang].
verb(v(doc)) --> [doc].
verb(v(den)) --> [den].

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

adverb(adv(gan)) --> [gan].
adverb(adv(thuong)) --> [thuong].
adverb(adv(rat)) --> [rat].
adverb(adv(moi)) --> [moi].

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

unit_noun(un(cuon)) --> [cuon].

quantity(q(mot)) --> [mot].
quantity(q(may)) --> [may].


% Hàm phân tích cú pháp
parse_tree(Sentence, Tree) :-
    phrase(sentence(Tree), Sentence).
```

Hình 3.2.1 chụp văn phạm DCG của tất cả các câu

3.2.2 Văn phạm DCG của câu "Nam thường đến thư viện"

```
% Nam thuong den thu vien
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP,VP
```

```
%---------- Noun phrase ----------
noun_phrase(np(N)) --> noun(N).  %NP -> N
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN


%---------- Verb phrase ----------
verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP). % VP -> V NP
verb_phrase(vp(ADVP, VP)) --> adverb_phrase(ADVP),
verb_phrase(VP). % VP -> ADVP VP

%---------- Adverb phrase ----------
adverb_phrase(advp(ADV)) --> adverb(ADV). % ADVP -> ADV

%---------- Lexicon ----------
proper_noun(propn(nam)) --> [nam].

noun(n(thu_vien)) --> [thu, vien].

verb(v(den)) --> [den].

adverb(adv(thuong)) --> [thuong].

% Hàm phân tích cú pháp
parse_tree(Sentence, Tree) :-
   phrase(sentence(Tree), Sentence).
```

```
:- use_rendering(svgtree).
% Nam thuong den thu vien
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP,VP

%--------------- Noun phrase ----------------------------
noun_phrase(np(N)) --> noun(N).  %NP -> N
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN


%--------------- Verb phrase ----------------------------
verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP). % VP -> V NP
verb_phrase(vp(ADVP, VP)) --> adverb_phrase(ADVP), verb_phrase(VP). % VP -> ADVP VP

%---------------- Adverb phrase ----------------------------
adverb_phrase(advp(ADV)) --> adverb(ADV). % ADVP -> ADV

%---------------- Lexicon ----------------------------
proper_noun(propn(nam)) --> [nam].

noun(n(thu_vien)) --> [thu, vien].

verb(v(den)) --> [den].

adverb(adv(thuong)) --> [thuong].

% Hàm phân tích cú pháp
parse_tree(Sentence, Tree) :-
    phrase(sentence(Tree), Sentence).
```

Hình 3.2.2 chụp văn phạm DCG câu 'Nam thường đến thư viện'

3.2.3 Văn phạm DCG của câu "Nam rất thích đọc sách ở thư viện"

```
% Nam rat thich doc sach o thu vien
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP,VP

%---------------- Noun phrase ----------------------------
noun_phrase(np(N)) --> noun(N).  %NP -> N
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN
noun_phrase(np(N, PP)) --> noun(N), prep_phrase(PP). % NP -> N, PP

%---------------- Verb phrase ----------------------------
```

```
verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP). % VP -> V NP
verb_phrase(vp(V, VP)) --> verb(V), verb_phrase(VP). % VP -> V VP
verb_phrase(vp(ADVP, VP)) --> adverb_phrase(ADVP),
verb_phrase(VP). % VP -> ADVP VP

%---------- Adverb phrase ----------
adverb_phrase(advp(ADV)) --> adverb(ADV). % ADVP -> ADV

%---------- Prep phrase ----------
prep_phrase(pp(PP, NP)) --> prep(PP), noun_phrase(NP). % PP -> P NP

%---------- Lexicon ----------
proper_noun(propn(nam)) --> [nam].

noun(n(thu_vien)) --> [thu, vien].
noun(n(sach)) --> [sach].

verb(v(thich)) --> [thich].
verb(v(doc)) --> [doc].

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

adverb(adv(rat)) --> [rat].

% Hàm phân tích cú pháp
parse_tree(Sentence, Tree) :-
   phrase(sentence(Tree), Sentence).
```

```
% Nam rat thich doc sach o thu vien
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP,VP

%--------------- Noun phrase ----------------------------
noun_phrase(np(N)) --> noun(N).  %NP -> N
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN
noun_phrase(np(N, PP)) --> noun(N), prep_phrase(PP). % NP -> N, PP


%--------------- Verb phrase ----------------------------
verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP). % VP -> V NP
verb_phrase(vp(V, VP)) --> verb(V), verb_phrase(VP). % VP -> V VP
verb_phrase(vp(ADVP, VP)) --> adverb_phrase(ADVP), verb_phrase(VP). % VP -> ADVP VP

%---------------- Adverb phrase ----------------------------
adverb_phrase(advp(ADV)) --> adverb(ADV). % ADVP -> ADV

%--------------- Prep phrase ----------------------------
prep_phrase(pp(PP, NP)) --> prep(PP), noun_phrase(NP). % PP -> P NP

%---------------- Lexicon ----------------------------
proper_noun(propn(nam)) --> [nam].

noun(n(thu_vien)) --> [thu, vien].
noun(n(sach)) --> [sach].

verb(v(thich)) --> [thich].
verb(v(doc)) --> [doc].

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

adverb(adv(rat)) --> [rat].
```

Hình 3.2.3 chụp văn phạm DCG câu "Nam rất thích đọc sách ở thư viện"

3.2.4 Văn phạm DCG của câu "Nhà của Nam ở gần trường"

```
% Nha cua Nam o gan truong
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP,VP

%--------------- Noun phrase ----------------------------
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN
noun_phrase(np(ADVP, N)) --> adverb_phrase(ADVP), noun(N). % NP -> ADVP N
noun_phrase(np(N, PP)) --> noun(N), prep_phrase(PP). % NP -> N, PP
```

```
%---------- Verb phrase ----------
verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP). % VP -> V, NP

%---------- Adverb phrase ----------
adverb_phrase(advp(ADV)) --> adverb(ADV). % ADVP -> ADV

%---------- Prep phrase ----------
prep_phrase(pp(PP, NP)) --> prep(PP), noun_phrase(NP). % PP -> P, NP

%---------- Lexicon ----------
proper_noun(propn(nam)) --> [nam].

noun(n(nha)) --> [nha].
noun(n(truong)) --> [truong].

verb(v(o)) --> [o].

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

adverb(adv(gan)) --> [gan].

% Hàm phân tích cú pháp
parse_tree(Sentence, Tree) :-
   phrase(sentence(Tree), Sentence).
```

```
% Nha cua Nam o gan truong
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP,VP

%--------------- Noun phrase ----------------------------
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN
noun_phrase(np(ADVP, N)) --> adverb_phrase(ADVP), noun(N). % NP -> ADVP N
noun_phrase(np(N, PP)) --> noun(N), prep_phrase(PP). % NP -> N, PP


%--------------- Verb phrase ----------------------------
verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP). % VP -> V, NP

%---------------- Adverb phrase ----------------------------
adverb_phrase(advp(ADV)) --> adverb(ADV). % ADVP -> ADV

%--------------- Prep phrase ----------------------------
prep_phrase(pp(PP, NP)) --> prep(PP), noun_phrase(NP). % PP -> P, NP

%---------------- Lexicon ----------------------------
proper_noun(propn(nam)) --> [nam].

noun(n(nha)) --> [nha].
noun(n(truong)) --> [truong].

verb(v(o)) --> [o].

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

adverb(adv(gan)) --> [gan].
```

Hình 3.2.4 chụp văn phạm DCG câu "Nhà của Nam ở gần trường"

3.2.5 Văn phạm DCG của câu "Nam mới mua mấy cuốn sách mới"

```
% Nam moi mua may cuon sach moi
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP, VP
%--------------- Noun phrase ----------------------------
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN
noun_phrase(np(DET, NM, ADJP)) --> determiner(DET),
```

```
noun_modifier(NM), adjective_phrase(ADJP). % NP --> DET, NM,
ADJP

%---------- Verb phrase ----------
verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP). % VP -> V, NP
verb_phrase(vp(ADVP, VP)) --> adverb_phrase(ADVP),
verb_phrase(VP). % VP -> ADVP, VP

%---------- Adverb phrase ----------
adverb_phrase(advp(ADV)) --> adverb(ADV). % ADVP -> ADV

%---------- Adjective phrase ----------
adjective_phrase(adjp(ADJ)) --> adjective(ADJ). % ADJP -> ADJ

%---------- Noun modifier ----------
noun_modifier(nm(UN, N)) --> unit_noun(UN), noun(N). % NM -> UN,
N

%---------- Determiner ----------
determiner(det(Q)) --> quantity(Q). % DET -> NUM

%---------- Lexicon ----------
proper_noun(propn(nam)) --> [nam].

noun(n(sach)) --> [sach].

verb(v(mua)) --> [mua].

adverb(adv(moi)) --> [moi].

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

unit_noun(un(cuon)) --> [cuon].

quantity(q(may)) --> [may].

% Hàm phân tích cú pháp
parse_tree(Sentence, Tree) :-
```

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

```
% Nam moi mua may cuon sach moi
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP, VP
%--------------- Noun phrase ----------------------------
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN
noun_phrase(np(DET, NM, ADJP)) --> determiner(DET), noun_modifier(NM), adjective_phrase(ADJP). % NP --> DET, NM, ADJP

%--------------- Verb phrase ----------------------------
verb_phrase(vp(V, NP)) --> verb(V), noun_phrase(NP). % VP -> V, NP
verb_phrase(vp(ADVP, VP)) --> adverb_phrase(ADVP), verb_phrase(VP). % VP -> ADVP, VP

%--------------- Adverb phrase ----------------------------
adverb_phrase(advp(ADV)) --> adverb(ADV). % ADVP -> ADV

%--------------- Adjective phrase ----------------------------
adjective_phrase(adjp(ADJ)) --> adjective(ADJ). % ADJP -> ADJ

%--------------- Noun modifier ----------------------------
noun_modifier(nm(UN, N)) --> unit_noun(UN), noun(N). % NM -> UN, N

%--------------- Determiner ----------------------------
determiner(det(Q)) --> quantity(Q). % DET -> NUMA
```

```
%--------------- Lexicon ----------------------------
proper_noun(propn(nam)) --> [nam].

noun(n(sach)) --> [sach].

verb(v(mua)) --> [mua].

adverb(adv(moi)) --> [moi].

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

unit_noun(un(cuon)) --> [cuon].

quantity(q(may)) --> [may].
```

hình 3.2.5 chụp văn phạm DCG câu "Nam mới mua mấy cuốn sách mới"

3.2.6 Văn phạm DCG của câu "Nam tặng Lan một cuốn sách rất hay"

```
% Nam tang lan mot cuon sach rat hay
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP VP
%-------------- Noun phrase ----------------------------
noun_phrase(np(N)) --> noun(N).  %NP -> N
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN
noun_phrase(np(DET, NM, ADJP)) --> determiner(DET),
noun_modifier(NM), adjective_phrase(ADJP). % NP --> DET, NM,
```

```
ADJP

%---------- Verb phrase ----------
verb_phrase(vp(V, NP1, NP2)) --> verb(V), noun_phrase(NP1),
noun_phrase(NP2). % VP -> V, NP1, NP2

%---------- Noun modifier ----------
noun_modifier(nm(UN, N)) --> unit_noun(UN), noun(N). % NM -> UN,
N
%---------- Determiner ----------
determiner(det(Q)) --> quantity(Q). % DET -> NUM
%---------- Adjective phrase ----------
adjective_phrase(adjp(ADV, ADJ)) --> adverb(ADV), adjective(ADJ). %
ADJP - ADV, ADJ

%---------- Lexicon ----------
proper_noun(propn(nam)) --> [nam].
proper_noun(propn(lan)) --> [lan].

noun(n(sach)) --> [sach].

verb(v(tang)) --> [tang].

adverb(adv(rat)) --> [rat].

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

unit_noun(un(cuon)) --> [cuon].

quantity(q(mot)) --> [mot].

% Hàm phân tích cú pháp
parse_tree(Sentence, Tree) :-
   phrase(sentence(Tree), Sentence).
```

```
sentence(s(NP, VP)) --> noun_phrase(NP), verb_phrase(VP). %S -> NP VP
%--------------- Noun phrase ---------------------------
noun_phrase(np(N)) --> noun(N).  %NP -> N
noun_phrase(np(PN))--> proper_noun(PN). % NP -> PROPN
noun_phrase(np(DET, NM, ADJP)) --> determiner(DET), noun_modifier(NM), adjective_phrase(ADJP). % NP --> DET, NM, ADJP

%--------------- Verb phrase ---------------------------
verb_phrase(vp(V, NP1, NP2)) --> verb(V), noun_phrase(NP1), noun_phrase(NP2). % VP -> V, NP1, NP2

%---------------- Noun modifier ---------------------------
noun_modifier(nm(UN, N)) --> unit_noun(UN), noun(N). % NM -> UN, N
%---------------- Determiner ---------------------------
determiner(det(Q)) --> quantity(Q). % DET -> NUM
%---------------- Adjective phrase ---------------------------
adjective_phrase(adjp(ADV, ADJ)) --> adverb(ADV), adjective(ADJ). % ADJP - ADV, ADJ

%---------------- Lexicon ---------------------------
proper_noun(propn(nam)) --> [nam].
proper_noun(propn(lan)) --> [lan].

noun(n(sach)) --> [sach].

verb(v(tang)) --> [tang].

adverb(adv(rat)) --> [rat].

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

unit_noun(un(cuon)) --> [cuon].
```

Hình 3.2.4 chụp văn phạm DCG câu "Nam tặng Lan một cuốn sách rất hay"

4. Thử Nghiệm

4.1 Môi trường thực hiện:

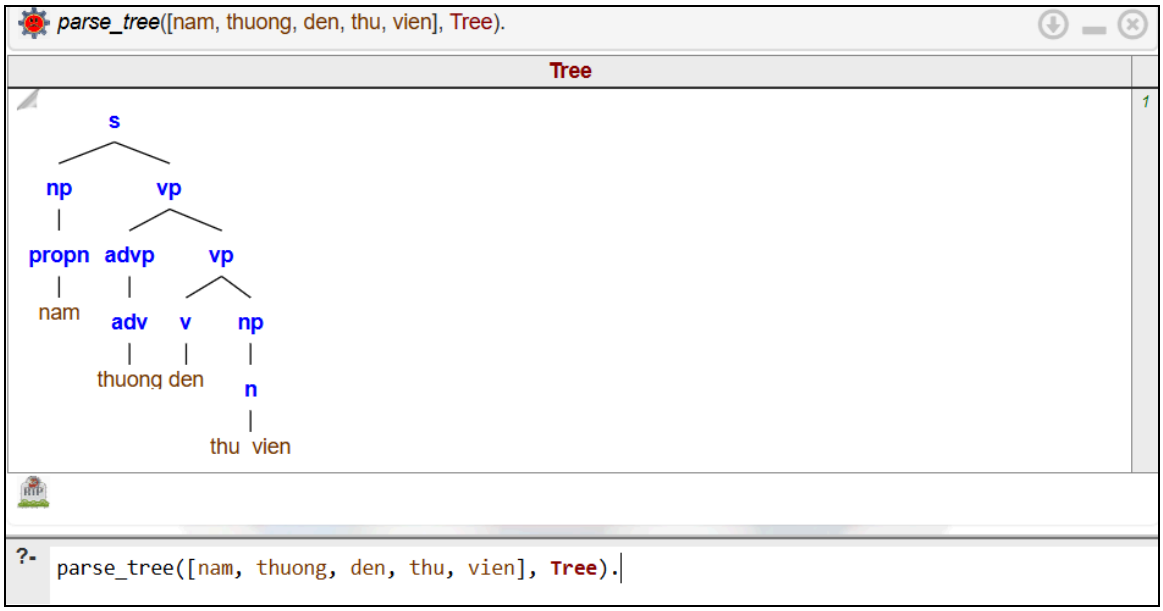
➔ SWISH SWI-Prolog

4.2 Các thử Nghiệm và kết quả

4.2.1 Các câu ở tập dữ liệu gốc

➔ Nam thường đến thư viện

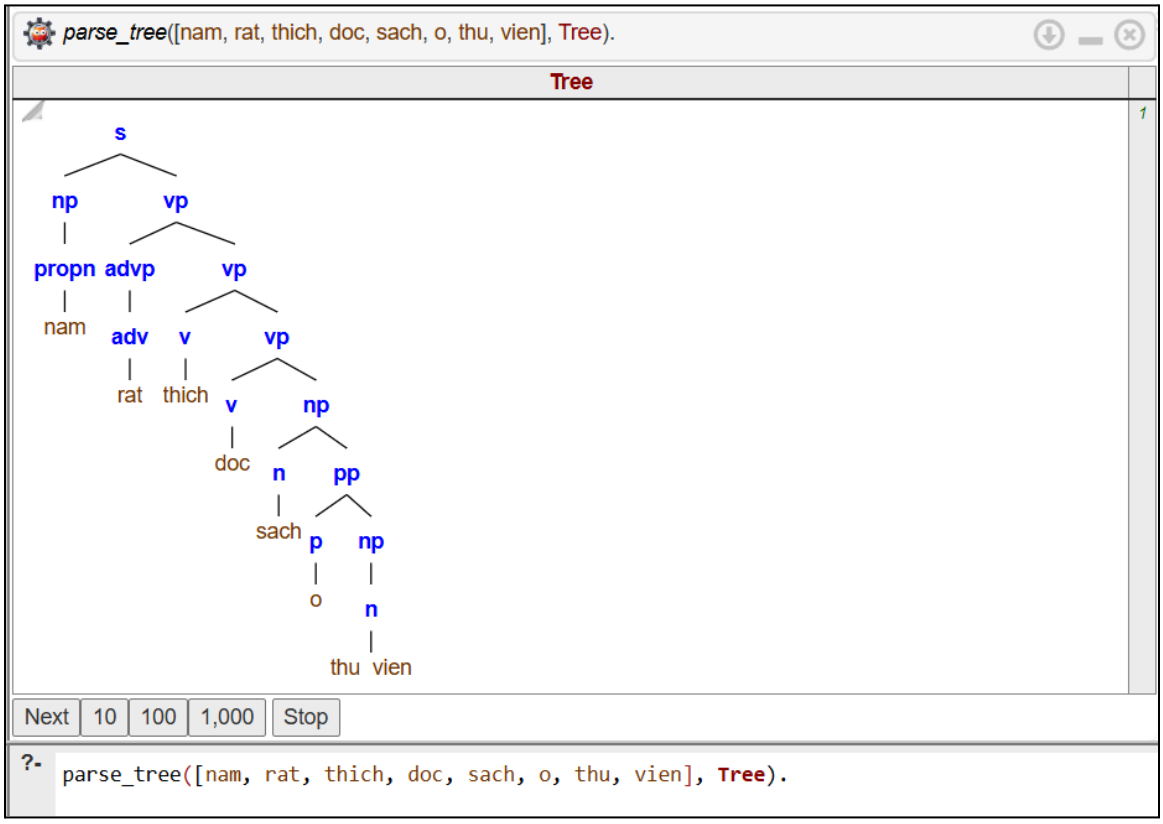
- **Hình vẽ (do Prolog xuất ra từ văn phạm DCG)**



hình. cây cú pháp của câu "Nam thường đến thư viện".

➔ Nam rất thích đọc sách ở thư viện

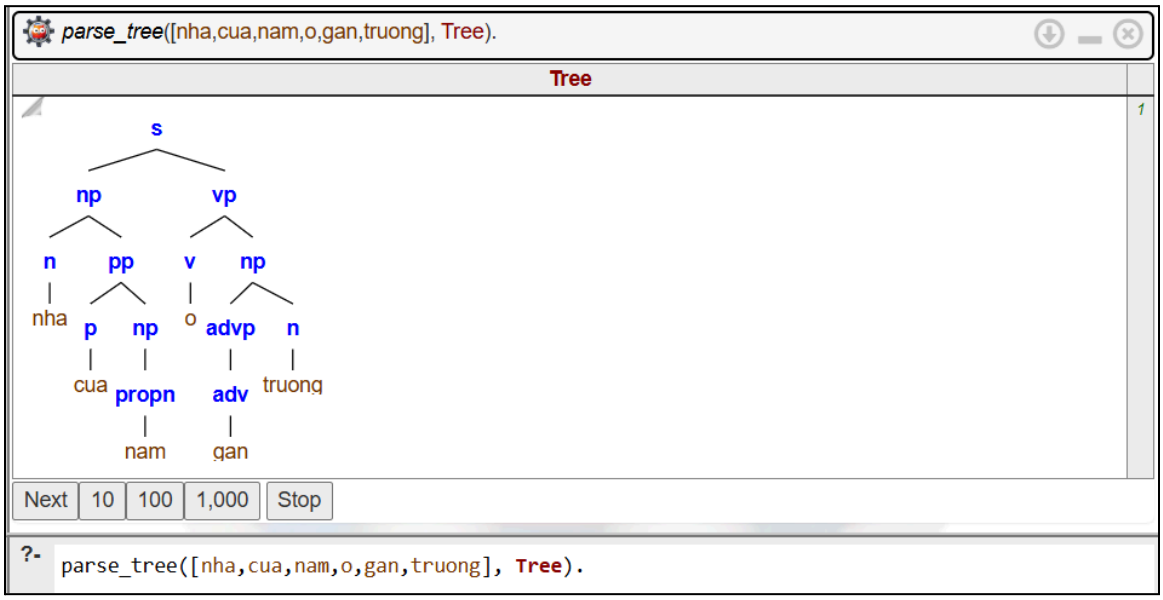
◆ Hình vẽ (do Prolog xuất ra từ văn phạm DCG)



hình cây cú pháp của câu "Nam rất thích đọc sách ở thư viện"

➔ **Nhà của Nam ở gần trường**

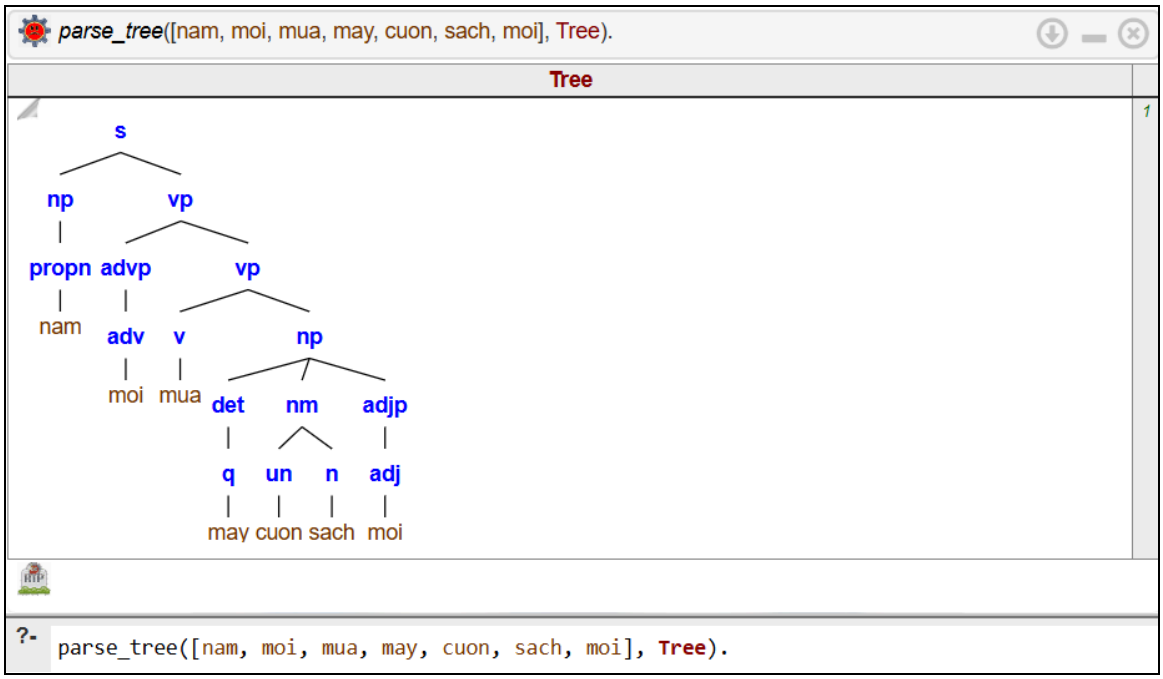
◆ Hình vẽ (do Prolog xuất ra từ văn phạm DCG)



hình 4. cây cú pháp của câu "Nhà của Nam ở gần trường"

➔ **Nam mới mua mấy cuốn sách mới**

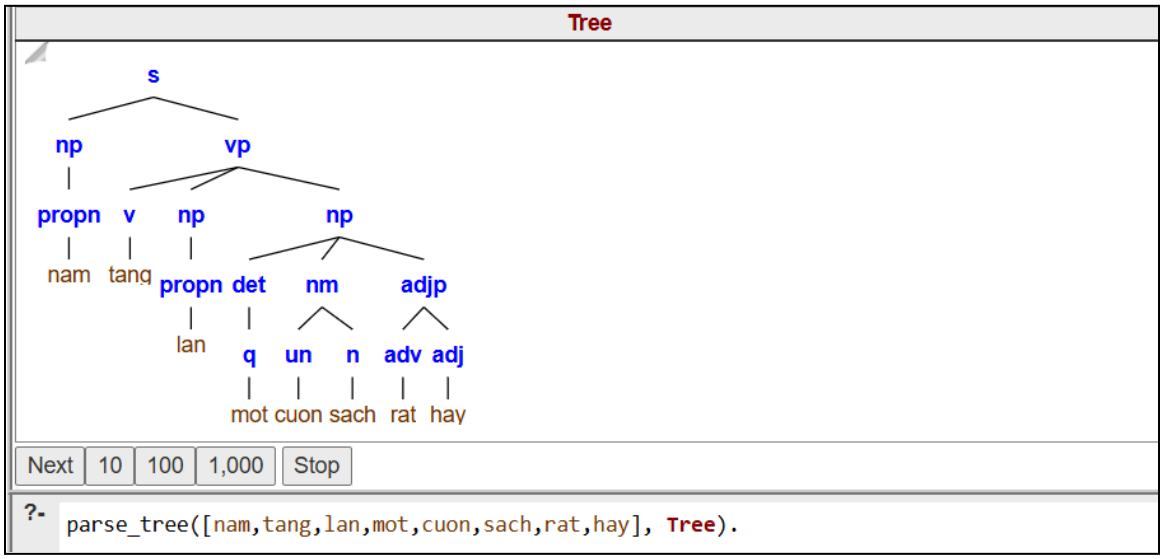
◆ Hình vẽ (do Prolog xuất ra từ văn phạm DCG)



hình cây cú pháp của câu "Nam mới mua mấy cuốn sách mới"

➔ **Nam tặng Lan một cuốn sách rất hay**

◆ Hình vẽ (do Prolog xuất ra từ văn phạm DCG)



hình cây cú pháp của câu "Nam tặng Lan một cuốn sách rất hay"

4.2.2 Các câu phái sinh từ tập dữ liệu gốc

- **Từ câu "Nam thường đến thư viện"**

➔ Nam đến thư viện.

```
?-parse_tree([nam,den,thu,vien], Tree).
Tree = s(np(propn(nam)),vp(v(den),np(n(thu_vien))))
```

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

➔ Nam thích đọc sách ở thư viện.

```
?-parse_tree([nam,thich,doc,sach,o,thu,vien], Tree).
Tree =
s(np(propn(nam)),vp(v(thich),vp(v(doc),np(n(sach),pp(p(o),np(n(thu_vien)
))))))
```

➔ Nam thích đọc sách.

```
?-parse_tree([nam,thich,doc,sach], Tree).
Tree = s(np(propn(nam)),vp(v(thich),vp(v(doc),np(n(sach)))))
```

➔ Nam đọc sách ở thư viện.

```
?-parse_tree([nam,doc,sach,o,thu,vien], Tree).
Tree = s(np(propn(nam)),vp(v(doc),np(n(sach),pp(p(o),np(n(thu_vien))))))
```

➔ Nam đọc sách.

```
?-parse_tree([nam,doc,sach], Tree).
Tree = s(np(propn(nam)),vp(v(doc),np(n(sach))))
```

➔ Nam rất thích sách ở thư viện.

```
?-parse_tree([nam,rat,thich,sach,o,thu,vien], Tree).
Tree =
s(np(propn(nam)),vp(advp(adv(rat)),vp(v(thich),vp(v(doc),np(n(sach))))))
```

➔ Nam rất thích sách.

```
?- parse_tree([nam,rat,thich,sach], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(rat)), vp(v(thich), np(n(sach)))))
```

➔ Nam thích sách ở thư viện.

```
?- parse_tree([nam,thich,sach,o,thu,vien], Tree).
Tree = s(np(propn(nam)), vp(v(thich), np(n(sach), pp(p(o),
np(n(thu_vien))))))
```

➔ Nam thích sách.

```
?- parse_tree([nam,thich,sach], Tree).
Tree = s(np(propn(nam)), vp(v(thich), np(n(sach))))
```

➔ Nam rất thích ở thư viện.

```
?- parse_tree([nam,rat,thich,o,thu,vien], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(rat)), vp(v(thich), vp(v(o),
np(n(thu_vien))))))
```

➔ Nam thích ở thư viện.

```
?- parse_tree([nam,thich,o,thu,vien], Tree).
Tree = s(np(propn(nam)), vp(v(thich), vp(v(o), np(n(thu_vien)))))
```

➔ Nam thích thư viện.

```
?- parse_tree([nam,thich,thu,vien], Tree).
Tree = s(np(propn(nam)), vp(v(thich), np(n(thu_vien))))
```

➔ Nam ở thư viện.

```
?- parse_tree([nam,o,thu,vien], Tree).
Tree = s(np(propn(nam)), vp(v(o), np(n(thu_vien))))
```

- **Từ câu "Nhà của Nam ở gần trường"**

➔ Nhà Nam ở gần trường.

```
?- parse_tree([nha,nam,o,gan,truong], Tree).
Tree = s(np(n(nha), propn(nam)), vp(v(o), np(advp(adv(gan)), n(truong))))
```

➔ Nhà Nam ở trường.

```
?- parse_tree([nha,nam,o,truong], Tree).
Tree = s(np(n(nha), propn(nam)), vp(v(o), np(n(truong))))
```

➔ Nhà của Nam ở trường.

```
?- parse_tree([nha,cua,nam,o,truong], Tree).
Tree = s(np(n(nha), pp(p(cua), np(propn(nam)))), vp(v(o), np(n(truong))))
```

➔ Nhà của Nam gần trường.

```
?- parse_tree([nha,cua,nam,gan,truong], Tree).
Tree = s(np(n(nha), pp(p(cua), np(propn(nam)))), vp(advp(adv(gan)),
np(n(truong))))
```

➔ Nhà Nam gần trường.

```
?- parse_tree([nha,nam,gan,truong], Tree).
Tree = s(np(n(nha), propn(nam)), vp(advp(adv(gan)), 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).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(mua), np(det(q(may)),
nm(un(cuon), n(sach))))))
```

➔ Nam mua mấy cuốn sách mới.

```
?- parse_tree([nam,mua,may,cuon,sach,moi], Tree).
Tree = s(np(propn(nam)), vp(v(mua), np(det(q(may)), nm(un(cuon),
n(sach)), adjp(adj(moi)))))
```

➔ Nam mua mấy cuốn sách.

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

```
Tree = s(np(propn(nam)), vp(v(mua), np(det(q(may)), nm(un(cuon),
n(sach)))))
```

➔ Nam mới mua sách mới.

```
?- parse_tree([nam,moi,mua,sach,moi], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(mua), np(n(sach),
adjp(adj(moi))))))
```

➔ Nam mới mua sách.

```
?- parse_tree([nam,moi,mua,sach], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(mua), np(n(sach)))))
```

➔ Nam mua sách mới.

```
?- parse_tree([nam,mua,sach,moi], Tree).
Tree = s(np(propn(nam)), vp(v(mua), np(n(sach), adjp(adj(moi)))))
```

➔ Nam mua sách.

```
?- parse_tree([nam,mua,sach], Tree).
Tree = s(np(propn(nam)), vp(v(mua), np(n(sach))))
```

➔ Nam mới mua cuốn sách mới.

```
?- parse_tree([nam,moi,mua,sach,moi], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(mua), np(n(sach),
adjp(adj(moi))))))
```

➔ Nam mua cuốn sách mới.

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

➔ Nam mua cuốn sách.

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

```
Tree = s(np(propn(nam)), vp(v(mua), np(nm(un(cuon), n(sach)))))
```

➔ Nam mới mua cuốn sách.

```
?- parse_tree([nam,moi,mua,cuon,sach], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(mua), np(nm(un(cuon),
n(sach))))))
```

➔ Nam mới mua nhà ở gần trường.

```
?- parse_tree([nam,moi,mua,nha,o,gan,truong], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(mua), np(n(nha),
pp(p(o), np(advp(adv(gan)), n(truong)))))))
```

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

➔ Nam tặng Lan một cuốn sách hay.

```
?- parse_tree([nam,tang,lan,mot,cuon,sach,hay], Tree).
Tree = s(np(propn(nam)), vp(v(tang), np(propn(lan)), np(det(q(mot)),
nm(un(cuon), n(sach)), adjp(adj(hay)))))
```

➔ Nam tặng Lan một cuốn sách.

```
?- parse_tree([nam,tang,lan,mot,cuon,sach], Tree).
Tree = s(np(propn(nam)), vp(v(tang), np(propn(lan)), np(det(q(mot)),
nm(un(cuon), n(sach)))))
```

➔ Nam tặng Lan cuốn sách rất hay.

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

➔ Nam tặng Lan cuốn sách hay.

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

➔ Nam tặng Lan sách rất hay.

```
?- parse_tree([nam,tang,lan,sach,rat,hay], Tree).
Tree = s(np(propn(nam)), vp(v(tang), np(propn(lan)), np(n(sach),
adjp(adv(rat), adj(hay)))))
```

➔ Nam tặng Lan sách hay.

```
?- parse_tree([nam,tang,lan,sach,hay], Tree).
Tree = s(np(propn(nam)), vp(v(tang), np(propn(lan)), np(n(sach),
adjp(adj(hay)))))
```

➔ Nam tặng Lan sách.

```
?- parse_tree([nam,tang,lan,sach], Tree).
Tree = s(np(propn(nam)), vp(v(tang), np(propn(lan)), np(n(sach))))
```

➔ Nam tặng Lan cuốn sách.

```
?- parse_tree([nam,tang,lan,cuon,sach], Tree).
Tree = s(np(propn(nam)), vp(v(tang), np(propn(lan)), np(nm(un(cuon),
n(sach)))))
```

➔ Nam tặng Lan.

```
?- parse_tree([nam,tang,lan], Tree).
Tree = s(np(propn(nam)), vp(v(tang), np(propn(lan))))
```

4.2.3 Các câu phái sinh kết hợp từ các câu trong dữ liệu gốc

- **Từ câu "Nam thường đến thư viện" với các câu còn lại**

➔ Nam thường đến trường.

```
parse_tree([nam,thuong,den,truong], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(thuong)), vp(v(den),
np(n(truong)))))
```

➔ Nam thường đến thư viện ở gần trường.

```
?- parse_tree([nam,thuong,den,thu,vien,o,gan,truong], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(thuong)), vp(v(den),
np(n(thu_vien), pp(p(o), np(advp(adv(gan)), n(truong)))))))
```

➔ Nam thích đến thư viện ở gần nhà.

```
?- parse_tree([nam,thich,den,thu,vien,o,gan,nha], Tree).
Tree = s(np(propn(nam)), vp(v(thich), vp(v(den), np(n(thu_vien), pp(p(o),
np(advp(adv(gan)), n(nha)))))))
```

➔ Nam thích đến thư viện gần nhà Lan.

```
?- parse_tree([nam,thich,den,thu,vien,gan,nha,lan], Tree).
false.
```

➔ Nam rất thích đến thư viện ở gần trường.

```
?- parse_tree([nam,thich,den,thu,vien,gan,nha,truong], Tree).
false.
```

➔ Nam đến nhà Lan.

```
?- parse_tree([nam,den,nha,lan], Tree).
Tree = s(np(propn(nam)), vp(v(den), np(n(nha), propn(lan))))
```

➔ Nam thường đến thư viện đọc mấy cuốn sách mới.

```
?- parse_tree([nam,thuong,den,thu,vien,doc,may,cuon,sach,moi], Tree).
false.
```

- **Từ câu "Nam rất thích đọc sách ở thư viện" với các câu còn lại**

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

```
?- parse_tree([nam,rat,thich,doc,sach,o,thu,vien,gan,truong], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(rat)), vp(v(thich), vp(v(doc),
np(n(sach), pp(p(o), np(n(thu_vien)))), np(advp(adv(gan)), n(truong)))))
```

➔ Nam thích đọc sách ở thư viện của trường.

```
?- parse_tree([nam,thich,doc,sach,o,thu,vien,cua,truong], Tree).
Tree = s(np(propn(nam)), vp(v(thich), vp(v(doc), np(n(sach), pp(p(o),
np(n(thu_vien), pp(p(cua), np(n(truong)))))))))
```

➔ Nam thường đọc sách ở thư viện gần trường.

```
?- parse_tree([nam,thuong,doc,sach,o,thu,vien,gan,truong], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(thuong)), vp(v(doc), np(n(sach),
pp(p(o), np(n(thu_vien)))), np(advp(adv(gan)), n(truong)))))
```

➔ Nam rất thích đọc sách ở nhà Lan.

```
?- parse_tree([nam,rat,thich,doc,sach,o,nha,lan], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(rat)), vp(v(thich), vp(v(doc),
np(n(sach), pp(p(o), np(n(nha), propn(lan)))))))
```

➔ Nam đọc sách ở thư viện ở gần nhà.

```
?- parse_tree([nam,doc,sach,o,thu,vien,o,gan,nha], Tree).
Tree = s(np(propn(nam)), vp(v(doc), np(n(sach), pp(p(o), np(n(thu_vien),
pp(p(o), np(advp(adv(gan)), n(nha)))))))
```

➔ Nam rất thích sách ở thư viện gần trường.

```
?- parse_tree([nam,rat,thich,sach,o,thu,vien,gan,truong], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(rat)), vp(v(thich), np(n(sach),
pp(p(o), np(n(thu_vien)))), np(advp(adv(gan)), n(truong)))))
```

➔ Nam thích sách ở thư viện ở gần nhà.

```
?- parse_tree([nam,thich,sach,o,thu,vien,o,gan,nha], Tree).
Tree = s(np(propn(nam)), vp(v(thich), np(n(sach), pp(p(o), np(n(thu_vien),
pp(p(o), np(advp(adv(gan)), n(nha)))))))
```

➔ Nam thích Lan đọc sách.

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

```
false.
```

➔ Nam thích Lan.

```
?- parse_tree([nam,thich,lan], Tree).
Tree = s(np(propn(nam)), vp(v(thich), np(propn(lan))))
```

➔ Nam thích sách của Lan.

```
?- parse_tree([nam,thich,sach,cua,lan], Tree).
Tree = s(np(propn(nam)), vp(v(thich), np(n(sach), pp(p(cua),
np(propn(lan))))))
```

➔ Nam thích Lan tặng sách ở thư viện gần nhà.

```
?- parse_tree([nam,thich,lan,tang,sach,o,thu,vien,gan,nha], Tree).
false.
```

- **Từ câu "Nhà của Nam ở gần trường" với các câu còn lại**

➔ Nhà của Nam ở gần nhà Lan.

```
?- parse_tree([nha,cua,nam,o,gan,nha,lan], Tree).
Tree = s(np(n(nha), pp(p(cua), np(propn(nam)))), vp(v(o),
vp(advp(adv(gan)), np(n(nha), propn(lan)))))
```

➔ Nhà Nam ở gần thư viện.

```
?- parse_tree([nha,nam,o,gan,thu,vien], Tree).
Tree = s(np(n(nha), propn(nam)), vp(v(o), np(advp(adv(gan)),
n(thu_vien))))
```

➔ Nhà Nam ở nhà Lan.

```
?- parse_tree([nha,nam,o,nha,lan], Tree).
Tree = s(np(n(nha), propn(nam)), vp(v(o), np(n(nha), propn(lan))))
```

➔ Nhà của Nam ở trường.

```
?- parse_tree([nha,cua,nam,o,truong], Tree).
Tree = s(np(n(nha), pp(p(cua), np(propn(nam)))), vp(v(o), np(n(truong))))
```

➔ Nhà Nam rất gần trường.

```
?- parse_tree([nha,nam,rat,gan,truong], Tree).
Tree = s(np(n(nha), propn(nam)), vp(advp(adv(rat)), vp(advp(adv(gan)),
np(n(truong)))))
```

➔ Nhà Nam gần Lan.

```
?- parse_tree([nha,nam,gan,lan], Tree).
Tree = s(np(n(nha), propn(nam)), vp(advp(adv(gan)), np(propn(lan))))
```

➔ Nhà của Nam mới mua tặng Lan.

```
?- parse_tree([nha,cua,nam,moi,mua,tang,lan], Tree).
Tree = s(np(n(nha), pp(p(cua), np(propn(nam)))), vp(advp(adv(moi)),
vp(v(mua), vp(v(tang), np(propn(lan))))))
```

- **Từ câu "Nam mới mua mấy cuốn sách mới" với các câu còn lại**

➔ Nam mới mua mấy cuốn sách mới ở thư viện.

```
?- parse_tree([nam,moi,mua,may,cuon,sach,moi], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(mua), np(det(q(may)),
nm(un(cuon), n(sach)), adjp(adj(moi))))))
```

➔ Nam mới mua mấy cuốn sách mới của thư viện gần nhà Lan.

```
?- parse_tree([nam,moi,mua,may,cuon,sach,moi,cua,thu,vien,gan,nha,lan],
Tree).
false.
```

➔ Nam mới mua mấy cuốn sách mới của thư viện gần nhà.

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

```
false.
```

➔ Nam mới mua mấy cuốn sách mới ở gần thư viện.

```
?- parse_tree([nam,moi,mua,may,cuon,sach,moi,o,gan,thu,vien], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(mua), np(det(q(may)),
nm(un(cuon), n(sach)), adjp(adj(moi))), pp(p(o), np(advp(adv(gan)),
n(thu_vien))))))
```

➔ Nam mua mấy cuốn sách mới ở thư viện.

```
?- parse_tree([nam,mua,may,cuon,sach,moi,o,thu,vien], Tree).
Tree = s(np(propn(nam)), vp(v(mua), np(det(q(may)), nm(un(cuon),
n(sach)), adjp(adj(moi))), pp(p(o), np(n(thu_vien)))))
```

➔ Nam mua mấy cuốn sách mới của thư viện.

```
?- parse_tree([nam,mua,may,cuon,sach,moi,cua,thu,vien], Tree).
Tree = s(np(propn(nam)), vp(v(mua), np(det(q(may)), nm(un(cuon),
n(sach)), adjp(adj(moi))), pp(p(cua), np(n(thu_vien)))))
```

➔ Nam mới mua sách mới.

```
?- parse_tree([nam,moi,mua,sach,moi], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(mua), np(n(sach),
adjp(adj(moi))))))
```

➔ Nam mua sách mới ở gần thư viện.

```
?- parse_tree([nam,mua,sach,moi,o,gan,thu,vien], Tree).
Tree = s(np(propn(nam)), vp(v(mua), np(n(sach), adjp(adj(moi))), pp(p(o),
np(advp(adv(gan)), n(thu_vien)))))
```

➔ Nam mua sách mới ở trường.

```
?- parse_tree([nam,mua,sach,moi,o,truong], Tree).
Tree = s(np(propn(nam)), vp(v(mua), np(n(sach), adjp(adj(moi))), pp(p(o),
np(n(truong)))))
```

➔ Nam mua sách mới của Lan.

```
?- parse_tree([nam,mua,sach,moi,cua,lan], Tree).
Tree = s(np(propn(nam)), vp(v(mua), np(n(sach), adjp(adj(moi))),
pp(p(cua), np(propn(lan)))))
```

➔ Nam mua sách mới tặng Lan.

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

- **Từ câu "Nam tặng Lan một cuốn sách rất hay" với các câu còn lại**

➔ Nam mới tặng Lan một cuốn sách rất hay.

```
?- parse_tree([nam,moi,tang,lan,mot,cuon,sach,rat,hay], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(tang), np(propn(lan)),
np(det(q(mot)), nm(un(cuon), n(sach)), adjp(adv(rat), adj(hay))))))
```

➔ Nam tặng Lan mấy cuốn sách rất hay.

```
?- parse_tree([nam,tang,lan,may,cuon,sach,rat,hay], Tree).
Tree = s(np(propn(nam)), vp(v(tang), np(propn(lan)), np(det(q(may)),
nm(un(cuon), n(sach)), adjp(adv(rat), adj(hay)))))
```

➔ Nam thường tặng Lan một cuốn sách rất hay.

```
?- parse_tree([nam,thuong,tang,lan,mot,cuon,sach,rat,hay], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(thuong)), vp(v(tang),
np(propn(lan)), np(det(q(mot)), nm(un(cuon), n(sach)), adjp(adv(rat),
adj(hay))))))
```

➔ Nam thích tặng Lan một cuốn sách rất hay.

```
?- parse_tree([nam,thich,tang,lan,mot,cuon,sach,rat,hay], Tree).
Tree = s(np(propn(nam)), vp(v(thich), vp(v(tang), np(propn(lan)),
np(det(q(mot)), nm(un(cuon), n(sach)), adjp(adv(rat), adj(hay))))))
```

➔ Nam mới tặng Lan mấy cuốn sách hay.

```
?- parse_tree([nam,moi,tang,lan,may,cuon,sach,rat,hay], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(tang), np(propn(lan)),
np(det(q(may)), nm(un(cuon), n(sach)), adjp(adv(rat), adj(hay))))))
```

➔ Nam mới tặng Lan một cuốn sách hay.

```
?- parse_tree([nam,moi,tang,lan,mot,cuon,sach,hay], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(tang), np(propn(lan)),
np(det(q(mot)), nm(un(cuon), n(sach)), adjp(adj(hay))))))
```

➔ Nam thích tặng Lan một cuốn sách hay.

```
?- parse_tree([nam,thich,tang,lan,mot,cuon,sach,hay], Tree).
Tree = s(np(propn(nam)), vp(v(thich), vp(v(tang), np(propn(lan)),
np(det(q(mot)), nm(un(cuon), n(sach)), adjp(adj(hay))))))
```

➔ Nam mới tặng Lan một cuốn sách.

```
?- parse_tree([nam,moi,tang,lan,mot,cuon,sach], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(moi)), vp(v(tang), np(propn(lan)),
np(det(q(mot)), nm(un(cuon), n(sach))))))
```

➔ Nam rất thích tặng Lan cuốn sách hay.

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

➔ Nam thích tặng Lan cuốn sách hay.

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

➔ Nam thích tặng Lan sách rất hay.

```
?- parse_tree([nam,thich,tang,lan,sach,rat,hay], Tree).
Tree = s(np(propn(nam)), vp(v(thich), vp(v(tang), np(propn(lan)),
```

```
np(n(sach), adjp(adv(rat), adj(hay))))))
```

➔ Nam rất thích tặng Lan sách hay.

```
?- parse_tree([nam,rat,thich,tang,lan,sach,hay], Tree).
Tree = s(np(propn(nam)), vp(advp(adv(rat)), vp(v(thich), vp(v(tang),
np(propn(lan)), np(n(sach), adjp(adj(hay)))))))
```

➔ Nam thích tặng Lan sách.

```
?- parse_tree([nam,thich,tang,lan,sach], Tree).
Tree = s(np(propn(nam)), vp(v(thich), vp(v(tang), np(propn(lan)),
np(n(sach)))))
```

➔ Nam tặng thư viện một cuốn sách rất hay.

```
?- parse_tree([nam,tang,thu,vien,mot,cuon,sach,rat,hay], Tree).
Tree = s(np(propn(nam)), vp(v(tang), np(n(thu_vien)), np(det(q(mot)),
nm(un(cuon), n(sach)), adjp(adv(rat), adj(hay)))))
```

III. Kết luận và hướng phát triển

1. Tổng Kết Các Công Việc Đã Thực Hiện

Trong quá trình nghiên cứu và phát triển hệ thống phân tích cú pháp, chúng tôi đã hoàn thành việc xây dựng các quy tắc ngữ pháp bằng cách sử dụng văn phạm DCG trong Prolog. Quá trình tiền xử lý văn bản đã được triển khai để đảm bảo tính chính xác trong phân tích cú pháp. tôi cũng đã thực thi và kiểm tra các quy tắc ngữ pháp để xác định khả năng phân tích các câu đầu vào một cách hiệu quả.

2. Ưu Điểm của Phân Tích Cú Pháp Dựa trên Luật với DCG

- **Tính linh hoạt**: Hệ thống cho phép mở rộng và điều chỉnh dễ dàng, giúp thích ứng với nhiều loại ngôn ngữ khác nhau.

- **Hiệu suất cao**: Việc sử dụng DCG giúp giảm thiểu độ phức tạp trong việc xây dựng các quy tắc ngữ pháp, đồng thời tăng cường khả năng phân tích chính xác.
- **Tính trực quan**: Cây phân tích cú pháp được sinh ra giúp người dùng dễ dàng hiểu cấu trúc câu và mối quan hệ giữa các thành phần trong câu.
- **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, giúp 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, chúng ta có thể kết hợp các biểu diễn ngữ nghĩa vào 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 và dễ dàng điều chỉnh để phù hợp với yêu cầu của ứng dụng.

3. Nhược Điểm của Phân Tích Cú Pháp Dựa trên Luật với DCG

- **Khó khăn trong việc xây dựng quy tắc**: Việc tạo ra và duy trì các quy tắc ngữ pháp có thể phức tạp và đòi hỏi nhiều công sức, đặc biệt khi làm việc với các ngôn ngữ tự nhiên phong phú và đa dạng.
- **Độ phức tạp của ngữ pháp**: Khi ngữ pháp trở nên phức tạp hơn, việc xây dựng cây phân tích cú pháp có thể dẫn đến việc quy tắc trở nên khó hiểu và khó quản lý. Điều này có thể làm giảm hiệu suất và độ chính xác của quá trình phân tích.

- **Tính không hoàn chỉnh**: Các quy tắc ngữ pháp dựa trên luật có thể không bao quát tất cả các cấu trúc ngôn ngữ. Do đó, có thể xuất hiện trường hợp mà các câu hợp lệ không được xử lý đúng cách.
- **Thiếu tính tự động hóa**: Trong khi DCG cho phép một số tự động hóa trong việc phân tích cú pháp, việc xác định và điều chỉnh các quy tắc vẫn cần sự can thiệp của con người, điều này có thể dẫn đến tốn thời gian và công sức.
- **Khả năng tổng quát hạn chế**: Các mô hình dựa trên quy tắc có thể không tổng quát tốt cho các ngữ cảnh khác nhau hoặc cho các dạng ngôn ngữ chưa được định nghĩa trước. Điều này có thể làm giảm tính linh hoạt của hệ thống.
- **Chi phí tính toán**: Đối với một số ngữ pháp phức tạp, quá trình phân tích cú pháp có thể tốn nhiều tài nguyên tính toán, đặc biệt khi làm việc với các tập dữ liệu lớn hoặc khi xử lý nhiều ngữ cảnh khác nhau.

Đây là các câu có kết quả trả về là false:

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

4. Kết luận

Phân tích cú pháp đóng vai trò quan trọng trong lĩnh vực khoa học máy tính, không chỉ phục vụ việc phát triển ngôn ngữ lập trình mà còn góp phần lớn trong xử lý ngôn ngữ tự nhiên. Từ khi ngôn ngữ lập trình đầu tiên xuất

hiện vào những năm 1950 cho đến sự đột phá của học sâu trong những năm gần đây, sự phát triển không ngừng của các phương pháp và công cụ phân tích cú pháp đã thể hiện rõ bước tiến vượt bậc của ngành.

Việc áp dụng các quy tắc ngữ pháp và mô hình phân tích giúp xây dựng các công cụ phân tích cú pháp hiệu quả, phục vụ nhiều mục đích từ biên dịch mã máy tính đến xử lý ngôn ngữ tự nhiên phức tạp, giúp máy tính có thể hiểu và xử lý văn bản một cách chính xác hơn. Nhờ đó, khả năng giao tiếp giữa con người và máy móc được cải thiện, mở ra nhiều ứng dụng thông minh trong các lĩnh vực khác nhau.

Trong bối cảnh công nghệ phát triển nhanh chóng, các phương pháp và công cụ phân tích cú pháp cũng sẽ không ngừng được nâng cấp, tạo ra nhiều cơ hội và thách thức mới. Việc chuyển từ ngữ pháp dựa trên quy tắc đến các mô hình học sâu tiếp tục nâng cao năng lực của máy tính trong việc xử lý ngôn ngữ và dữ liệu phức tạp, góp phần làm giàu thêm khả năng tương tác giữa con người và máy tính, mở ra những triển vọng lớn trong tương lai.

5. Hướng Phát Triển

Trong tương lai, chúng tôi dự định cải thiện hệ thống phân tích cú pháp bằng cách:

- **Nâng cao khả năng hiểu ngữ nghĩa**: Phát triển các quy tắc ngữ nghĩa để cải thiện khả năng phân tích ngữ nghĩa trong các câu phức tạp.
- **Tích hợp học máy**: Kết hợp các kỹ thuật học máy để tự động hóa quá trình xây dựng quy tắc và cải thiện độ chính xác trong phân tích.
- **Mở rộng ngôn ngữ hỗ trợ**: Mở rộng hệ thống để hỗ trợ nhiều ngôn ngữ tự nhiên khác nhau, tạo ra một công cụ phân tích cú pháp đa ngôn ngữ.

TÀI LIỆU THAM KHẢO

1. **Knuth, Donald E.** "On the Translation of Languages from Left to Right." *Information and Control*, 1965.
2. **Aho, Alfred V., và Jeffrey D. Ullman.** *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.
3. **Backus, John, và Peter Naur.** "Revised Report on the Algorithmic Language ALGOL 60." *Communications of the ACM*, 1960.
4. **Earley, Jay.** "An Efficient Context-Free Parsing Algorithm." *Communications of the ACM*, 1970.
5. **Cocke, John, Daniel H. Younger, và Robert M. Kasami.** "An Efficient Parsing Method for Context-Free Languages." *ACM Transactions on Programming Languages and Systems*, 1965.
6. **Chomsky, Noam.** "Three Models for the Description of Language." *IRE Transactions on Information Theory*, 1956.
7. **Manning, Christopher D. và Hinrich Schütze.** *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
8. **Vaswani, Ashish, et al.** "Attention Is All You Need." *Advances in Neural Information Processing Systems*, 2017.
9. **Hugging Face.** "Transformers." (https://huggingface.co/transformers/)
10. **spaCy Documentation.** (https://spacy.io/)
11. https://en.wikipedia.org/wiki/Definite_clause_grammar
12. https://ongthovuive.wordpress.com/2016/04/21/xu-ly-ngon-ngu-tu-nhien-phan-1/
13. https://ongthovuive.wordpress.com/2016/04/22/xu-ly-ngon-ngu-tu-nhien-phan-2/

14. https://ongthovuive.wordpress.com/2016/04/23/xu-ly-ngon-ngu-tu-nhien-phan-3/
15. https://ongthovuive.wordpress.com/2016/04/29/xu-ly-ngon-ngu-tu-nhien-phan-4/