ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Báo cáo Bài thực hành 2

Logic

Cơ sở trí tuệ nhân tạo - CSC14003

Sinh viên thực hiện:

Nguyễn Anh Thái (21120331)

Giảng viên hướng dẫn:

Thầy Bùi Duy Đăng
Thầy Lê Nhựt Nam

Thành phố Hồ Chí Minh, tháng 6 năm 2024

Mục lục

1 Tự đánh giá **2**

2 Lý thuyết nền tảng **2**
2.1 Hợp giải (*Resolution*) 2
2.2 Dạng hội chuẩn (*Conjunctive normal form - CNF*) 3

3 Thuật toán hợp giải (*PL-Resolution*) **3**
3.1 Ý tưởng và mã giả 3
3.2 Ưu và khuyết điểm 4
3.3 Giải pháp cải tiến 5
3.3.1 Thuật toán Davis–Putnam–Logemann–Loveland (DPLL) 5
3.3.2 Các giải pháp khác 6

4 Mô tả mã nguồn **6**

5 Kịch bản kiểm thử **9**

6 Tài liệu tham khảo **13**

1 Tự đánh giá

STT	Tiêu chí đánh giá	Mức độ hoàn thành
1	Đọc dữ liệu đầu vào và lưu trong cấu trúc dữ liệu phù hợp	100%
2	Cài đặt giải thuật hợp giải trên logic mệnh đề	100%
3	Các bước suy diễn phát sinh đủ mệnh đề và kết luận đúng	100%
4	Tuân thủ mô tả định dạng của đề bài	100%
5	Báo cáo test case và đánh giá	100%

Bảng 1: Bảng tiêu chí

2 Lý thuyết nền tảng

2.1 Hợp giải (*Resolution*)

Hợp giải là một phương pháp suy diễn dựa trên **luật hợp giải**, thường được áp dụng trong các hệ thống logic để tìm ra sự mâu thuẫn trong tập hợp các mệnh đề (*clauses*).
Ví dụ:

$$\frac{\begin{array}{c}P \vee \neg R \\ Q \vee R\end{array}}{P \vee Q} \quad \text{hay} \quad (P \vee \neg R) \wedge (Q \vee R) \implies (P \vee Q)$$

Suy luận từ phép hợp giải trên là đúng. Ta chứng minh dễ dàng bằng các phép biến đổi logic (phép hội, hợp, suy ra). Ngoài ra, có thể xét nhanh các trường hợp như sau: Nếu P đúng, suy luận đúng. Nếu P sai, để $P \vee \neg R$ đúng (vì trong cơ sở tri thức) thì $\neg R$ đúng hay R sai (còn gọi là **luật hợp giải đơn**), tương tự để $Q \vee R$ đúng thì Q đúng (do R đã sai), suy luận vẫn đúng.

Tổng quát hơn:

$$\frac{\begin{array}{c}l_1 \vee \cdots l_k \\ m_1 \vee \cdots m_n\end{array}}{l_1 \vee \cdots l_{i-1} \vee l_{i+1} \vee l_k \vee m_1 \vee \cdots m_{j-1} \vee m_{j+1} \vee \cdots m_n} \quad \text{với } l_i \text{ và } m_j \text{ là 2 literal bù nhau}$$

2 mệnh đề hợp giải sẽ tạo ra 1 mệnh đề mới không chứa cặp literal bù nhau.

Một số lưu ý: mệnh đề kết quả không chứa literal trùng nhau, chỉ áp dụng hợp giải cho một cặp literal bù nhau duy nhất tại mỗi lần.

Sử dụng hợp giải mệnh đề (không cần sử dụng các luật khác) có thể xây dựng một chương trình chứng minh lý thuyết đúng và đủ cho tất cả logic mệnh đề. Tuy nhiên, phương pháp này yêu cầu cơ sở tri thức và các câu truy vấn phải ở dạng hội chuẩn (CNF).

2.2 Dạng hội chuẩn (*Conjunctive normal form - CNF*)

Dạng hội chuẩn là dạng hội (liên kết và - `AND`) của các mệnh đề (*clause*). Mỗi clause là một dạng tuyển (liên kết hợp - `OR`) của các literal.

Ví dụ:

$$(A \vee B \vee \neg C) \wedge (B \vee D) \wedge (\neg A)$$

Cách biến đổi một biểu thức mệnh đề (sentence) bất kỳ theo CNF:

1. **Loại bỏ $\Leftrightarrow$**
   Thay $\alpha \Leftrightarrow \beta$ bằng $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
2. **Loại bỏ $\Rightarrow$**
   Thay $\alpha \Rightarrow \beta$ bằng $\neg\alpha \vee \beta$.
3. **CNF đòi hỏi dấu $\neg$ chỉ được xuất hiện trước các literal, do đó cần phân phối dấu phủ định $\neg$:**
   - $\neg(\neg\alpha) \equiv \alpha$
   - $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ (luật De Morgan)
   - $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ (luật De Morgan)
4. **Áp dụng luật phân phối $\vee$ vào $\wedge$ cho tất cả các trường hợp có thể.**

3 Thuật toán hợp giải (*PL-Resolution*)

3.1 Ý tưởng và mã giả

Thuật toán hợp giải hoạt động dựa trên nguyên tắc chứng minh bằng phản chứng (*proof by contradiction*). Tức là, để chứng minh $\texttt{KB} \models \alpha$, ta chứng minh $\texttt{KB} \vee \neg\alpha$ là không thể thỏa mãn được (*unsatisfiable*).

Các bước chính:

- Biểu thức $\texttt{KB} \vee \neg\alpha$ được chuyển đổi thành dạng CNF.
- Đối với mỗi cặp mệnh đề trong tập hợp các mệnh đề CNF, quy tắc hợp giải được áp dụng. Nếu một cặp mệnh đề có các literal bù nhau (*complementary literals*), chúng sẽ được hợp giải để tạo ra một mệnh đề mới. Mệnh đề mới này được thêm vào tập hợp nếu nó chưa có trong đó.
- Quá trình hợp giải tiếp tục cho đến khi không còn mệnh đề mới nào có thể được thêm vào, hoặc cho đến khi hai mệnh đề hợp giải được tạo ra mệnh đề rỗng (*empty clause*).
- Nếu một mệnh đề rỗng được tạo ra, $\texttt{KB} \vee \neg\alpha$ là không thỏa mãn được, và do đó $\texttt{KB} \not\models \alpha$. Ngược lại, nếu không có mệnh đề rỗng được tạo ra, $\texttt{KB} \models \alpha$.

Algorithm 1 PL-RESOLUTION

```
function PL-RESOLUTION(KB, α)
    Inputs: KB, the knowledge base, a sentence in propositional logic
            α, the query, a sentence in propositional logic

    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← {}
    while true do
        for each pair of clauses C_i, C_j in clauses do
            resolvents ← PL-RESOLVE(C_i, C_j)
            if resolvents contains the empty clause then
                return true
            new ← new ∪ resolvents
        if new ⊆ clauses then
            return false
        clauses ← clauses ∪ new
```

3.2 Ưu và khuyết điểm

Ưu điểm:

- Độ tin cậy cao (*soundness*): Các mệnh đề kết quả luôn là hệ quả được suy ra từ cơ sở tri thức và truy vấn ban đầu, do đó thuật toán đảm bảo tính đúng đắn của suy luận.
- Độ hoàn thiện cao (*completeness*): Nếu tập hợp các mệnh đề không thỏa mãn, thì kết quả khi kết thúc quá trình hợp giải sẽ chứa mệnh đề trống, đảm bảo tính hoàn thiện của suy luận.
- Dễ hiểu, dễ cài đặt, là nền tảng cho nhiều phương pháp hiện đại khác.

Khuyết điểm:

- Thuật toán đòi hỏi toàn bộ mệnh đề đều ở dạng CNF.
- Thuật toán vét cạn tất cả trường hợp hợp giải, tạo ra một số lượng lớn các mệnh đề tạm thời trong quá trình thực thi, gây ra khó khăn về không gian lưu trữ và thời gian thực thi. Độ phức tạp có thể lên đến $O(mn^2)$ với n là số clause trong $\texttt{KB} + \alpha$ và m là số lần mà ta phải chạy lại thuật toán trên KB mới.
- Các cặp mệnh đề khi hợp giải sẽ tạo ra nhiều mệnh đề kết quả, nhiều kết quả là không cần thiết cho quá trình suy diễn, gây tốn không gian lưu trữ.
- Quá trình hợp giải thường tạo ra các mệnh đề kết quả trùng lặp trong cơ sở tri thức, gây tốn thời gian tính toán không cần thiết.
- Khó khăn hoặc thậm chí không thể đưa ra kết quả khi số lượng mệnh đề ban đầu lớn và phức tạp.

3.3 Giải pháp cải tiến

3.3.1 Thuật toán Davis–Putnam–Logemann–Loveland (DPLL)

Thuật toán DPLL là một phiên bản cải tiến của thuật toán Davis-Putnam, được thiết kế để giải quyết các bài toán về logic mệnh đề (propositional logic). DPLL nhận đầu vào là một câu ở dạng CNF và tiến hành tìm kiếm đệ quy theo chiều sâu các mô hình khả dĩ. So với các thuật toán như TT-ENTAILS? và PL-Resolution, nó thể hiện 3 cải tiến chính:

- **Dừng sớm (Early termination)**: Thuật toán có thể phát hiện một câu phải đúng hoặc sai ngay cả khi mô hình chưa hoàn tất. Một mệnh đề là đúng nếu bất kỳ một literal nào trong nó đúng, và một mệnh đề là sai nếu tất cả các literal trong nó đều sai. Điều này giúp tránh phải kiểm tra toàn bộ cây tìm kiếm.
- **Heuristic ký hiệu thuần khiết (Pure symbol heuristic)**: Một ký hiệu thuần khiết là ký hiệu luôn xuất hiện với cùng một dấu trong tất cả các mệnh đề. Việc xác định độ thuần khiết của một ký hiệu có thể bỏ qua các mệnh đề đã biết là TRUE trong mô hình hiện tại.
- **Heuristic mệnh đề đơn vị (Unit clause heuristic)**: Một mệnh đề đơn vị là mệnh đề chỉ có một literal, hoặc một mệnh đề mà tất cả các literal ngoại trừ nó đã được gán giá trị FALSE bởi mô hình.

So với PL-RESOLUTION, DPLL có khả năng dừng sớm mà không cần thông qua các mô hình khả dĩ, sử dụng các heristic hiệu quả giúp giảm số lượng phân nhánh cần kiểm tra và tăng tốc độ tìm kiếm.

Algorithm 2 DPLL

function DPLL-SATISFIABLE?(s)
  Inputs: s, a sentence in propositional logic
  $clauses \leftarrow$ the set of clauses in the CNF representation of s
  $symbols \leftarrow$ a list of the proposition symbols in s
  return DPLL($clauses, symbols, \{\}$)

function DPLL($clauses, symbols, model$)
  if every clause in $clauses$ is true in $model$ **then**
    return true
  if some clause in $clauses$ is false in $model$ **then**
    return false
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if P is non-null **then**
    return DPLL($clauses, symbols - P, model \cup \{P = value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if P is non-null **then**
    return DPLL($clauses, symbols - P, model \cup \{P = value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P = \text{true}\}$) or
      DPLL($clauses, rest, model \cup \{P = \text{false}\}$)

3.3.2 Các giải pháp khác

- Cải tiến chiến lược chọn cặp mệnh đề để hợp giải: áp dụng các heuristic để họn các cặp mệnh đề có khả năng dẫn đến hợp giải hiệu quả hơn, Ưu tiên các mệnh đề đơn vị và các mệnh đề có ít biến để hợp giải trước...
- Học và lưu trữ các mệnh đề trung gian hữu ích: Conflict-driven Clause Learning (CDCL), Non-Chronological Backtracking...
- Kết hợp với các kỹ thuật tìm kiếm khác: Integrating DPLL (Kết hợp PL-Resolution với DPLL để tận dụng ưu điểm của cả hai), Local Search Integration (Kết hợp với các thuật toán tìm kiếm cục bộ như WALKSAT để tìm kiếm các mô hình thỏa mãn nhanh chóng, sau đó dùng PL-Resolution để xác nhận tính chính xác và hoàn thiện việc chứng minh)...

4 Mô tả mã nguồn

Mã nguồn được tổ chức thành các lớp: `Literal`, `Clause` và `KnowledgeBase`. Mỗi lớp thực hiện chức năng như tên. Trong đó, `Literal` và `Clause` được nạp chồng các toán tử (==, <, hash), mục đích để sử dụng `set` của thư viện chuẩn của Python.

- `Literal`: một literal bao gồm một symbol và dấu của nó. Có các phương thức sau:
  - `negate()`: phủ định literal (đổi dấu).
  - `is_complement(other)`: kiểm tra 2 literal có bù nhau hay không.
  - `parse(str)`: static method, nhận 1 string và trả về 1 `Literal` object.
- `Clause`: một clause là một tuyển (hợp) của các literal, ta lưu `list` các literal. Có các phương thức sau:
  - `add_literal(literal: Literal)`: thêm literal vào clause.
  - `is_empty()`: kiểm tra clause rỗng.
  - `refactor()`: loại bỏ trùng và sắp xếp lại các literal trong clause, dùng `sorted(set())` từ thư viện chuẩn.
  - `is_always_true()`: kiểm tra một clause có luôn đúng hay không. Ví dụ: '-A OR B OR -B'.
  - `merge(other)`: hợp 2 clause, đồng thời bỏ trùng. Ví dụ: 'A OR B', '-A OR C' → 'B OR C'.
  - `clone_without(literal: Literal)`: sao chép một clause ngoại trừ một literal được xác định. Ví dụ: 'A OR B OR C' without 'A' → 'B OR C'.
  - `parse(str)`: static method, nhận 1 string và trả về 1 `Clause` object.
  - `pl_resolve(other)`: hàm xử lý hợp giải 2 clause, trả về một `set` là tất cả các trường hợp có thể khi thực hiện hợp giải với từng cặp literal trong 2 clause đồng thời kèm một biến bool xác nhận set chứa clause rỗng hay không. Ví dụ:

```
'A OR B OR C', '-A OR D OR -C':
+ 'A' is complement of '-A' => resolvent = 'B OR C OR -C OR D' = TRUE => bỏ
+ 'C' is complement of '-C' => resolvent = 'A OR B OR -B OR D' = TRUE => bỏ
+ Result: resolvents = {}, has_empty_clause = FALSE
```

- `KnowledgeBase`: tập hợp các clause được cho trước, ta lưu `list` các `Clause` object. Có các phương thưc ssa
  - `add_clause(clause: Clause)`: thêm clause vào KB.
  - `parse(str)`: static method, nhận 1 `list` string đại diện cho các clause và trả về 1 `KB` object.
  - `pl_resolution(alpha: str)`: hàm xử lý thuật toán hợp giải, được cài đặt như mã giả 1, trả về một bool xác nhận có thể suy luận alpha từ KB hay không và một `list` các clause mới.

Hàm xử lý chính như sau:

```
1
2     def pl_resolution(self, alpha: str) -> Tuple[bool, List[List[str]]]:
3         """
4         - PL resolution algorithm
5         - Return a boolean value to check if alpha is entailed by KB and a list
      of new clauses
6         - alpha: a string represents a clause. We need to negate it before
      adding to KB, i.e. negate each literal and convert to CNF clause
7             - Eg: 'A OR B' => '-A AND -B'
8         """
9         # 1. Create 'clauses' is a set of clauses of KB and -alpha
10        clauses = self.clauses.copy()
11        alpha_literals = alpha.split(" OR ")
12        for literal_str in alpha_literals:
13            literal_str = literal_str.strip()
14            literal = Literal.parse(literal_str)
15            literal.negate()
16            clause = Clause()
17            clause.add_literal(literal)
18            if clause not in clauses:  # Remove duplicate clauses
19                clauses.append(clause)
20
21        # 2. Loop
22        clauses = set(clauses)
23        output = []  # output is a list of new clauses
24        is_unsatisfiable = False
25
26        while True:
27            # 2.1. Loop through all pairs of clauses and resolve them
28            new_clauses = set()
29            for clause1 in clauses:
30                for clause2 in clauses:
31                    resolvents, has_empty_clause = clause1.pl_resolve(clause2)
32                    new_clauses.update(resolvents)
33                    if has_empty_clause:
```

```
                        is_unsatisfiable = True
                        break

            # Create a set of new clauses that are not in the 'clauses' set to
    the 'output' list
            output.append(new_clauses.difference(clauses))

            # 2.2. Check if there is an empty clause
            if is_unsatisfiable:
                return True, output

            # 2.3. Check if new_clauses is a subset of clauses
            if new_clauses.issubset(clauses):
                return False, output

            # 2.4. Update clauses
            clauses.update(new_clauses)
```

Thuật toán được cài đặt như mã giả. Gồm 2 giai đoạn chính:

1. Tạo tập hợp các clause: Chuyển đổi alpha sang CNF, ta tách từng literal trong string và thực hiện parse sang `Clause`, mỗi literal lúc này là một clause và được thêm vào danh sách các clause của KB.
2. Lặp: thực hiện hợp giải từng cặp clause trong tập hợp, tạo các clause mới và xét các điều kiện. Trong đó, việc nạp các clause mới vào mảng `output` được thực hiện sau khi một vòng lặp kết thúc, bằng cách đưa vào những clause sinh ra mà không có trong tập hợp ban đầu. Việc kiểm tra tính thỏa mãn được thay vì được thực hiện trong vòng lặp, ta sử dụng biến bool và kiểm tra ngay sau khi thêm `output`, mục đích để `output` cập nhật. Việc kiểm tra có tồn tại clause mới (bằng cách xem clause mới có là tập con tập ban đầu) và update tập hợp được thực hiện cuối cùng.

Do dùng `set` để lưu trữ nên thứ tự các mệnh đề in ra ở mỗi lần lặp có thể không giống nhau.

5 Kịch bản kiểm thử

Input	**Output**	**Chú thích**
A	6	
6	C	(-B OR C) hợp giải với (B)
B	-B OR D	(-B OR C) hợp giải với (-C OR D)
-B OR C	-C OR E	(-C OR D) hợp giải với (-D OR E)
-C OR D	-D OR F	(-D OR E) hợp giải với (-E OR F)
-D OR E	A OR -E	(-E OR F) hợp giải với (A OR -F)
-E OR F	-F	(A OR -F) hợp giải với (negative of A)
A OR -F	9	
	A OR -C	(A OR -E) hợp giải với (-C OR E)
	D	(B) hợp giải với (-B OR D)
	-E	(negative of A) hợp giải với (A OR -E)
	E	(C) hợp giải với (-C OR E)
	-B OR E	(-B OR C) hợp giải với (-C OR E)
	-D	(-D OR F) hợp giải với (-F)
	-C OR F	(-C OR D) hợp giải với (-D OR F)
	-B OR F	(-B OR D) hợp giải với (-D OR F)
	A OR -D	(A OR -F) hợp giải với (-D OR F)
	6	
	A OR -B	(A OR -E) hợp giải với (-B OR E)
	A	(C) hợp giải với (A OR -C)
	F	(-E OR F) hợp giải với (E)
	-B	(-B OR D) hợp giải với (B)
	-C	(-C OR D) hợp giải với (-D)
	{}	(D) hợp giải với (-D)
	YES	KB entails α vì xuất hiện {} trong KB

Bảng 2: Test case 1

Input	**Output**	**Chú thích**
B OR C	4	
4	A OR -D	(A OR -C) hợp giải với (C OR -D)
A OR -C	A OR C	(A OR D) hợp giải với (C OR -D)
A OR D	A OR B	(A OR D) hợp giải với (B OR -D)
B OR -D	-D	(B OR -D) hợp giải với (negative of B)
C OR -D	1	
	A	(negative of B) hợp giải với (A OR B)
	0	
	NO	Không {} và không phát sinh mệnh đề mới

Bảng 3: Test case 2

Input	**Output**	**Chú thích**
C OR -E	7	
5	D OR F	(C OR D OR F) hợp giải với (negative of C)
B OR -E	C OR F	(-D) hợp giải với (C OR D OR F)
-D	B	(B OR -E) hợp giải với (negative of E)
A OR D	A	(-D) hợp giải với (A OR D)
-A OR -B OR C	-B OR C OR D	(A OR D) hợp giải với (-A OR -B OR C)
C OR D OR F	-A OR C OR -E	(B OR -E) hợp giải với (-A OR -B OR C)
	-A OR -B	(-A OR -B OR C) hợp giải với (negative of C)
	10	
	F	(-D) hợp giải với (D OR F)
	-B	A hợp giải với (-A OR -B)
	-A OR -E	(negative of C) hợp giải với (-A OR C OR -E)
	C OR D	(B) hợp giải với (-B OR C OR D)
	-A OR C	(-A OR -B OR C) hợp giải với (B)
	-B OR D	(negative of C) hợp giải với (-B OR C OR D)
	C OR -E	(A) hợp giải với (-A OR C OR -E)
	-A	(B) hợp giải với (-A OR -B)
	C OR D OR -E	(B OR -E) hợp giải với (-B OR C OR D)
	-B OR C	(-A OR -B OR C) hợp giải với (A)
	5	
	D	(negative of C) hợp giải với (C OR D)
	-E	A hợp giải với (-A OR -E)
	C	A hợp giải với (-A OR C)
	D OR -E	(A OR D) hợp giải với (-A OR -E)
	{}	(A) hợp giải với (-A)
	YES	Xuất hiện {}

Bảng 4: Test case 3

Input	**Output**	**Chú thích**
-D OR F	6	
5	B OR F	(A OR B) hợp giải với (-A OR F)
A OR B	A OR C	(A OR B) hợp giải với (-B OR C)
B OR D OR F	B OR D	(B OR D OR F) hợp giải với (negative of F)
-B OR C	C OR D OR F	(B OR D OR F) hợp giải với (-B OR C)
-C OR E	-B OR E	(-B OR C) hợp giải với (-C OR E)
-A OR F	-A	(-A OR F) hợp giải với (negative of F)
	8	
	D OR E	(B OR D) hợp giải với (-B OR E)
	A OR E	(A OR B) hợp giải với (-B OR E)
	E OR F	(B OR F) hợp giải với (-B OR E)
	B	(A OR B) hợp giải với (-A)
	C OR D	(-B OR C) hợp giải với (B OR D)
	C OR F	(-B OR C) hợp giải với (B OR F)
	D OR E OR F	(-C OR E) hợp giải với (B OR D OR F)
	C	(A OR C) hợp giải với (-A)
	1	
	E	(negative of F) hợp giải với (E OR F)
	0	
	NO	Không {} và không phát sinh mệnh đề mới

Bảng 5: Test case 4

Input	**Output**	**Chú thích**
A OR B OR C	5	
3	C	(B OR C) hợp giải với (negative of B)
A OR B OR -D	A OR -D	(A OR -D OR B) hợp giải với (negative of B)
-A OR C	B OR -D	(A OR -D OR B) hợp giải với (negative of A)
B OR C	B	(B OR C) hợp giải với (negative of C)
	B OR C OR -D	(A OR -D OR B) hợp giải với (-A OR C)
	3	
	C OR -D	(-A OR C) hợp giải với (A OR -D)
	-D	(negative of B) hợp giải với (B OR -D)
	{}	(negative of B) hợp giải với (B)
	YES	Xuất hiện {}

Bảng 6: Test case 5

Nhận xét:

- Giải thuật chạy hiệu quả trên dữ liệu đã chuẩn hóa theo quy ước.
- Số lượng mệnh đề phát sinh tương đối lớn, tăng nhanh khi số lượng các literal khác nhau trong các mệnh đề ban đầu của KB nhiều.

6 Tài liệu tham khảo

[1] Slide bài giảng môn học *Cơ sở trí tuệ nhân tạo*.

[2] Stuart Russel, Peter Norgiv. *Artificial Intelligence: A Modern Approach (4th ed)*.

[3] *logic.ipynb, aima-python repo*
https://github.com/aimacode/aima-python/blob/master/logic.ipynb