

**TRƯỜNG ĐẠI HỌC CẦN THƠ**  
**TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**  
**KHOA KHOA HỌC MÁY TÍNH**



**BÁO CÁO HP TIN HỌC LÝ THUYẾT (CT121)**  
**NHÓM HỌC PHẦN: NHÓM 1**

**ĐỀ TÀI:**  
**ĐỌC VĂN PHẠM VÀ CHUYỂN VĂN PHẠM**  
**VỀ DẠNG CHUẨN CHOMSKY**

<b>NHÓM THỰC HIỆN:</b>	<b>NHÓM 18</b>	
<b>STT SV TRONG DS</b>	<b>HỌ TÊN SINH VIÊN</b>	<b>MSSV</b>
18	Quách Tuấn Khang	B2207526
37	Nguyễn Minh Thuận	B2207568
38	Lai Trung Tín	B2207570

HK2 2024-2025

*Cần Thơ, 4/2025*

## NHẬN XÉT CỦA GIẢNG VIÊN

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

*Cần Thơ, ngày 27 tháng 03 năm 2025*

(GV Ký và ghi rõ họ tên)

## MỤC LỤC

<b>DANH MỤC HÌNH ẢNH.....</b>	<b>4</b>
<b>DANH MỤC BẢNG BIỂU.....</b>	<b>5</b>
<b>DANH MỤC TỪ VIẾT TẮT.....</b>	<b>6</b>
<b>BẢNG PHÂN CÔNG CÔNG VIỆC.....</b>	<b>7</b>
<b>TỔNG QUAN.....</b>	<b>8</b>
<b>PHẦN I. GIỚI THIỆU.....</b>	<b>9</b>
1. Đặt vấn đề.....	9
2. Những nghiên cứu liên quan.....	9
3. Mục tiêu đề tài.....	10
4. Đối tượng và phạm vi nghiên cứu.....	10
5. Phương pháp nghiên cứu.....	10
6. Bố cục quyển báo cáo.....	11
<b>PHẦN II. NỘI DUNG.....</b>	<b>12</b>
<b>Chương 1. Mục tiêu bài toán.....</b>	<b>12</b>
1.1. Mô tả chi tiết bài toán.....	12
1.1.1. Văn phạm vi ngữ cảnh.....	12
1.1.2. Các ký hiệu vô ích.....	12
1.1.3. Luật sinh.....	12
1.1.4. Luật sinh đơn vị.....	13
1.1.5. Dạng chuẩn Chomsky.....	13
1.2. Công nghệ sử dụng.....	13
1.2.1. Ngôn ngữ lập trình Python.....	13
1.2.2. Thư Viện Tkinter.....	14
1.2.3. Visual Studio Code (VSCode).....	14
<b>Chương 2. Phương pháp thực hiện.....</b>	<b>15</b>
2.1. Đọc văn phạm phi ngữ cảnh.....	15
2.2. Giảm lược văn phạm phi ngữ cảnh.....	15
2.3. Chuẩn hóa văn phạm phi ngữ cảnh thành dạng chuẩn Chomsky.....	15
<b>Chương 3. Thiết kế và cài đặt.....</b>	<b>16</b>
3.1. Cấu trúc dữ liệu và cấu trúc tập tin.....	16
3.2. Tạo phương thức đọc văn phạm.....	17
3.3. Tạo phương thức loại bỏ ký hiệu vô ích.....	17
3.4. Tạo phương thức loại bỏ luật sinh epsilon.....	18
3.5. Tạo phương thức loại bỏ luật sinh đơn vị.....	19
3.6. Tạo phương thức chuyển các quy tắc thành dạng chuẩn Chomsky.....	20
3.7. Thiết kế giao diện bằng thư viện Tkinter.....	21

<b>PHẦN III. ĐÁNH GIÁ KIỂM THỬ.....</b>	<b>23</b>
1. Mục tiêu.....	23
2. Kiểm thử.....	23
2.1 Kiểm thử lần 1.....	23
2.2 Kiểm thử lần 2.....	24
<b>PHẦN IV . KẾT LUẬN.....</b>	<b>26</b>
1. Kết quả đạt được.....	26
2. Hướng phát triển.....	26
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>26</b>

## DANH MỤC HÌNH ẢNH

Hình 3.1. Biểu diễn cấu trúc file text (.txt).....	16
Hình 3.1. Biểu diễn phương thức đọc văn phạm.....	17
Hình 3.2. Biểu diễn phương thức loại bỏ ký hiệu vô ích.....	17
Hình 3.3. Biểu diễn phương thức loại bỏ luật sinh epsilon.....	18
Hình 3.4. Biểu diễn phương thức loại bỏ luật sinh đơn vị.....	19
Hình 3.4. Biểu diễn phương thức chuyển về cnf.....	20
Hình 3.5. Giao diện hệ thống ban đầu sau khi được biên dịch.....	21
Hình 3.6. Giao diện hệ thống sau khi được điền dữ liệu.....	22
Hình 3.7. Giao diện hệ thống sau khi xử lý chuyển văn phạm về dạng chuẩn Chomsky.....	22

## **DANH MỤC BẢNG BIỂU**

## DANH MỤC TỪ VIẾT TẮT

STT	Từ viết tắt (Abbreviation)	Từ viết đầy đủ (Origin word)
1	CFG	Context Free Grammar
2	CNF	Chomsky Normal Form

## BẢNG PHÂN CÔNG CÔNG VIỆC

STT	MSSV	Họ và tên	Công việc phụ trách
1	B2207526	Quách Tuấn Khang	<ul style="list-style-type: none"> <li>- Viết báo cáo:                             <ul style="list-style-type: none"> <li>+ Phần I: Giới thiệu</li> <li>+ Phần III: Đánh giá và kiểm thử</li> </ul> </li> <li>- Viết chương trình demo:                             <ul style="list-style-type: none"> <li>+ Đánh giá, kiểm thử code.</li> </ul> </li> </ul>
2	B2207568	Nguyễn Minh Thuận	<ul style="list-style-type: none"> <li>- Viết báo cáo:                             <ul style="list-style-type: none"> <li>+ Phần I: Giới thiệu</li> <li>+ Phần II: Nội dung                                     <ul style="list-style-type: none"> <li>Chương 1: Mục tiêu bài toán.</li> <li>Chương 2: Phương pháp thực hiện.</li> </ul> </li> <li>+ Phần IV: Kết luận.</li> </ul> </li> <li>- Viết chương trình demo:                             <ul style="list-style-type: none"> <li>+ Xây dựng giao diện.</li> </ul> </li> </ul>
3	B2207570	Lai Trung Tín	<ul style="list-style-type: none"> <li>- Viết báo cáo:                             <ul style="list-style-type: none"> <li>+ Phần I: Giới thiệu</li> <li>+ Phần II: Nội dung                                     <ul style="list-style-type: none"> <li>Chương 3: Thiết kế cài đặt</li> </ul> </li> </ul> </li> <li>- Viết chương trình demo:                             <ul style="list-style-type: none"> <li>+ Xây dựng các hàm xử lý</li> </ul> </li> </ul>



## TỔNG QUAN

Đề tài “ Đọc văn phạm và Chuyển văn phạm về dạng chuẩn Chomsky” cho ta cái nhìn tổng quan về đọc vào văn phạm phi ngữ cảnh (CFG) và chuyển văn phạm về dạng chuẩn Chomsky (CNF) cũng như có nhiều ứng dụng thực tế rất quan trọng, đặc biệt trong việc biểu diễn ngôn ngữ lập trình. Trong quá trình demo, người tham gia sẽ thực hiện hướng dẫn cơ quả về các khái niệm đến các thuật toán xử lý.

Chúng ta xây dựng một công cụ đơn giản nhưng ổn định, cho phép người dùng nhập vào văn phạm và tập luật sinh trực tiếp hoặc thông qua đọc dữ liệu từ file. Sau đó thông qua xử lý ngôn ngữ tự nhiên và các cú pháp, hệ thống sẽ phân tích văn phạm và tiến hành đánh giá xử lý văn phạm.

Cuối cùng, thông qua quá trình demo đọc văn phạm và chuyển văn phạm về dạng chuẩn Chomsky, người thực hiện không chỉ nắm vững kiến thức hơn về CFG và CNF mà còn hơn về cách áp dụng và tập dụng để giải quyết các bài toán thực tế trong lĩnh vực khoa học máy tính.

# PHẦN I. GIỚI THIỆU

## 1. Đặt vấn đề

Trong lĩnh vực lý thuyết ngôn ngữ hình thức và khoa học máy tính, văn phạm phi ngữ cảnh (Context-Free Grammar - CFG) đóng một vai trò then chốt trong việc định nghĩa các ngôn ngữ phi ngữ cảnh (Context-Free Languages - CFL)[2]. Chúng cung cấp một ký pháp đệ quy tự nhiên để mô tả cấu trúc của các ngôn ngữ này. Đặc biệt, CFG đã trở thành nền tảng trong công nghệ biên dịch, giúp biến việc xây dựng các bộ phân tích cú pháp (parsers) từ một công việc phức tạp, tùy chỉnh thành một quy trình có thể thực hiện một cách dễ dàng.

Dạng chuẩn Chomsky (CNF) là một dạng rút gọn của CFG, trong đó mọi luật sinh có dạng:  $A \rightarrow BC$  (hai ký hiệu không kết thúc), hoặc  $A \rightarrow a$  (một ký hiệu kết thúc). Việc chuyển đổi CFG sang CNF là bắt buộc để áp dụng các thuật toán phân tích cú pháp hiệu quả, nhưng quá trình này đòi hỏi loại bỏ các luật sinh không phù hợp (vô ích,  $\epsilon$ , đơn vị) và tối ưu hóa cấu trúc văn phạm. Bài báo cáo này tập trung vào việc xây dựng giải pháp chuyển đổi CFG sang CNF tối ưu, kèm công cụ hỗ trợ trực quan hóa quá trình chuyển đổi.

## 2. Những nghiên cứu liên quan

Lý thuyết nền tảng: Để thực hiện nghiên cứu này là “Chomsky, Noam (1959)” giới thiệu sơ lược CNF và các quy tắc chuyển đổi cơ bản từ CFG thành CNF. Tài liệu nghiên cứu tiếp theo sách giáo trình Introduction to Automata Theory, Languages and Computation của John E. Hopcroft, Jeffrey D. Ullman, 1979 [2] đã mô tả thuật toán loại bỏ  $\epsilon$ -production bằng cách xác định các ký hiệu nullable.

Công cụ hiện có: Thư viện NLTK (Python) cung cấp hàm `chomsky_normal_form()`, có thể chuẩn hóa văn phạm thành dạng chuẩn Chomsky. nhưng kết quả chưa tối ưu về số lượng luật sinh. Công cụ ANTLR hỗ trợ sinh cú pháp từ CFG nhưng không trực quan hóa quá trình chuyển đổi.

Một số hạn chế của nghiên cứu trước:

- Tối ưu hóa: Một số thuật toán tạo ra số lượng lớn ký hiệu không kết thúc không cần thiết.
- Thiếu minh bạch: Không có công cụ hiển thị chi tiết từng bước chuyển đổi, gây khó hiểu cho người dùng.
- Xử lý trường hợp đặc biệt: Chưa giải quyết hiệu quả văn phạm có chu trình hoặc đệ quy phức tạp.

### 3. Mục tiêu đề tài

Mục tiêu tổng quát: Xây dựng giải pháp chuyển đổi CFG sang CNF tối ưu, kèm công cụ trực quan hóa quy trình, đảm bảo tính chính xác và hiệu suất cao.

Mục tiêu cụ thể:

- Phân tích và đọc đầu vào CFG từ các định dạng phổ biến.
- Triển khai thuật toán chuyển đổi CNF theo từng bước:
  - + Loại bỏ luật sinh vô ích
  - + Loại bỏ luật sinh  $\epsilon$ .
  - + Loại bỏ luật sinh đơn vị ( $A \rightarrow B$ ).
  - + Phân rã luật sinh có độ dài  $> 2$  thành các luật sinh đôi.
  - + Giảm thiểu số lượng ký hiệu không kết thúc mới sinh ra.
- Phát triển giao diện đồ họa hiển thị từng bước chuyển đổi cụ thể.
- Đánh giá hiệu suất trên các văn phạm mẫu.

### 4. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu: Văn phạm phi ngữ cảnh không chứa ký hiệu vô ích hoặc không kết thúc. Các luật sinh có dạng  $A \rightarrow \alpha$ , trong đó  $\alpha$  là chuỗi ký hiệu kết thúc/không kết thúc.

Phạm vi nghiên cứu:

- Giới hạn chức năng: Chỉ tập trung vào chuyển đổi sang CNF, không bao gồm dạng chuẩn Greibach.
- Giới hạn dữ liệu: Văn phạm tĩnh, không xét trường hợp văn phạm xác suất hoặc mờ.

### 5. Phương pháp nghiên cứu

Phương pháp tổng hợp lý thuyết nghiên cứu các thuật toán kinh điển:

- Loại bỏ  $\epsilon$ -production: Xác định ký hiệu nullable và điều chỉnh luật sinh.
- Loại bỏ unit production: Sử dụng bao đóng bắc cầu.
- Phân rã luật sinh: Thay thế luật sinh dài bằng các cặp ký hiệu mới.

Phương pháp mô phỏng

- Triển khai thuật toán: Sử dụng Python và thư viện Tinker để xây dựng giao diện đồ họa.
- Kiểm thử: Áp dụng Test-Driven Development (TDD) với các test case từ sách giáo khoa và văn phạm thực tế.

Phương pháp đánh giá hiệu suất

- Chỉ số đánh giá: Thời gian chuyển đổi và Số lượng luật sinh và ký hiệu không kết thúc mới được tạo ra.
- So sánh: Đối chiếu kết quả với công cụ như NLTK và ANTLR.

## **6. Bố cục quyển báo cáo**

### Phần I. Giới thiệu

- Giới thiệu tổng quan đề tài.

### Phần II. Nội dung

- Chương 1. Mục tiêu bài toán: Giới thiệu chi tiết bài toán.
- Chương 2. Phương pháp thực hiện bài toán.
- Chương 3. Thiết kế và cài đặt xử lý bài toán.

### Phần III. Đánh giá kiểm thử.

- Thực hiện đánh giá kết quả.

### Phần IV. Kết luận

- Kết quả đạt được và hướng phát triển đề tài.

## PHẦN II. NỘI DUNG

### Chương 1. Mục tiêu bài toàn

#### 1.1. Mô tả chi tiết bài toán

##### 1.1.1. Văn phạm vi ngữ cảnh

Văn phạm phi ngữ cảnh [1] (CNF) là một tập hợp hữu hạn các *biến* (còn gọi là các ký hiệu chưa kết thúc), mỗi biến biểu diễn một ngôn ngữ. Ngôn ngữ biểu diễn bởi các biến được mô tả một cách đệ quy theo thuật ngữ của một khái niệm khác gọi là *ký hiệu kết thúc*. Quy tắc quan hệ giữa các biến gọi là luật sinh. Mỗi luật sinh có dạng một biến ở vế trái sinh ra một chuỗi có thể gồm các biến lẫn các ký hiệu kết thúc trong văn phạm.

Văn phạm phi ngữ cảnh là một hệ thống gồm bốn thành phần, ký hiệu văn phạm  $G$  ( $V, T, P, S$ ), trong đó:

- +  $V$  là tập hữu hạn các biến (hay ký tự chưa kết thúc).
- +  $T$  là tập hữu hạn các ký tự kết thúc, ( $V \cap T = \emptyset$ ).
- +  $P$  là tập hữu hạn các luật sinh mà mỗi luật sinh có dạng  $A \rightarrow \alpha$  với  $A$  là biến và  $\alpha$  là chuỗi các ký hiệu  $\in (V \cup T)^*$ .
- +  $S$  là một chuỗi biến đặc biệt gọi là ký hiệu bắt đầu văn phạm.

Các quy tắc biểu diễn:

- +  $V$ : chữ in hoa ( $A, B, C, \dots$ );
- +  $T$ : chữ in thường ( $a, b, c, \dots, w, x, y, \dots$ )
- +  $\alpha, \beta, \gamma, \dots$  biểu diễn chuỗi ký hiệu kết thúc và biến.

##### 1.1.2. Các ký hiệu vô ích

Một ký hiệu  $X$  gọi là có ích nếu có dẫn xuất  $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$  với các chuỗi  $\alpha, \beta$  bất kỳ và  $w \in T^*$ . Ngược lại  $X$  gọi là vô ích.

Vậy, có 2 đặc điểm nhận dạng cho ký hiệu vô ích:

- $X$  phải dẫn ra một chuỗi ký hiệu kết thúc.
- $X$  phải nằm trong dẫn xuất từ  $S$ .

Tuy nhiên 2 dấu hiệu trên không đủ đảm bảo  $X$  có ích vì  $X$  có thể nằm trong dạng câu chứa biến nhưng từ đó không có ký hiệu kết thúc được sinh ra.

##### 1.1.3. Luật sinh $\epsilon$

Luật sinh  $\epsilon$  là quy tắc trong ngữ pháp phi ngữ cảnh (CFG) có dạng  $A \rightarrow \epsilon$ , với  $A$  là một biến và  $\epsilon$  là chuỗi rỗng.

**Định lý 5.3** cho phép loại bỏ các luật sinh  $A \rightarrow \epsilon$  từ CFG  $G = (V, T, P, S)$ , trừ khi chuỗi rỗng  $\epsilon$  nằm trong ngôn ngữ  $L$  sinh bởi  $G$ . Sau khi loại bỏ:

- Ngữ pháp mới:  $G' = (V, T, P', S)$ .
- $G'$  không chứa ký hiệu vô ích (biến hoặc ký hiệu không sinh ra chuỗi hợp lệ).
- $G'$  không chứa bất kỳ luật sinh  $\epsilon$  nào.

Ngôn ngữ sinh bởi  $G'$  là  $L - \{\epsilon\}$ , tức là giống  $L$  nhưng không chứa chuỗi rỗng

#### 1.1.4. Luật sinh đơn vị

Luật sinh  $A \rightarrow B$ , với  $A, B \in V$  và cả  $A$  và  $B$  đều là biến, được gọi là luật sinh đơn vị. Mỗi CFL không chứa  $\epsilon$  được sinh ra bởi CFG không có ký hiệu vô ích, không có luật sinh  $\epsilon$  hoặc luật sinh đơn vị. (**Định lý 5.4**)

#### 1.1.5. Dạng chuẩn Chomsky

Dạng chuẩn Chomsky (Chomsky Normal Form - CNF) là một dạng biểu diễn đặc biệt của văn phạm phi ngữ cảnh, trong đó mọi luật sinh phải tuân theo một trong hai dạng:

1.  $A \rightarrow BC$ : Luật sinh có vế phải gồm đúng hai ký hiệu không kết thúc.
2.  $A \rightarrow a$ : Luật sinh có vế phải gồm đúng một ký hiệu kết thúc.

Trong các luật trên:

- +  $A, B, C$  là các ký hiệu phi kết thúc.
- +  $a$  là ký hiệu kết thúc.

Một tính chất quan trọng của CNF là mọi ngôn ngữ phi ngữ cảnh, ngoại trừ ngôn ngữ chứa chuỗi rỗng  $\epsilon$ , đều có thể được sinh ra bởi một văn phạm phi ngữ cảnh được viết dưới dạng chuẩn Chomsky. Điều này được phát biểu qua định lý 5.5:

**Định lý 5.5:** Một ngôn ngữ phi ngữ cảnh bất kỳ không chứa chuỗi rỗng  $\epsilon$  đều được sinh ra bởi một văn phạm mà các luật sinh của nó chỉ có dạng  $A \rightarrow BC$  hoặc  $A \rightarrow a$ , với  $A, B, C$  là biến còn  $a$  là ký hiệu kết thúc.

## 1.2. Công nghệ sử dụng

### 1.2.1. Ngôn ngữ lập trình Python

Python là ngôn ngữ lập trình bậc cao cho các mục đích lập trình đa năng, do Guido van Rossum tạo ra và lần đầu ra mắt vào năm 1991. Python được thiết kế với ưu điểm là dễ đọc, dễ học và dễ nhớ, có ngôn ngữ hình thức sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình và được dùng rộng rãi trong phát triển trí tuệ nhân tạo. Python hỗ trợ nhiều phong cách lập trình khác nhau, từ hướng đối tượng (OOP) đến lập trình hàm hay lập trình thủ tục và từ đó cho phép lập trình viên có thể phát triển nhiều loại ứng dụng khác nhau.

Python cũng hỗ trợ rất nhiều thư viện và framework phong phú, mạnh mẽ cũng như cộng đồng người dùng to lớn giúp cho quá trình lập trình của chúng ta trở nên dễ dàng và nhanh chóng hơn. Trong phạm vi đề tài này, chúng ta sẽ sử dụng thư viện của python gồm tkinter dùng để tạo giao diện.

### **1.2.2. Thư Viện Tkinter**

Thư viện tkinter là một trong những thư viện phổ biến nhất để phát triển giao diện người dùng đồ họa (GUI) trong Python. Tkinter cung cấp các công cụ cần thiết để tạo ra các ứng dụng có giao diện đồ họa trực quan và dễ sử dụng. Dù là một thư viện nhỏ gọn, tkinter vẫn cung cấp đầy đủ các thành phần cần thiết để xây dựng giao diện, bao gồm các widget như button, label, entry, và canvas. Bạn cũng có thể tùy chỉnh giao diện bằng cách sử dụng các thuộc tính và phương thức có sẵn trong các widget này. Với sự linh hoạt và tính tiện dụng, tkinter đã trở thành lựa chọn phổ biến cho việc phát triển các ứng dụng desktop trong Python, từ các ứng dụng nhỏ đến các dự án lớn hơn.

### **1.2.3. Visual Studio Code (VSCode)**

VSCode là một trình soạn thảo mã nguồn được phát triển bởi Microsoft dành cho Windows, Linux và macOS. Nó hỗ trợ chức năng debug, đi kèm với Git, có chức năng nổi bật cú pháp (syntax highlighting), tự hoàn thành mã thông minh, snippets, và cải tiến mã nguồn, VSCode có thể hỗ trợ hầu hết các ngôn ngữ lập trình hiện nay. Với khả năng tùy biến mạnh mẽ cũng như có một kho tiện ích mở rộng đồ sộ sẽ hỗ trợ và giúp ích cho việc lập trình của người dùng 1 cách tối đa. VSCode cũng có một cộng đồng người dùng lớn mạnh sẵn sàng hỗ trợ với bất kỳ vấn đề nào mà chúng ta gặp phải.

## **Chương 2. Phương pháp thực hiện**

### **2.1. Đọc văn phạm phi ngữ cảnh**

Khi đọc một văn phạm phi ngữ cảnh, cần thực hiện các bước sau:

- Xác định tập ký hiệu: Liệt kê các non-terminal, terminal, ký hiệu bắt đầu và các luật sinh.
- Kiểm tra tính hợp lệ: Đảm bảo rằng các luật sinh không vi phạm định nghĩa của CFG.
- Xác định cấu trúc văn phạm: Phân tích cú pháp để hiểu rõ cách các luật sinh hoạt động.
- Nhận diện các vấn đề có thể gặp phải: Văn phạm có thể chứa các ký hiệu vô ích, luật sinh đơn vị hoặc -luật sinh, cần được giản lược hoặc chuẩn hóa

### **2.2. Giản lược văn phạm phi ngữ cảnh**

Quá trình giản lược văn phạm phi ngữ cảnh giúp loại bỏ các phần dư thừa để làm cho văn phạm gọn gàng và hiệu quả hơn. Các bước thực hiện bao gồm:

#### **2.2.1. Loại bỏ ký hiệu vô ích**

- Xóa các ký hiệu không sinh ra chuỗi kết thúc: Xác định các non-terminal không thể sinh ra chuỗi chỉ chứa terminal và loại bỏ chúng.
- Xóa các ký hiệu không đến được từ ký hiệu bắt đầu: Xác định các ký hiệu không thể truy cập từ và loại bỏ chúng.

#### **2.2.2. Loại bỏ luật sinh epsilon**

- Xác định các ký hiệu nullable (có thể sinh ra chuỗi rỗng).
- Tạo luật sinh mới bằng cách loại bỏ các ký hiệu nullable trong các luật sinh.

#### **2.2.3. Loại bỏ luật sinh đơn vị**

- Xác định các cặp đơn vị, trong đó là một luật sinh đơn vị.
- Thay thế bằng các luật sinh của.

### **2.3. Chuẩn hóa văn phạm phi ngữ cảnh thành dạng chuẩn Chomsky**

- Thay thế chuỗi kết thúc trong luật sinh dài bằng non-terminal mới: Nếu có một luật dạng, thay thế bằng một non-terminal mới sao cho.
- Phân tách luật sinh dài: Nếu có luật, thay thành các luật, , tiếp tục cho đến khi tất cả các luật có độ dài không quá 2 non-terminal.



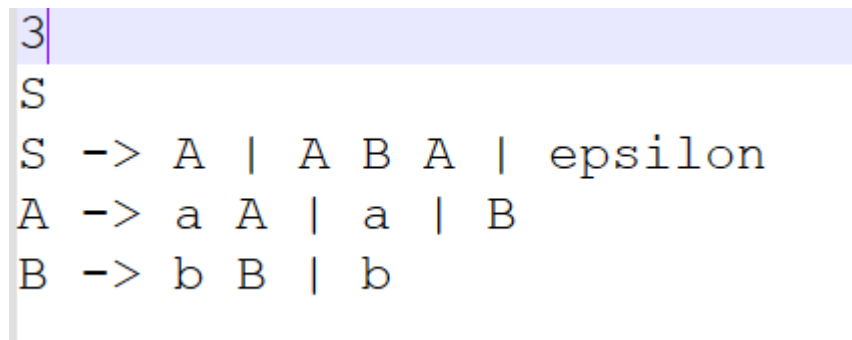
### Chương 3. Thiết kế và cài đặt

Để thực hiện đề tài, chúng ta sử dụng VSCode làm công cụ phát triển chính, với ngôn ngữ lập trình Python và cài đặt tiện ích mở rộng hỗ trợ. Chúng ta tạo tệp mã nguồn `cnf.py` để viết các hàm xử lý và giao diện được viết bằng thư viện Tkinter, các hàm xử lý bao gồm: loại bỏ ký hiệu vô ích, loại bỏ luật sinh epsilon ( $\epsilon$ ), loại bỏ luật sinh đơn vị, và chuyển các quy tắc thành dạng chuẩn chỉ chứa hai biến ( $A \rightarrow B C$ ) hoặc một ký hiệu kết thúc ( $A \rightarrow a$ ). Sau khi hoàn thành mã, chạy và lệnh trên Terminal của VSCode bằng lệnh `python cnf.py` để đảm bảo kết quả chính xác và giao diện cho người dùng.

#### 3.1. Cấu trúc dữ liệu và cấu trúc tệp tin

Cấu trúc dữ liệu lưu trữ văn phạm loại từ điển (dictionary) trong Python, trong đó key chứa ký hiệu không kết thúc, value chứa các quy tắc sinh của ký hiệu không kết thúc tương ứng.

Văn phạm đầu vào được lưu trong file text (.txt), cấu trúc như sau:



```
3
S
S -> A | A B A | epsilon
A -> a A | a | B
B -> b B | b
```

Hình 3.1. Biểu diễn cấu trúc file text (.txt)

Trong đó:

- Dòng đầu tiên : Số luật sinh trong văn phạm
- Dòng thứ hai: Ký hiệu bắt đầu (*start symbol*)
- Từ dòng thứ 3 trở đi: các luật sinh của văn phạm

Trong đó  $\epsilon$  được viết là “epsilon”

### 3.2. Tạo phương thức đọc văn phạm

Tạo phương thức `read_grammar(filename)` dùng để đọc dữ liệu từ file và tách rời về trái và về phải bởi dấu “ $\rightarrow$ ”, với mỗi về phải tách thành các danh sách được ngăn cách “`|`” sau đó tiếp tục tách thành các ký tự riêng lẻ được cách nhau bằng khoảng trắng “”.

```
def read_grammar(filename):
    grammar = {}
    start = None
    terminal = set()

    with open(filename, 'r') as f:
        # Đọc số lượng quy tắc và ký tự bắt đầu
        n = int(f.readline().strip())
        start = f.readline().strip()

        # Đọc các quy tắc ngữ pháp
        for _ in range(n):
            rule = f.readline().strip()
            left, rights = rule.split("→")
            left = left.strip()
            productions = [right.strip().split() for right in rights.split("|")]
            grammar[left] = productions

        for left, productions in grammar.items():
            for prod in productions:
                for symbol in prod:
                    if symbol.islower() and symbol != 'epsilon':
                        terminal.add(symbol)

    terminal = sorted(terminal) # Sắp xếp và chuyển thành list
    return grammar, start, terminal
```

Hình 3.1. Biểu diễn phương thức đọc văn phạm

### 3.3. Tạo phương thức loại bỏ ký hiệu vô ích

Tạo phương thức `eliminate_useless(grammar, start)` để loại bỏ các ký tự vô ích.

```
def eliminate_useless(grammar, start):
    """
    Loại bỏ các ký tự vô ích (không thể sinh ra chuỗi terminal)
    """
    # Tìm các ký tự có thể sinh ra chuỗi terminal
    useful = set()
    flag = True

    while flag:
        flag = False
        # Left chứa ký hiệu không kết thúc
        for left in grammar:
            productions = grammar[left]
            for prod in productions:
                # Hữu ích nếu nó đã trong tập hữu ích hoặc nếu nó nằm trong tập ký hiệu kết thúc
                if all(symbol in useful or symbol.islower() for symbol in prod):
                    if left not in useful:
                        useful.add(left)
                    flag = True

    # Loại bỏ các quy tắc không hữu ích
    new_grammar = {}
    for left, productions in grammar.items():
        if left in useful:
            new_grammar[left] = [prod for prod in productions
                                if all(symbol in useful or symbol.islower() for symbol in prod)]

    return new_grammar
```

Hình 3.2. Biểu diễn phương thức loại bỏ ký hiệu vô ích

Mục đích: Loại bỏ các ký hiệu không kết thúc (non-terminal) không thể sinh ra chuỗi chỉ gồm các ký hiệu kết thúc (terminal), giúp đơn giản hóa văn phạm.

Bước thực hiện:

- Xác định tập ký hiệu hữu ích
- Khởi tạo tập useful rỗng.
- Lặp lại cho đến khi không có sự thay đổi:
  - + Duyệt qua từng non-terminal trong văn phạm.
  - + Nếu có ít nhất một vế phải của nó chỉ chứa terminal hoặc các non-terminal đã nằm trong useful, thêm nó vào useful.
  - + Loại bỏ các quy tắc không hữu ích
  - + Duyệt lại văn phạm, chỉ giữ lại các non-terminal thuộc tập useful.
- Đồng thời, lọc bỏ các vế phải chứa non-terminal không hữu ích.

### 3.4. Tạo phương thức loại bỏ luật sinh epsilon

Tạo phương thức `eliminate_epsilon(grammar, start)` để loại bỏ luật sinh epsilon của văn phạm

```
def eliminate_epsilon(grammar, start):  
    """  
    Loại bỏ các quy tắc epsilon ( $\epsilon$ )  
    """  
    # Tạo ngữ pháp mới không có  $\epsilon$   
    new_grammar = {}  
    for left, productions in grammar.items():  
        new_productions = []  
        for prod in productions:  
            # print(prod[0])  
            if prod[0] == 'epsilon': # Bỏ qua quy tắc  $\epsilon$   
                continue  
            new_productions.append(prod)  
        if len(new_productions) > 0:  
            new_grammar[left] = new_productions  
    return new_grammar
```

Hình 3.3. Biểu diễn phương thức loại bỏ luật sinh epsilon

Mục đích: Loại bỏ các quy tắc có epsilon ( $\epsilon$ ) trong văn phạm mà vẫn giữ nguyên ngữ nghĩa của ngôn ngữ sinh ra.

Bước thực hiện:

- Duyệt qua tất cả các quy tắc trong văn phạm
  - + Duyệt từng non-terminal và danh sách các vế phải của nó.

- + Nếu một vế phải là epsilon, bỏ qua nó (không thêm vào ngữ pháp mới).
- Lưu vào văn phạm mới
  - + Nếu một non-terminal vẫn còn ít nhất một vế phải sau khi loại bỏ  $\epsilon$ , thêm nó vào văn phạm kết quả.

### 3.5. Tạo phương thức loại bỏ luật sinh đơn vị

Tạo phương thức `eliminate_unit(grammar)` để loại bỏ luật sinh đơn vị

```
def eliminate_unit(grammar):
    """
    Loại bỏ các quy tắc đơn vị ( $A \rightarrow B$ )
    """
    new_grammar = {}
    for left, productions in grammar.items():
        new_productions = []
        for prod in productions:
            if len(prod) == 1 and prod[0].isupper():
                # Nếu là quy tắc đơn vị, thêm các quy tắc của ký tự bên phải
                if prod[0] in grammar:
                    new_productions.extend(grammar[prod[0]])
            else:
                new_productions.append(prod)
        new_grammar[left] = new_productions
    return new_grammar
```

Hình 3.4. Biểu diễn phương thức loại bỏ luật sinh đơn vị.

Mục đích: Loại bỏ các quy tắc đơn vị dạng  $A \rightarrow B$ , trong đó A và B đều là non-terminal, giúp đơn giản hóa văn phạm.

Bước thực hiện:

- Duyệt qua từng non-terminal trong văn phạm
  - + Với mỗi A, duyệt qua danh sách vế phải của nó.
  - + Nếu gặp quy tắc đơn vị  $A \rightarrow B$  (B là non-terminal duy nhất trong vế phải):
    - + Thêm tất cả các quy tắc của B vào danh sách vế phải của A.
    - + Nếu không phải quy tắc đơn vị, giữ nguyên vế phải.
- Cập nhật văn phạm mới
  - + Lưu kết quả vào văn phạm mới không chứa quy tắc đơn vị.

### 3.6. Tạo phương thức chuyển các quy tắc thành dạng chuẩn Chomsky

Tạo phương thức `convert_to_cnf(grammar, terminal)` để chuyển thành chuẩn Chomsky

```
def convert_to_cnf(grammar, terminal):
    for left, productions in grammar.items():
        for prod in productions:
            if len(prod) == 1 and prod[0].islower():
                continue
            for i in range(len(prod)):
                if prod[i].islower():
                    position = terminal.index(prod[i])
                    prod[i] = f'S{position}'
    for i in range(len(terminal)):
        grammar[f'S{i}'] = [[terminal[i]]]
    new_grammar = {}
    new_symbols = 0
    def get_new_symbol(symbol):
        nonlocal new_symbols
        new_symbols += 1
        return f'{symbol}{new_symbols}'
    def get_sybol(symbol, index):
        return f'{symbol}{index}'
    new_prod = []
    for left, productions in grammar.items():
        new_productions = []
        for prod in productions:
            if len(prod) <= 2:
                new_productions.append(prod)
            else:
                current = left
                for i in range(len(prod) - 2):
                    new_symbol = get_new_symbol('X')
                    new_prod.append([prod[i + 1], prod[i + 2]])
                    current = new_symbol
                new_productions.append([prod[-1], current])
        new_grammar[left] = new_productions
    for i in range(len(new_prod)):
        new_grammar[get_sybol('X', i+1)] = [new_prod[i]]
    return new_grammar
```

Hình 3.4. Biểu diễn phương thức chuyển về cnf

Mục đích: Chuyển đổi một văn phạm bất kỳ sang CNF (Chomsky Normal Form), trong đó mỗi quy tắc chỉ có một trong hai dạng:

- $A \rightarrow B C$  (hai non-terminal)
- $A \rightarrow a$  (một terminal)

Bước thực hiện:

Bước 1: Chuyển đổi các terminal trong vế phải có độ dài lớn hơn 1

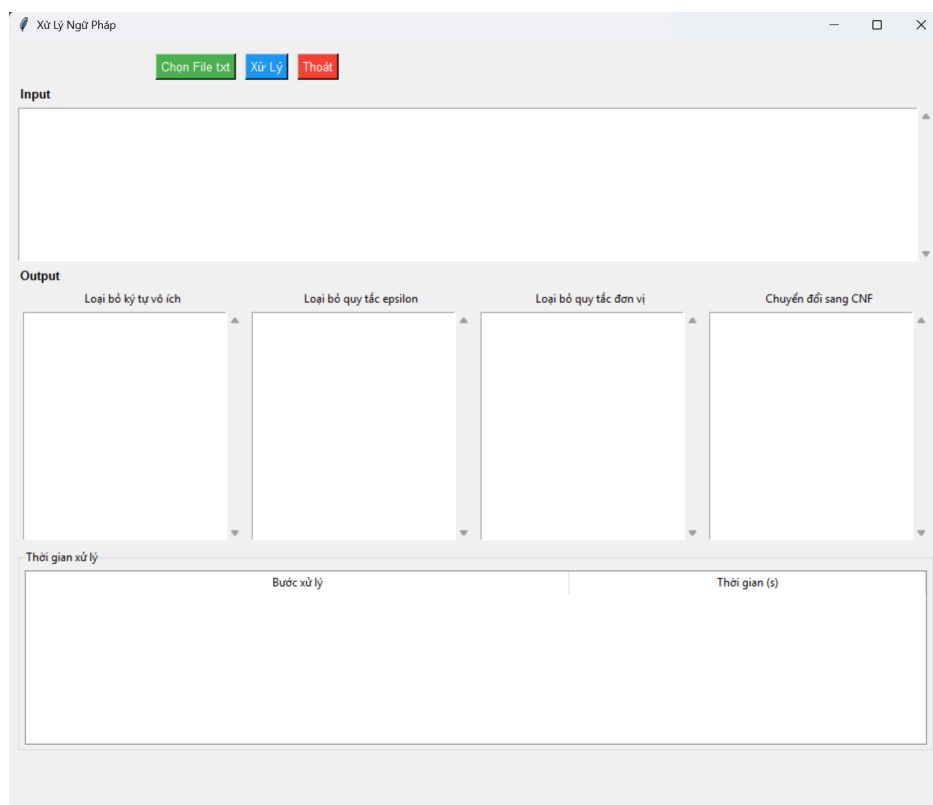
- + Duyệt qua từng quy tắc trong văn phạm.
- + Nếu một quy tắc có chứa terminal cùng với non-terminal thay thế terminal bằng một non-terminal mới  
(ví dụ:  $S \rightarrow aA \mid a$  được chuyển thành  $S \rightarrow S_1A \mid a$ ,  $S_1 \rightarrow a$ ).

Bước 2: Chuyển đổi các quy tắc có vế phải dài hơn 2

- + Nếu một quy tắc có vế phải chứa nhiều hơn hai ký hiệu (ví dụ:  $A \rightarrow BCD$ ), ta tạo ra các non-terminal trung gian để chia nhỏ.
- + Tiếp tục quá trình này cho đến khi tất cả các quy tắc đều có dạng đúng của CNF.

### 3.7. Thiết kế giao diện bằng thư viện Tkinter

Sử dụng thư viện Tkinter của Python để thực hiện viết phần giao diện cho demo.



Hình 3.5. Giao diện hệ thống ban đầu sau khi được biên dịch

Dữ liệu văn phạm ta có thể đọc file bằng cách nhấn vào nút “Chọn File txt” hoặc có thể nhập trực tiếp lên ô trống có sẵn.

Hình 3.6. Giao diện hệ thống sau khi được điền dữ liệu

Bước xử lý	Thời gian (s)
Đọc ngữ pháp	0.0024
Loại bỏ ký tự vô ích	0.0001
Loại bỏ quy tắc epsilon	0.0000
Loại bỏ quy tắc đơn vị	0.0000
Chuyển đổi sang CNF	0.0001
Tổng thời gian	0.0003

Hình 3.7. Giao diện hệ thống sau khi xử lý chuyển văn phạm về dạng chuẩn Chomsky

## PHẦN III. ĐÁNH GIÁ KIỂM THỬ

### 1. Mục tiêu

Độ chính xác: Đảm bảo hệ thống xây dựng có thể đọc được văn phạm và chuyển văn phạm đúng về dạng chuẩn Chomsky.

Hiệu suất: Đảm bảo hệ thống vận hành hiệu quả, phản hồi nhanh và sử dụng tài nguyên hợp lý.

Thiết kế sáng tạo: Đánh giá mức độ sáng tạo trong giao diện và trải nghiệm người dùng, tối ưu giao diện để mang lại sự tiện lợi nhất.

### 2. Kiểm thử

Yêu cầu: Đọc văn phạm phi ngữ cảnh đã cho và chuyển văn phạm về dạng chuẩn Chomsky CNF

#### 2.1 Kiểm thử lần 1

Dữ liệu:

3

S

$S \rightarrow A \mid A B A$

$A \rightarrow a A \mid a \mid B$

$B \rightarrow b B \mid b$

Kết quả khi chạy qua hệ thống:

Loại bỏ các ký tự vô ích:

$S \rightarrow A \mid A B A$

$A \rightarrow a A \mid a \mid B$

$B \rightarrow b B \mid b$

Loại bỏ các luật sinh epsilon:

$S \rightarrow A \mid A B A$

$A \rightarrow a A \mid a \mid B$

$B \rightarrow b B \mid b$

Loại bỏ luật sinh đơn vị:

$S \rightarrow a A \mid a \mid B \mid A b A$

$A \rightarrow a A \mid a \mid b B \mid b$

$B \rightarrow b B \mid b$



Chuyển sang dạng chuẩn Chomsky (CNF):

$$S \rightarrow S_0 A \mid a \mid B \mid A X_1$$
$$A \rightarrow S_0 A \mid a \mid S_1 B \mid b$$
$$B \rightarrow S_1 B \mid b$$
$$S_0 \rightarrow a$$
$$S_1 \rightarrow b$$
$$X_1 \rightarrow S_1 A$$

## 2.2 Kiểm thử lần 2

Dữ liệu:

8

S

$$S \rightarrow A B C \mid D E$$
$$A \rightarrow a \mid B C \mid \text{epsilon}$$
$$B \rightarrow b \mid C D \mid \text{epsilon}$$
$$C \rightarrow c \mid D E$$
$$D \rightarrow d \mid E F$$
$$E \rightarrow e \mid F G$$
$$F \rightarrow f \mid G H$$
$$G \rightarrow g \mid H I$$

Kết quả khi chạy qua hệ thống:

Loại bỏ các ký tự vô ích:

$$S \rightarrow A B C \mid D E$$
$$A \rightarrow a \mid B C \mid \text{epsilon}$$
$$B \rightarrow b \mid C D \mid \text{epsilon}$$
$$C \rightarrow c \mid D E$$
$$D \rightarrow d \mid E F$$
$$E \rightarrow e \mid F G$$
$$F \rightarrow f$$
$$G \rightarrow g$$

Loại bỏ các luật sinh epsilon:

$$S \rightarrow A B C \mid D E$$
$$A \rightarrow a \mid B C$$
$$B \rightarrow b \mid C D$$
$$C \rightarrow c \mid D E$$

$D \rightarrow d \mid EF$

$E \rightarrow e \mid FG$

$F \rightarrow f$

$G \rightarrow g$

Loại bỏ luật sinh đơn vị:

$S \rightarrow ABC \mid DE$

$A \rightarrow a \mid BC$

$B \rightarrow b \mid CD$

$C \rightarrow c \mid DE$

$D \rightarrow d \mid EF$

$E \rightarrow e \mid FG$

$F \rightarrow f$

$G \rightarrow g$

Chuyển sang dạng chuẩn Chomsky (CNF):

$S \rightarrow AX1 \mid DE$

$A \rightarrow a \mid BC$

$B \rightarrow b \mid CD$

$C \rightarrow c \mid DE$

$D \rightarrow d \mid EF$

$E \rightarrow e \mid FG$

$F \rightarrow f$

$G \rightarrow g$

$X1 \rightarrow BC$

## **PHẦN IV . KẾT LUẬN**

### **1. Kết quả đạt được**

Trong nghiên cứu này nhóm đã đạt được những yêu cầu nhất định về các chức năng cơ bản của đề tài như demi đọc văn phạm phi ngữ cảnh và chuyển văn phạm thành dạng chuẩn Chomsky. Ngoài ra, tạo được một giao diện đơn giản bằng thư viện tkinter để hiển thị được nhập xuất văn phạm, các kết quả của từng bước đọc và chuyển văn phạm về dạng chuẩn Chomsky

### **2. Hướng phát triển**

Trong tương lai, đề tài có thể được phát triển theo các hướng sau:

- Tối ưu hóa: Tối ưu hóa hiệu suất và xử lý ngữ cảnh phức tạp
- Mở rộng: có thể mở rộng chương trình này bằng cách thêm vào các chức năng như chuyển văn phạm thành dạng chuẩn Greibach (GNF), mở rộng cho văn phạm có ký hiệu hạn.
- Thiết kế giao diện người dùng: thiết kế một giao diện đồ họa (GUI) trực quan và hoàn chỉnh để có thể sử dụng chương trình một cách chính chu và phù hợp với những tính năng phát triển thêm về sau.
- Tích hợp vào các ứng dụng thực tế: nghiên cứu cách tích hợp chương trình vào các ứng dụng thực tế như trình biên dịch, công cụ phân tích ngôn ngữ, công cụ kiểm tra lỗi, cũng như ứng dụng trong xử lý ngôn ngữ tự nhiên.

## TÀI LIỆU THAM KHẢO

- [1] Võ Huỳnh Trâm. (2009). Giáo trình Tin Học Lý Thuyết. Đại học Cần Thơ.
- [2] John E. Hopcroft & Jeffrey D. Ullman. (1979). Introduction to Automata Theory, Languages and Computation. Addison Wesley Publishing Company, Inc.
- [3] John C. Martin. (2011). Introduction to Languages and the Theory of Computation (4th ed.). McGraw Hill.