ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
Khoa công nghệ thông tin

Cơ sở trí tuệ nhân tạo

Báo cáo bài tập lab 2: LOGIC

Lớp: CQ2022/4

Sinh viên: Nguyễn Tường Bách Hỷ - 22120455

Ngày nộp: Ngày 27 tháng 11 năm 2024

Mục lục

1 Giới thiệu

Bài toán kiểm tra khả năng suy diễn logic là một trong những vấn đề quan trọng của Trí tuệ Nhân tạo. Trong bối cảnh này, thuật toán **PL-Resolution** được sử dụng để xác định liệu một truy vấn (query) có thể được suy diễn từ cơ sở tri thức (Knowledge Base - KB) hay không. Thuật toán dựa trên việc phân giải các mệnh đề logic được biểu diễn dưới dạng CNF (Conjunctive Normal Form).
Chương trình được thiết kế để tự động thực hiện các bước sau:

- Đọc cơ sở tri thức và truy vấn từ tệp đầu vào.
- Chuyển đổi các mệnh đề logic về dạng CNF.

- Thực hiện thuật toán PL-Resolution, bao gồm việc phân giải các mệnh đề để kiểm tra mâu thuẫn.
- Ghi lại quá trình phân giải và kết luận vào tệp đầu ra.

Cấu trúc chương trình bao gồm các hàm chính trong tệp điều khiển `main.py` cùng các hàm hỗ trợ được xây dựng trong tệp `logic.py`. Trong các phần tiếp theo, các hàm và cách hoạt động của chương trình sẽ được phân tích chi tiết.

2 Danh sách công việc

STT	Đặc tả tiêu chí	Tiến độ
1	Đọc dữ liệu đầu vào và lưu trong cấu trúc dữ liệu phù hợp	Done
2	Cài đặt giải thuật hợp giải trên logic mệnh đề	Done
3	Các bước suy diễn phát sinh đủ mệnh đề và kết luận đúng	Done
4	Tuân thủ mô tả định dạng của đề bài	Done
5	Báo cáo test case và đánh giá	Done

3 Cấu trúc mã nguồn

Mã nguồn được tổ chức theo ba phần chính:

1. Các lớp đại diện cho biểu thức logic.
2. Các hàm chuyển đổi và đơn giản hóa biểu thức.
3. Thuật toán PL-Resolution và xử lý tệp dữ liệu.

3.1 Lớp biểu diễn logic mệnh đề

3.1.1 Lớp `Atom`

Lớp `Atom` biểu diễn biến logic. Các đặc điểm chính:

- `__init__(name)`: Hàm khởi tạo nhận tên biến.
- `__repr__()`: Trả về biểu diễn chuỗi của biến.
- `__eq__(other)`: Xác định hai biến bằng nhau nếu tên của chúng trùng khớp.
- `__hash__()`: Tính toán giá trị băm dựa trên ký tự Unicode của tên biến.

3.1.2 Lớp `Not`

Lớp `Not` biểu diễn phủ định ($\neg$) của một biến hoặc biểu thức logic.

- `__init__(operand)`: Khởi tạo với một toán hạng. Nếu toán hạng đã là phủ định, lớp sẽ giảm thiểu bằng cách loại bỏ phủ định kép.
- `__repr__()`: Trả về biểu diễn dạng chuỗi với tiền tố dấu trừ (`-`).
- `__hash__()`: Tính toán giá trị băm để đảm bảo lớp phủ định có thể sử dụng trong các tập hợp.

3.1.3 Lớp `And` và `Or`

Hai lớp này biểu diễn phép hội ($\wedge$) và phép tuyển ($\vee$) của nhiều toán hạng.

- `__init__(...)`: Tự động làm phẳng các biểu thức lồng nhau (e.g., `And(And(a, b), c)` thành `And(a, b, c)`).
- `__repr__()`: Biểu diễn chuỗi với các phép nối `AND` hoặc `OR`.
- `simplify()`: Hàm giảm thiểu loại bỏ các giá trị hằng (`True`, `False`) và xử lý phân phối.

3.2 Hàm xử lý biểu thức logic

3.2.1 `simplify`

Hàm này áp dụng các quy tắc giảm thiểu:

- Loại bỏ các phần tử dư thừa.
- Xử lý phân phối để đưa biểu thức về dạng tối giản.

3.2.2 `to_cnf`

Hàm chuyển đổi biểu thức logic thành dạng chuẩn tắc chập (CNF). Quy trình:

- Loại bỏ phủ định kép.
- Áp dụng luật phân phối với phép tuyển và phép hội.

3.2.3 `convert_to_clauses`

Chuyển đổi biểu thức CNF thành danh sách các mệnh đề (clauses), phù hợp với thuật toán PL-Resolution.

3.3 Thuật toán PL-Resolution

Hàm `pl_resolution(query, clauses)` kiểm tra tính đúng của mệnh đề `query` dựa trên cơ sở tri thức `clauses` với hàm bổ trợ là hàm `resolve`.

3.3.1 `resolve`

Hàm `resolve` thực hiện bước phân giải trong thuật toán **PL-Resolution**. Mục đích của hàm này là kết hợp hai mệnh đề để loại bỏ một literal và phủ định của nó, tạo ra một mệnh đề mới mà không chứa literal bị loại bỏ.

Hàm `resolve` là bước cốt lõi trong thuật toán **PL-Resolution**, cho phép tìm kiếm sự mâu thuẫn giữa các mệnh đề bằng cách loại bỏ các literal đối lập. Nếu việc phân giải dẫn đến một mệnh đề rỗng (`Atom("")`), điều này chứng tỏ rằng tập hợp mệnh đề là không nhất quán và câu hỏi đã được chứng minh đúng.

Hàm `resolve` giúp tối ưu quá trình tìm kiếm mâu thuẫn trong tập mệnh đề, tuy nhiên, hiệu suất của nó có thể giảm khi số lượng mệnh đề tăng lên, do cần kiểm tra tất cả các cặp mệnh đề và tính toán sự tương thích giữa chúng. Ngoài ra, việc sắp xếp lại các literal trong mệnh đề để tránh trùng lặp và đảm bảo tính nhất quán cũng tốn thời gian tính toán.

3.3.2 `pl_resolution`

1. Phủ định `query` và thêm vào tập mệnh đề.
2. Thực hiện kết hợp từng cặp mệnh đề để tìm ra mâu thuẫn.
3. Nếu tìm được mệnh đề rỗng (`Atom("{}")`), `query` là đúng.

4 Xử lý dữ liệu đầu vào/đầu ra

Hàm `readKB` đọc tệp cơ sở tri thức từ thư mục `INPUT`, trong khi hàm `write_output` ghi kết quả vào thư mục `OUTPUT`. Tệp đầu vào bao gồm:

- Mệnh đề cần kiểm chứng.
- Số lượng mệnh đề và danh sách các mệnh đề trong cơ sở tri thức.

5 Testcases

Testcase 1:

Input.txt	**Output.txt**	**Ghi chú**
-A	3	
4	-A	(-A OR B) hợp giải với (-B)

-A OR B	(B	-A OR B) hợp giải với (A)
B OR -C	(-C	B OR -C) hợp giải với (-B)
A OR -B OR C	4	
-B	-B OR C	(A OR -B OR C) hợp giải với (-A)
	A OR C	(A OR -B OR C) hợp giải với (B)
	A OR -B	(A OR -B OR C) hợp giải với (-C)
	{}	(-B) hợp giải với (B)
	YES	KB entails α vì tồn tại mệnh đề rỗng trong KB

Testcase 2:

Input.txt	**Output.txt**	**Ghi chú**
A	2	
4	-C	(B OR -C) hợp giải với (-B)
-A OR B	-B OR C	(A OR -B OR C) hợp giải với (-A)
B OR -C	2	
A OR -B OR C	-A OR C	(-A OR B) hợp giải với (-B OR C)
-B	A OR -B	(A OR -B OR C) hợp giải với (-C)
	1	
	A OR -C	(B OR -C) hợp giải với (A OR -B)
	0	
	NO	KB KHÔNG entail α vì không phát sinh được mệnh đề mới và không tìm thấy mệnh đề rỗng

Testcase 3:

Input.txt	**Output.txt**	**Ghi chú**
U	6	
5	P OR R	(P OR Q) hợp giải với (-Q OR R)
P OR Q	P OR S	(P OR Q) hợp giải với (-Q OR S)
-Q OR R	Q OR U	(P OR Q) hợp giải với (-P OR U)
-Q OR S	-Q OR U	(-Q OR R) hợp giải với (-R OR U)
-P OR U	-P	(-P OR U) hợp giải với (-U)
-R OR U	-R	(-R OR U) hợp giải với (-U)
	9	
	P OR U	(P OR Q) hợp giải với (-Q OR U)
	Q	(P OR Q) hợp giải với (-P)
	R OR U	(-Q OR R) hợp giải với (Q OR U)
	-Q	(-Q OR R) hợp giải với (-R)
	S OR U	(-Q OR S) hợp giải với (Q OR U)
	R	(P OR R) hợp giải với (-P)
	P	(P OR R) hợp giải với (-R)
	S	(P OR S) hợp giải với (-P)
	U	(Q OR U) hợp giải với (-Q OR U)
	1	
	{}	(-U) hợp giải với (U)
	YES	KB entails α vì tồn tại mệnh đề rỗng trong KB

Testcase 4:

Input.txt	**Output.txt**	**Ghi chú**

-U	4	
5	P OR R	(P OR Q) hợp giải với (-Q OR R)
P OR Q	P OR S	(P OR Q) hợp giải với (-Q OR S)
-Q OR R	Q OR U	(P OR Q) hợp giải với (-P OR U)
-Q OR S	-Q OR U	(-Q OR R) hợp giải với (-R OR U)
-P OR U	3	
-R OR U	P OR U	(P OR Q) hợp giải với (-Q OR U)
	R OR U	(-Q OR R) hợp giải với (Q OR U)
	S OR U	(-Q OR S) hợp giải với (Q OR U)
	0	
	NO	KB KHÔNG entail α vì không phát sinh được mệnh đề mới và không tìm thấy mệnh đề rỗng

Testcase 5:

Input.txt	**Output.txt**	**Ghi chú**
-R	9	
6	-Q OR R OR S	(P OR -Q) hợp giải với (-P OR R OR S)
P OR -Q	P	(P OR -Q) hợp giải với (Q)
-P OR R OR S	-Q OR -R	(P OR -Q) hợp giải với (-P OR -R)
Q	P OR R	(P OR -Q) hợp giải với (P OR Q OR R)
-P OR -R	-P OR S	(-P OR R OR S) hợp giải với (-P OR -R)
P OR Q OR R	Q OR R OR S	(-P OR R OR S) hợp giải với (P OR Q OR R)
-Q OR S	S	(Q) hợp giải với (-Q OR S)

	-P	(-P OR -R) hợp giải với (R)
	P OR R OR S	(P OR Q OR R) hợp giải với (-Q OR S)
	8	
	-Q	(P OR -Q) hợp giải với (-P)
	R OR S	(-P OR R OR S) hợp giải với (P)
	-P OR -Q OR S	(-P OR R OR S) hợp giải với (-Q OR -R)
	-R	(Q) hợp giải với (-Q OR -R)
	-P OR Q OR S	(-P OR -R) hợp giải với (Q OR R OR S)
	Q OR R	(P OR Q OR R) hợp giải với (-P)
	{}	(P) hợp giải với (-P)
	P OR -Q OR S	(-Q OR -R) hợp giải với (P OR R OR S)
	YES	KB entails α vì tồn tại mệnh đề rỗng trong KB

6 Ưu nhược điểm và hướng cải thiện

6.1 Ưu điểm

1. **Tính chính xác cao:**

   Hàm `resolve` thực hiện phân giải giữa hai mệnh đề rất chính xác. Nó sử dụng nguyên lý của `PL-Resolution` để loại bỏ các literal đối lập và hợp nhất các mệnh đề, tạo ra các mệnh đề mới mà không vi phạm tính hợp lệ của biểu thức. Kết quả trả về là một mệnh đề có thể tiếp tục được xử lý trong các bước tiếp theo của thuật toán.

2. **Khả năng tái sử dụng:**

   Cách triển khai sử dụng các hàm con như `convert_to_clauses`, `check_complementary`, và `clauses_to_expr`, giúp mã dễ hiểu và tái sử dụng trong các phần khác của hệ thống. Điều này giúp tổ chức mã rõ ràng và dễ bảo trì.

3. **Tiết kiệm bộ nhớ:**

   Việc loại bỏ các literal đối lập và lọc mệnh đề khi chúng đã được xử lý giúp giảm lượng mệnh đề cần lưu trữ trong bộ nhớ. Hàm này chỉ thêm những mệnh đề không bị mâu thuẫn và chưa có trong danh sách hiện tại, giúp tiết kiệm tài nguyên.

4. **Tính linh hoạt cao:**

   Hàm `resolve` có thể xử lý các mệnh đề phức tạp với các toán tử khác nhau (ATOM, NOT, AND, OR) và có thể áp dụng nhiều lần trong quá trình giải quyết vấn đề logic.

6.2 Nhược điểm

1. **Hiệu suất kém với số lượng mệnh đề lớn:**

   Một trong những nhược điểm lớn nhất của cách triển khai này là hiệu suất. Việc duyệt qua tất cả các cặp mệnh đề có thể gây tốn nhiều thời gian khi số lượng mệnh đề tăng lên. Sự kết hợp giữa các mệnh đề có thể tạo ra nhiều mệnh đề mới, khiến việc tính toán trở nên chậm chạp.

2. **Khó khăn trong việc tối ưu hóa:**

   Hàm sử dụng phương pháp kiểm tra các mệnh đề đối lập và loại bỏ chúng, nhưng việc phân giải sẽ sinh ra nhiều mệnh đề phụ thuộc vào số lượng các literal có mặt. Việc giảm thiểu số lượng mệnh đề trong quá trình phân giải là một thử thách lớn, đặc biệt khi mệnh đề không đơn giản và có sự kết hợp phức tạp của các toán tử.

3. **Không tối ưu về bộ nhớ:**

   Mặc dù hàm có khả năng tiết kiệm bộ nhớ khi loại bỏ các mệnh đề đã được xử lý, nhưng cách sử dụng danh sách để lưu trữ tất cả các mệnh đề trong suốt quá trình phân giải vẫn có thể khiến bộ nhớ tiêu tốn rất nhiều khi tập mệnh đề lớn.

4. **Phức tạp trong việc kiểm tra mệnh đề bổ sung:**

   Quá trình kiểm tra các mệnh đề bổ sung (tìm kiếm các mệnh đề đã tồn tại trong `clauses`) có thể làm giảm hiệu suất vì mỗi lần phân giải lại phải so sánh với các mệnh đề cũ. Điều này làm tăng độ phức tạp của thuật toán và có thể dẫn đến việc xử lý không hiệu quả trong các tình huống phức tạp.

5. **Không xử lý được trường hợp mệnh đề vô hạn:**

   Cách triển khai hàm `resolve` trong thuật toán `PL-Resolution` rất hữu ích trong việc giải quyết các vấn đề logic đơn giản đến trung bình. Tuy nhiên, khi đối diện với số lượng mệnh đề lớn hoặc các mệnh đề phức tạp, hiệu suất của hàm có thể bị ảnh hưởng. Các tối ưu hóa thêm về cách tổ chức bộ nhớ và xử lý mệnh đề có thể là cần thiết nếu muốn ứng dụng hàm này cho các bài toán logic quy mô lớn.

6.3 Cải tiến

Ta có thể áp dụng các thuật toán heuristic để thay thế thuật toán `PL-Resolution`, có thể kể đến như thuật toán `dpll` (Có trình bày code trong file `logic.py`)

7 Kết luận

Mã nguồn trình bày một cách hiệu quả việc biểu diễn và kiểm chứng logic mệnh đề. Các lớp và hàm được thiết kế theo hướng tái sử dụng, giúp mở rộng hệ thống dễ dàng. Tuy nhiên, một số tối ưu hóa có thể thực hiện, chẳng hạn như sử dụng cấu trúc dữ liệu phù hợp hơn cho xử lý mệnh đề.