 Trường ĐHBK TPHCM

Khoa CNTT

LUẬN VĂN TỐT NGHIỆP

ĐỀ TÀI:

CHỨNG MINH TỰ ĐỘNG LOGIC VỊ TỪ BẰNG PHƯƠNG PHÁP SUY LUẬN TỰ NHIÊN

GVHD : NGUYỄN THANH SƠN

SVTH : NGUYỄN DUY NHẤT – 50301940

NGUYỄN NGỌC PHƯỚC – 50302178

GIỚI THIỆU ĐỀ TÀI

**Logic** hay **luận lý học**, từ tiếng Hy Lạp cổ điển λόγος (logos), nghĩa nguyên thủy là *từ ngữ*, hoặc *điều đã được nói*, (nhưng trong nhiều ngôn ngữ châu Âu đã trở thành có ý nghĩa là *suy nghĩ* hoặc *lập luận* hay *lý trí*). *Logic* thường được nhắc đến như là một ngành nghiên cứu về tiêu chí đánh giá các luận cứ, mặc dù định nghĩa chính xác của logic vẫn là vấn đề còn đang được bàn cãi giữa các triết gia. Tuy nhiên khi môn học được xác định, nhiệm vụ của nhà logic học vẫn như cũ: làm đẩy mạnh tiến bộ của việc phân tích các suy luận có hiệu lực và suy luận ngụy biện để người ta có thể phân biệt được luận cứ nào là hợp lý và luận cứ nào có chỗ không hợp lý.

Theo truyền thống, logic được nghiên cứu như là một nhánh của triết học. Kể từ giữa thế kỉ 19 logic đã thường được nghiên cứu trong toán học và luật. Gần đây nhất logic được áp dụng vào khoa học máy tính và trí tuệ nhân tạo. Là một ngành khoa học hình thức, logic nghiên cứu và phân loại cấu trúc của các khẳng định và các lý lẽ, cả hai đều thông qua việc nghiên cứu các hệ thống hình thức của việc suy luận và qua sự nghiên cứu lý lẽ trong ngôn ngữ tự nhiên. Tầm bao quát của logic do vậy là rất rộng, đi từ các đề tài cốt lõi như là nghiên cứu các lý lẽ ngụy biện và nghịch lý, đến những phân tích chuyên gia về lập luận, chẳng hạn lập luận có xác suất đúng và các lý lẽ có liên quan đến quan hệ nhân quả. Ngày nay, logic còn được sử dụng phổ biến trong lý thuyết lý luận.

Qua suốt quá trình lịch sử, đã có nhiều sự quan tâm trong việc phân biệt lập luận tốt và lập luận không tốt, và do đó logic đã được nghiên cứu trong một số dạng ít nhiều là quen thuộc đối với chúng ta. Logic Aristotle chủ yếu quan tâm đến việc dạy lý luận thế nào cho tốt, và ngày nay vẫn được dạy với mục đích đó, trong khi trong logic toán học và triết học phân tích (*analytical philosophy*) người ta nhấn mạnh vào logic như là một đối tượng nghiên cứu riêng, và do vậy logic được nghiên cứu ở một mức độ trừu tượng hơn.

Các quan tâm về các loại logic khác nhau giải thích rằng logic không phải là được nghiên cứu trong chân không. Trong khi logic thường có vẻ tự cung cấp sự thúc đẩy chính nó, môn học này phát triển tốt nhất khi lý do mà chúng ta quan tâm đến logic được đặt ra một cách rõ ràng.

Môn Logic như được nghiên cứu ngày nay rất khác với môn học đã được nghiên cứu trước đây, và sự khác biệt chính là sự phát minh của **logic vị từ**. Trong khi logic tam đoạn luận của Aristote định ra những dạng thức cho những phần có liên quan với nhau trong mỗi phán đoán, logic vị từ cho phép các câu được phân tích thành chủ đề và các luận cứ theo nhiều cách khác nhau, do vậy cho phép logic vị từ giải quyết được vấn đề tổng quát hóa nhiều lần - vấn đề đã làm bối rối các nhà logic học thời trung cổ. Với logic vị từ, lần đầu tiên, các nhà logic học đã có khả năng đưa ra các phép lượng hóa (*quantifiers*) đủ tổng quát để diễn tả mọi luận cứ có mặt trong ngôn ngữ tự nhiên.

Sự khám phá ra logic vị từ thường được coi là công của Gottlob Frege, người cũng được xem là một trong những sáng lập viên của ngành triết học phân tích, nhưng dạng phát biểu có hệ thống thông dụng nhất ngày nay của logic vị từ là logic bậc nhất (*first-order logic*) được trình bày trong cuốn sách Các nguyên lý về logic lý thuyết (*Grundzüge der theoretischen Logik*) của David Hilbert và Wilhelm Ackermann vào năm 1928. Tính tổng quát có tính phân tích của logic vị từ cho phép hình thức hóa toán học và đẩy mạnh nghiên cứu về lý thuyết tập hợp, cho phép sự phát triển của cách tiếp cận của Alfred Tarski đối với lý thuyết mô hình; và không quá lời khi nói rằng nó là nền tảng của logic toán học hiện đại.

Hệ thống nguyên thủy của Frege về logic vị từ không phải là bậc nhất mà là bậc hai. Logic bậc hai được bảo vệ mạnh mẽ nhất bởi George Boolos và Stewart Shapiro (trước các phê phán của Willard Van Orman Quine và những người khác).

# LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn thầy Nguyễn Thanh Sơn đã bỏ công sức, tận tình hướng dẫn, giúp đỡ, theo sát và động viên chúng em hoàn thành đề tài này.

Cảm ơn các thầy cô giảng viên trong khoa Công Nghệ Thông Tin đã truyền đạt kiến thức cho chúng em trong thời gian học ở trường Đại học Bách Khoa.

Cảm ơn gia đình tôi đã động viên tôi, là nguồn cỗ vũ mạnh mẽ, là chỗ dựa vững chắc cho tôi tự tin thực hiện đề tài này.

Báo cáo luận văn này còn nhiều thiếu sót về nội dung và hình thức trình bày. Chúng em kính mong thầy cùng hội đồng góp ý để có thể hoàn thiện hơn.

**MỤC LỤC**

PHẦN I: TỔNG QUAN VỀ ĐỀ TÀI

1. Nội dung đề tài: Chứng minh logic vị từ bằng phương pháp suy luận tự nhiên.
2. Yêu cầu đề tài:
   * Input: một hoặc nhiều biểu thức logic vị từ và kết quả cần chứng minh
   * Output: Quá trình chứng minh từ các biểu thức đã cho ra kết quả cần chứng minh bằng phương pháp suy luận tự nhiên.

Ví dụ:

PHẦN II: HIỆN THỰC CHƯƠNG TRÌNH

### INPUT

* 1. – BNF BNF Logic vị từ

Sentence → AtomicSentence | Sentence Connective Sentence

| Quantifier Variable, ... Sentence | ¬ Sentence | (Sentence)

AtomicSentence → Predicate(Term, …) | Term = Term

Term → Function(Term, …) | Constant | Variable

Connective → ∧ | ∨ | 🡪

Quantifier → ∀ | ∃

Constant → A, B, C, X1 , X2, Jim, Jack

Variable → a, b, c, x1 , x2, counter, position

Predicate → Adjacent-To, Younger-Than,

Function → Father-Of, Square-Position, Sqrt, Cosine

* 1. – Input của chương trình

Input của chương trình là một đề toán logic vị từ gồm có 2 phần: CONDITION và GOAL. Theo cấu trúc:

CONDITION |- GOAL

* CONDITION: gồm một hoặc nhiều công thức logic vị từ theo cấu trúc của một BNF đã cho ở phần 1. Các công thức này được ngăn cách nhau bởi dấu phẩy ‘,’ .
* GOAL: gồm một công thức logic vị từ cũng theo cấu trúc BNF trên.

### SCANNER

Nhận dạng chuỗi nhập và chuyển chuỗi nhập thành dạng Token, đồng thời cho biết vị trí của Token đó. Các Token và các lexeme tương ứng được qui ước như sau:

LGC\_VAR = ‘x’, ‘y’, ‘z’, …(chuỗi bắt đầu bằng kí tự viết thường)

LGC\_INTERSECTION\_OP = ‘and’, ‘AND’, ‘∧’

LGC\_UNION\_OP = ‘or’, ‘OR’, ‘∨’

LGC\_NEGATION\_OP = ‘not’, ‘NOT’, ‘!’, ‘~’

LGC\_MAPPING\_OP = ‘map’, ‘MAP’, ‘=>’, ‘→’

LGC\_EQUIVALENT\_OP = ‘<=>’ , ‘<->’

LGC\_LEFTPAR = ‘ (‘

LGC\_RIGHTPAR = ‘)’

LGC\_BOOLEANLITERAL = ‘true’, ‘false’, ‘TRUE’, ‘FALSE’

LGC\_CONTRADITION\_OP = ‘<>’

LGC\_ALL\_OP = ‘V-‘ , ‘all’ , ‘ALL’

LGC\_EXIST\_OP = ‘ -]’ , ‘exists’ , ‘EXISTS’

LGC\_RESULT\_OP = ‘=|’ , ‘|-‘

LGC\_COMMA = ‘,’

LGC\_CON = ‘P’, ‘Q’, …( chuỗi bắt đầu bằng kí tự viết hoa)

LGC\_NIL = 15,

LGC\_ERROR = others

Ví dụ: Chuỗi nhập là:

p(t) , all x p(x)→ q(x) |- q(t)

Các Token tương ứng là:

Token.LGC\_VAR Lexeme: p CharStart:2 CharFinish:2

Token.LGC\_LEFTPAR Lexeme: ( CharStart:3 CharFinish:3

Token.LGC\_VAR Lexeme: t CharStart:4 CharFinish:4

Token.LGC\_RIGHTPAR Lexeme: ) CharStart:5 CharFinish:5

Token.COMMA Lexeme: , CharStart:7 CharFinish:7

Token.LGC\_ALL\_OP Lexeme: all CharStart:9 CharFinish:11

Token.LGC\_VAR Lexeme: x CharStart:13 CharFinish:13

Token.LGC\_VAR Lexeme: p CharStart:15 CharFinish:15

Token.LGC\_LEFTPAR Lexeme: ( CharStart:16 CharFinish:16

Token.LGC\_VAR Lexeme: x CharStart:17 CharFinish:17

Token.LGC\_RIGHTPAR Lexeme: ) CharStart:18 CharFinish:18

Token.LGC\_MAPPING\_OP Lexeme: -> CharStart:20 CharFinish:21

Token.LGC\_VAR Lexeme: q CharStart:23 CharFinish:23

Token.LGC\_LEFTPAR Lexeme: ( CharStart:24 CharFinish:24

Token.LGC\_VAR Lexeme: x CharStart:25 CharFinish:25

Token.LGC\_RIGHTPAR Lexeme: ) CharStart:26 CharFinish:26

Token.LGC\_RESULT\_OP Lexeme: |- CharStart:28 CharFinish:29

Token.LGC\_VAR Lexeme: q CharStart:31 CharFinish:31

Token.LGC\_LEFTPAR Lexeme: ( CharStart:32 CharFinish:32

Token.LGC\_VAR Lexeme: t CharStart:33 CharFinish:33

Token.LGC\_RIGHTPAR Lexeme: ) CharStart:34 CharFinish:34

Token.LGC\_NIL Lexeme: $ CharStart:35 CharFinish:35

### PARSER

* Nhận kết quả từ bộ SCANNER
* Kiểm tra cú pháp chuỗi nhập có đúng với văn phạm của logic vị từ hay không
  + Nếu đúng cú pháp thì sẽ đưa vào bảng cấu trúc dữ liệu (bảng TERMVECTOR)
  + Nếu sai thì sẽ ngừng chương trình, đồng thời báo lỗi sai và vị trí sai đầu tiên để người dùng có thể dễ dàng sửa chữa.

Các qui ước như sau

* CONDITION
  + Các tiên đề ngăn cách nhau bởi dấu phẩy và được gọi là 1 câu (sentence).
  + Một câu sẽ có cú pháp theo cấu trúc văn phạm BNF logic vị từ đã nêu ở trên.
* GOAL
  + Là một câu.

Ví dụ 1: Chuỗi nhập là:

p(t) , all x p(x)→ q(x) |- q(t)

Kết quả:

Sẽ gồm 3 câu như sau:

BeginSentence

p(t)

EndSentence

BeginSentence

all x p(x)→ q(x)

EndSentence

BeginSentence

q(t)

EndSentence

Có nghĩa là 3 câu trên sẽ được đưa vào cấu trúc dữ liệu

Ví dụ 2: Chuỗi nhập là:

p(t) , all x p(x **→** q(x) |- q(t) // (thiếu dấu đóng ngoặc tại biến x)

Kết quả:

Unexpected Token.LGC\_MAPPING\_OP Lexeme: -> CharStart:19 CharFinish:20

Chương trình sẽ báo lỗi sai tại Token. LGC\_MAPPING\_OP tức là dấu : → (vì thiếu đóng ngoặc)

### CONVERTER

* Chuyển các toán tử logic thành dạng hàm

Qui ước:

* + A ∨ B thành or(A,B)
  + A∧ B thành and(A,B)
  + A→B thành map(A,B)
  + ¬ A thành not(A)
* Độ ưu tiên của các toán tử
  + Theo tứ tự: ( ) ¬ ∧ ∨ 🡪

Ví dụ:

A → ¬B ∧ (¬C ∨ D) thành map ( A, and( not(B), or( not(C),D)))

### TERM VECTOR

4.1 – Mô hình **Warren Abstract Machine**

4.1.1 – Lịch sử

* Vào năm 1983, David H. D. Warren đã thiết kế ra mô hình abstract machine để thực thi chương trình Prolog gồm:

+ Kiến trúc bộ nhớ

+ Tập lệnh

* Bản thiết kế này được biết đến với tên là **Warren Abstract Machine (WMA)** và đã trở thành một chuẩn trong chương trình dịch của Prolog trong thực tế.

4.1.2 – Kiến trúc bộ nhớ trong mô hình **Warren Abstract Machine**

- Dùng cấu trúc dữ liệu HEAP, là một array các data cell. Địa chỉ của cell chính là chỉ số index trong array.

- Chỉ có 2 loại dữ liệu được lưu trong data cell. Đó là variable và function

### INFERENCE

5.1 – Tập luật

* Trong suy luận tự nhiên logic vị từ, tập luật được chia thành 2 loại
  + Elimination rules : gồm những luật dùng để phân rã một biểu thức thành nhiều biểu thức con
    - A ∧ B ├─ A (e ∧ 1)
    - A ∧ B ├─ B (e ∧ 2)
  + Introduction rules : gồm những luật dùng để sinh ra một biểu thức mới từ những biểu thức đang có
    - A , A→ B ├─ B ( i→)

5.2 – Giải thuật chính

* 5.2.1 - Elimination
  + Tìm 1 công thức hoặc 1 cặp công thức mà có thể áp dụng luật elimination
  + Áp luật elimination theo độ ưu tiên : ∧ ∨ → ¬ ∀ ∃
  + Một vài tiên đề cần phải dùng kĩ thuật riêng để giải (sẽ được nói rõ hơn trong phần sau
  + Cập nhật lại list\_proof
* 5.2.2 – Introduction
  + Gọi ∑ = {P0,…,Pn} : tập hợp các công thức trong CONDITION   
     Ω : tập hợp các công thức trong GOAL
  + Nếu Gn không phải là ⊥ và đồng thời thuộc các dạng bên dưới thì ta có phép biến đổi tương đương sau:
  + ∑ ⊢ Ω ,F = ∑ , ¬F ⊢Ω ,F, ⊥
  + ∑ ⊢ Ω , ¬F = ∑ , F ⊢ Ω , ¬F, ⊥
  + ∑ ⊢ Ω , A ∧ B = ∑ ⊢ Ω , A ∧ B, A, B
  + ∑ ⊢ Ω , A ∨ B = ∑ ⊢ Ω , A ∨ B, A (failed 5)
  + ∑ ⊢ Ω , A ∨ B = ∑ ⊢ Ω , A ∨ B, B (failed 6)
  + ∑ ⊢ Ω , A ∨ B = ∑ , ¬ (A ∨ B) ⊢ Ω , A ∨ B, ⊥
  + ∑ ⊢ Ω , A →B = ∑ ,A ⊢ Ω , A→B, B
  + ∑ ⊢ Ω , ∀x A(x) = ∑ ⊢ Ω , ∀ x A(x) , A( xo/x) (any xo)
  + ∑ ⊢ Ω , ∃x A(x) = ∑ ⊢ Ω , ∃x A(x) , A( xo/x) (relative xo)
* 5.2.3 – Contradition deduction
  + Khi không thể áp dụng luật elimination và goal hiện tại đang là ⊥ (nghĩa là cũng không thể áp dụng luật introduction) thì procedure này sẽ được gọi với sự tương đương sau:
  1. ∑ ,¬F ⊢ Ω , ⊥ = ∑ ,¬F ⊢ Ω, ⊥, F
  2. ∑ ,A ∨ B ⊢ Ω , ⊥ = ∑ ,A ∨ B ⊢ Ω, ⊥, ¬ A
  3. ∑ ,A → B ⊢ Ω , ⊥ = ∑ ,A → B ⊢ Ω, ⊥, A
* 5.2.4 – Matching
  + Nếu goal là 1 term:
    - Unification ( A, A)
  + Nếu goal là 1 mâu thuẫn (contradition):
    - Unification ( A, ¬ A)
* 5.2.5 – Rule Apply
  + Áp luật R cho phân giải Gn-1 từ Gn
  + Cập nhật Goal\_list và Proof\_list
  + Xóa goal Gn
  + Xóa tất cả các giả thiết.

TÀI LIỆU THAM KHẢO