1 Forward chaining

In this section, we will be proving a statement using forward chaining.

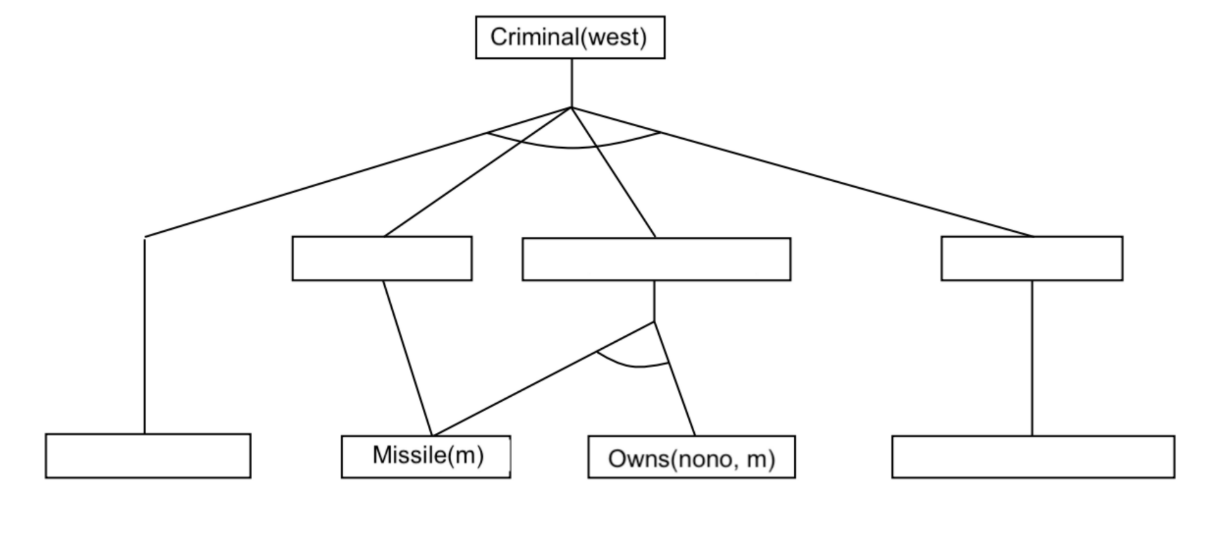
There is currently a war going on and the United States is desperate to round up all the criminals. We want to determine whether Colonel West is a criminal. Let’s start with what we know.

We know that it is a crime for an American to sell weapons to hostile nations. The country Nono is an enemy of America. Furthermore, we know that Nono has some missiles, all of which were sold to it by Colonel West, who is American.

(a) Represent your knowledge base using first order logic. You can use the following function predicates: American(x), Criminal(x), Hostile(x), Missile(x), Weapon(x), Enemy(x,y), Owns(x,y), Sells(x,y,z).

1. American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
2. Missile(x) ⇒ Weapon(x)
3. Missile(m)
4. Owns(nono, m)
5. Missile(x) ∧ Owns(nono, x) ⇒ Sells(west, x, nono)
6. Enemy(x, america) ⇒ Hostile(x)
7. Enemy(nono, america)
8. American(west)

(b) Fill in the blanks below using your knowledge base to prove that Colonel West is a criminal.



A diagram of a company

AI-generated content may be incorrect.

1

2 Planning Tower of Hanoi

In the Tower of Hanoi problem, you are given *n* disks, each of a distinct size, and 3 rods, *A, B* and *C*. The disks start off stacked on top of each other on rod *A*, stacked from largest being the lowest to smallest being the highest in a “tower”, and the goal is to move that tower to the rod *C*. You can only move a disk to an empty rod or on top of a larger disks, and disks may only have one other disk on its surface (they must be stacked linearly).



1. Assume we have 3 disks. Formulate the problem as a graph-planning problem, specifying instances, operators, and start/goal states.

**1. Instance (thành phần cố định)**

Đĩa: D = {1,2,3} với 1 là nhỏ nhất, 3 là lớn nhất.

Cọc (rods): P = {A, B, C}.

**2. State representation (cách biểu diễn trạng thái)**

Một trạng thái là ánh xạ vị trí pos cho mỗi đĩa:

pos : D → P (ví dụ pos(1)=A nghĩa đĩa 1 ở cọc A).

Từ pos ta có các nguyên tử (propositions) có ích:

On(d, p) — đĩa d đang ở cọc p (tương đương pos(d)=p).

Clear(d) — trên đĩa d không có đĩa nào (tức là d là đĩa trên cùng tại cọc của nó).

TopEmpty(p) (tùy chọn) — cọc p rỗng (không có đĩa).

Ta thường dùng On + định nghĩa Clear suy ra từ On:

Clear(d) ⇔ (∀k < d) pos(k) ≠ pos(d) (không có đĩa nhỏ hơn (nhỏ hơn = chỉ số nhỏ hơn) ở cùng cọc).

**3. Operators**

**Action chung**: Move(i, X, Y) — di chuyển đĩa i từ cọc X sang cọc Y.

**Preconditions:**

On(i, X) (đĩa i ở X)

Clear(i) (i là đĩa trên cùng ở X) — không có đĩa nhỏ hơn ở X

Top(Y) > i hoặc Y rỗng (tức là nếu có đĩa đang ở trên cùng của Y thì nó phải lớn hơn i). Biểu diễn logic: (∄k < ∞ s.t. On(k,Y)) ∨ (let j = smallest k with On(k,Y) then j > i) — ngắn gọn: i nhỏ hơn đĩa đang top ở Y (hoặc Y rỗng).

**Effects:**

On(i, X) bị loại bỏ (negative effect).

On(i, Y) được thêm (positive effect).

Clear(i) vẫn đúng (sau khi di chuyển, i là top ở Y).

Clear(j) cho đĩa j từng top ở Y bị xóa nếu j tồn tại.

Clear(k) cho đĩa nhỏ hơn trên X có thể được thêm nếu X trở nên rỗng hoặc top thay đổi — (thực tế các effect được xác định bằng cập nhật On cho mỗi đĩa).

Cụ thể với 3 đĩa, có các action ground (đặc biệt hóa) như Move(1,A,B), Move(1,A,C), Move(2,A,B), ... tổng cộng tối đa 3 \* 2 \* n action ground (cho mỗi đĩa i: từ 2 cọc khác nhau).

**4. Start state (khởi đầu)**

Tất cả đĩa trên A, lớn nhất dưới cùng:

On(1,A), On(2,A), On(3,A).

Từ đó suy ra Clear(1) = True (đĩa 1 top), Clear(2) = False, Clear(3) = False.

5. Goal state (mục tiêu)

Tất cả đĩa trên C theo thứ tự:

On(1,C), On(2,C), On(3,C).

1. Draw the planning graph for the first 3 moves. You may use pictures instead of propositions.

* A diagram of a flowchart

  AI-generated content may be incorrect.

1. Generalize the problem formulation for *n* disks.

**1. State**

pos : {1..n} → {A,B,C}, 1 là nhỏ nhất, n là lớn nhất.

Propositions: On(i, p) cho mọi i,p; Clear(i) (i top trên cọc của nó) suy ra từ pos.

**2. Operators**

Action schema: Move(i, X, Y) với i ∈ {1..n}, X≠Y ∈ {A,B,C}.

Preconditions:

On(i, X)

i là đĩa nhỏ nhất ở X (không có k < i với On(k,X)) ⇒ viết: Top(X) = i.

Y trống OR Top(Y) > i (đĩa top của Y có chỉ số lớn hơn i).

**Effects:**

On(i,X) false, On(i,Y) true,

update Clear cho đĩa bị lộ ra trên X (nếu có) và cho i (i là top trên Y).

**3. Kích thước không gian trạng thái**

Có 3^n trạng thái khả dĩ (mỗi đĩa có 3 lựa chọn cọc). Tuy nhiên chỉ một phần là hợp lệ theo thứ tự (nhưng thực tế với định nghĩa pos bất kỳ vẫn hợp lệ nếu ta cho phép trạng thái “không đúng thứ tự”? Trong Tower of Hanoi mọi trạng thái hợp lệ là bất kỳ ánh xạ pos; ràng buộc thứ tự được đảm bảo bởi phép đi nên không cần giới hạn thêm).

Chiều dài plan tối thiểu để chuyển cả tower từ A→C là 2^n - 1 (định lý cổ điển).

**4. Thuật toán giải (cổ điển, đệ quy)**

Đệ quy tối ưu cho n đĩa từ A→C dùng B là auxiliary:

Hanoi(n, A, C, B):

if n == 1:

move disk 1 from A to C

else:

Hanoi(n-1, A, B, C) # di chuyển n-1 nhỏ nhất từ A sang B

move disk n from A to C

Hanoi(n-1, B, C, A) # di chuyển n-1 từ B sang C

Số bước T(n) = 2\*T(n-1) + 1 → T(n) = 2^n - 1.

2

3 Discussion-Based Questions

Let us consider forward-chaining in both a first-order logic and propositional logic setting. Find someone sitting near you to talk through the following questions with, and take some time to look through the following pseudocode snippets from the textbook.

A close-up of a paper

AI-generated content may be incorrect.

Figure 1: Forward-chaining algorithm for propositional logic, from p. 258 of the course textbook. A page of a math exam

AI-generated content may be incorrect.

Figure 2: Forward-chaining algorithm for first-order logic, from p. 332 of the course textbook.

* 1. First things first, what are some similarities and distinctions between propositional logic and first order logic?

1 Giống nhau

Cả Propositional Logic (PL) và First-Order Logic (FOL) đều:

Dựa trên logic mệnh đề hợp lý

Đều dùng các phép:

• ∧ (AND)

• ∨ (OR)

• ¬ (NOT)

• → (IMPLIES)

Cho phép suy diễn (inference) và kiểm tra entailment

Forward-chaining, backward-chaining, resolution đều có thể áp dụng (với điều chỉnh tương ứng).

Kiến thức được biểu diễn qua các mệnh đề/klaus

Cả hai đều chứa:

• Tập các luật (rules)

• Tập các sự kiện (facts)

• Một câu truy vấn cần được chứng minh (query).

Forward-chaining trong cả hai đều làm:

• Lặp qua tất cả các luật

• Tìm các luật có thể áp dụng

• Thêm suy diễn mới vào KB

• Kết thúc khi không tạo được luật mới hoặc khi suy diễn được query

**2. Khác nhau**

Đây là phần quan trọng nhất — và liên quan trực tiếp đến hai thuật toán trong hình.

Propositional Logic First-Order Logic

Không có biến, chỉ có mệnh đề nguyên tử như P, Q Có biến, hàm, đối tượng, quantifiers như ∀x, ∃y

Mỗi mệnh đề là true/false Một predicate có thể áp dụng cho nhiều đối tượng

Mỗi fact là riêng biệt Một luật có thể mô tả vô hạn sự kiện bằng biến

(ii) Luật suy diễn

Propositional rule:

P ∧ Q → R

Mệnh đề cụ thể, không thay đổi.

FOL rule:

P(x) ∧ Q(x, y) → R(y)

Có biến → phải thay thế (substitution).

(iii) Thuật toán forward-chaining

Trong Propositional Logic

• Chỉ cần kiểm tra các tiền đề có tất cả đều True hay chưa.

• Không có bước:

o chuẩn hoá biến (standardize variables)

o thay thế (substitution)

o hợp nhất (unification)

➜ Thuật toán rất nhanh, vì chỉ đếm xem đủ premise hay chưa.

3

* 1. Compare and contrast these two algorithms. At a high level, what similarities can you identify, and where are there differences?

Similarities:

• Cùng là suy diễn tiến (forward chaining).

• Cùng mở rộng KB bằng cách áp dụng luật khi tiền đề thỏa mãn.

• Cùng lặp cho đến khi tìm ra query hoặc không rút ra được câu mới.

Differences:

• PL-FC hoạt động trên mệnh đề không có biến; FOL-FC có biến và cần unification + substitution.

• PL-FC dùng bộ đếm (count) để kiểm tra luật; FOL-FC phải thử mọi phép thay thế khả dĩ.

• PL-FC luôn kết thúc (finite); FOL-FC có thể sinh vô hạn facts và không đảm bảo kết thúc.

• PL-FC rất hiệu quả; FOL-FC phức tạp và kém hiệu quả hơn nhiều.

* 1. Now, consider the forward-chaining algorithm presented in Figure 2. It is designed to be conceptually straightforward, but is rather inefficient. What inefficiencies can you identify in this code?

|  |  |
| --- | --- |
| Vấn đề | Lý do kém hiệu quả |
| Duyệt lại toàn bộ KB mỗi iteration | Lặp lại không cần thiết |
| Chuẩn hóa biến mỗi lần | Tốn tài nguyên, lặp lại nhiều |
| SUBST và UNIFY lặp quá nhiều | Gây complexity rất cao |
| Không dùng agenda | Không tối ưu rules cần xét |
| Không ghi nhớ trạng thái đã xử lý | Mỗi vòng lặp giống như chạy lại từ đầu |
| KB phình to và gây nhiều kiểm tra trùng lặp | Complexity tăng nhanh theo thời gian |

4