

## BÀI TẬP LÝ THUYẾT – NHẬP MÔN TRÍ TUỆ NHÂN TẠO – LẦN 2

Họ và tên: Nguyễn Thanh Kiên.

MSSV: 22110092

### Nhắc lại lý thuyết:

Khái niệm Admissible Heuristics và Consistent Heuristics là hai khái niệm quan trọng trong tính tối ưu của thuật toán A\*. Trong đó:

**Admissible Heuristics** – hàm đánh giá có thể chấp nhận được, là một hàm không bao giờ đánh giá cao chi phí để đạt đến Node Goal, tức là chi phí mà nó ước tính để đạt đến Node Goal không cao hơn chi phí thấp nhất có thể từ Node hiện tại trong đường đi. Vì  $g(n)$  là chi phí thực tế để đạt đến Node  $n$  dọc theo đường đi hiện tại, và

$$f(n) = g(n) + h(n) \text{ và } h(n) \leq h^*(n)$$

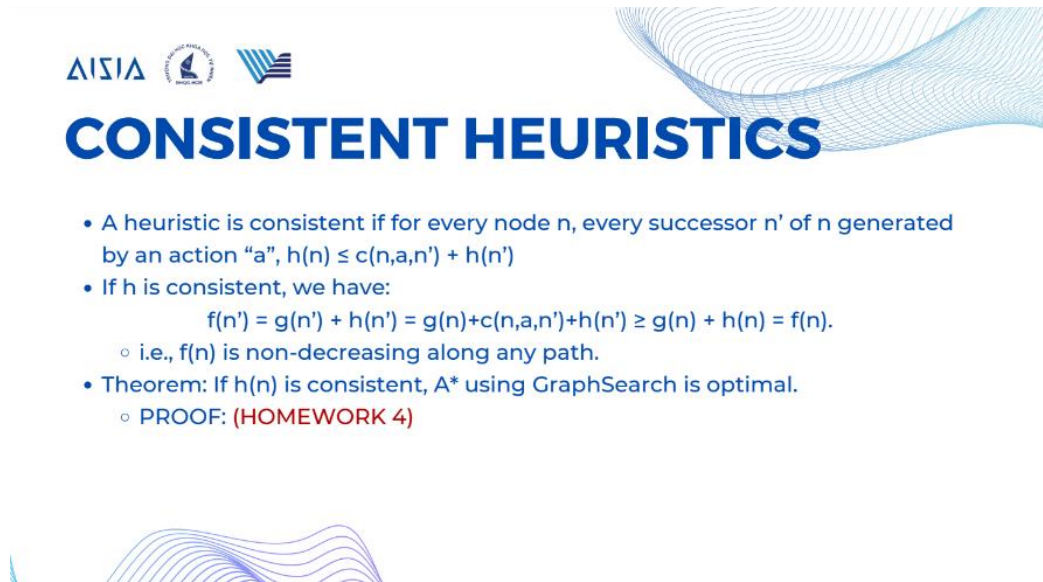
nên ngay lập tức kết quả là  $f(n)$  không bao giờ đánh giá cao chi phí thực sự của một giải pháp dọc theo đường đi hiện tại qua  $n$ . Điều kiện để *Admissible Heuristics* có tính tối ưu là nó chỉ được sử dụng trong Tree Graph (điều này có nghĩa là AH sử dụng trong Graph Search cũng được)

**Consistent Heuristics** – hàm đánh giá nhất quán được sử dụng để nghiên cứu các bài toán tìm đường đi trong trí tuệ nhân tạo, nó được gọi là nhất quán hoặc đơn điệu nếu khoảng cách ước tính của nó luôn nhỏ hơn hoặc bằng khoảng cách ước tính từ bất kỳ đỉnh liền kề nào đến Node Goal, cộng với chi phí để đạt đến Node đó. Một hàm gọi là nhất quán nếu, đối với mỗi Node  $n$  và mỗi Node kế cận của  $n$  được tạo bởi bất kỳ hành động  $a$  nào, chi phí ước tính để đạt được mục tiêu từ  $n$  không lớn hơn chi phí bước để đến  $n$  cộng với chi phí ước tính để đạt được mục tiêu từ  $n$ :

$$h(n) \leq c(n, a, n') + h(n') \text{ và } f(n') \geq f(n)$$

Một hàm nhất quán cũng được coi là chấp nhận được, nghĩa là nó không bao giờ đánh giá cao chi phí để đạt đến mục tiêu (tuy nhiên điều ngược lại không đúng hoàn toàn, hay một hàm chấp nhận được không phải lúc nào cũng nhất quán, chứng minh bằng cách quy nạp toán học với các đường đi không có giá trị âm). Điều kiện để *Consistent Heuristics* có tính tối ưu là nó chỉ được sử dụng trong Graph Search (điều này có nghĩa CH sử dụng trong Tree Search cũng được).

## Bài 1.



**CONSISTENT HEURISTICS**

- A heuristic is consistent if for every node  $n$ , every successor  $n'$  of  $n$  generated by an action " $a$ ",  $h(n) \leq c(n,a,n') + h(n')$
- If  $h$  is consistent, we have:
  - $f(n') = g(n') + h(n') = g(n) + c(n,a,n') + h(n') \geq g(n) + h(n) = f(n)$ .
    - i.e.,  $f(n)$  is non-decreasing along any path.
- Theorem: If  $h(n)$  is consistent, A\* using GraphSearch is optimal.
  - PROOF: (HOMEWORK 4)

Chứng minh nếu hàm  $h(n)$  là nhất quán thì thuật toán A\* sử dụng trong Graph Search có tính tối ưu (optimal):

Giả sử đồ thị tồn tại một đường đi  $p$  với Node bắt đầu là  $\delta_1$  và Node Goal là  $\delta_{i+1}$ .

Vì  $h(n)$  là hàm nhất quán nên nó cũng là hàm chấp nhận được, nên hàm tổng chi phí:

$$f(\delta_{i+1}) = g(\delta_{i+1}) + h(\delta_{i+1})$$

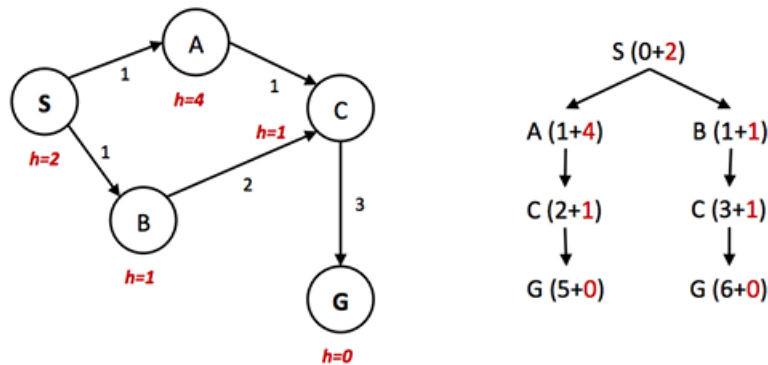
Sử dụng tính chất của hàm nhất quán, với  $g(\delta_{i+1}) = g(\delta_i) + c(\delta_i, a, \delta_{i+1})$

Và  $h(\delta_i) \leq c(\delta_i, a, \delta_{i+1}) + h(\delta_{i+1})$

Ta được:  $f(\delta_{i+1}) = g(\delta_i) + c(\delta_i, a, \delta_{i+1}) + h(\delta_{i+1}) \geq g(\delta_i) + h(\delta_i) = f(\delta_i)$

Hay, với mọi cặp cha con (parent – child) -  $(\delta_i, \delta_{i+1})$  dọc theo đường đi, và  $f(\delta_{i+1}) \geq f(\delta_i)$ , thì các giá trị của  $f(\delta_i)$  sẽ không giảm trên đường đi đó.

Lấy ví dụ đường đi của chúng ta như sau:



Trước hết, ta kiểm tra 2 node A và C.

☉ Từ S đến A:

- $f(S) = 0 + h(S) = 0 + 2$
- $f(A) = g(S) + \text{cost}(S, A) + h(A) = 0 + 1 + 4 = 5$

☉ Từ S đến C:

- Qua B:  $f(C) = g(S) + \text{cost}(S, B) + \text{cost}(B, C) + h(C) = 0 + 1 + 2 + 1 = 4$

Điều này vi phạm quy tắc hàm nhất quán vì  $f(C) < f(A)$  mặc dù C nằm sau A theo một đường đi trong đồ thị. Vì vậy ta có thể nói rằng, bất cứ khi nào một Node  $\delta_i$  được loại bỏ để mở rộng, đường đi tối ưu của nó đã được tìm thấy.

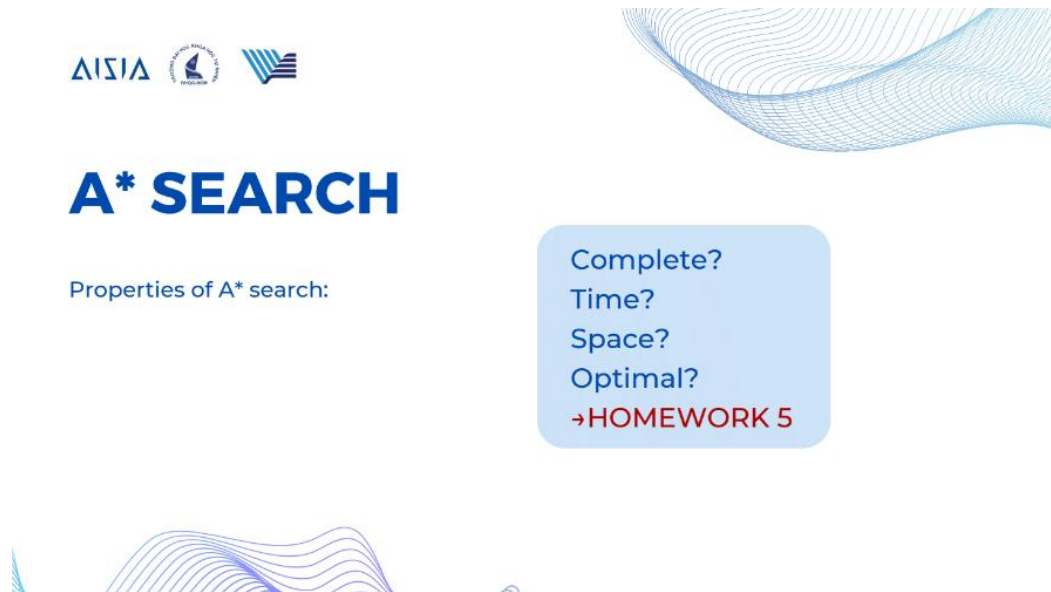
Giả sử điều này là sai - rằng khi  $\delta_i$  bị loại bỏ khỏi fringe, đường đi tìm đến  $\delta_i$  là không tối ưu. Có nghĩa là phải có tổ tiên nào đó của  $\delta_i$ ,  $\delta_{i+2}$  ở fringe chưa bao giờ được mở rộng nhưng nằm trên đường đi tối ưu tới  $\delta_i$ . Điều này là mâu thuẫn, vì ta đã chỉ ra rằng các giá trị của  $f$  dọc theo một đường đi là không giảm, và do đó  $\delta_{i+2}$  sẽ bị loại bỏ để mở rộng trước  $\delta_i$ .

Ta tiếp tục đảm bảo rằng đích đến tối ưu A luôn được mở rộng trước đích đến không tối ưu B. Có nghĩa là nếu A là Node tối ưu, thì  $A^*$  sẽ chọn A để mở rộng trước bất kỳ mục tiêu không tối ưu B. Điều này có nghĩa là nếu node A tối ưu thì  $f(A) < f(B)$  với  $h(A) = h(B) = 0$ . Đẳng thức chỉ xảy ra khi  $h$  là hàm đánh giá chấp nhận được, nhưng vì  $h$  là một hàm nhất quán nên nó cũng là hàm đánh giá chấp nhận được. Vì vậy ta luôn đảm bảo đích đến tối ưu A luôn được mở rộng trước đích đến không tối ưu B.

Tổng kết lại, khi một Node  $\delta_i$  được loại bỏ để mở rộng, đường đi tối ưu của nó đã được tìm thấy và luôn đảm bảo rằng, đường đi tối ưu đến node  $\delta_i$  luôn được mở rộng trước

đường đi đến một Node không tối ưu  $\psi$  nào đó. Nên khi thuật toán A\* tìm kiếm trên đồ thị với hàm nhất quán, nó sẽ luôn tìm ra đường đi tối ưu đến đích.

Bài 2:



Thuật toán A\* là một thuật toán kết hợp giữa Uniform Cost Search (Thuật toán tìm kiếm có chi phí) và Best First Search (Thuật toán tìm kiếm tốt nhất). Trong thuật toán A\*, ta sử dụng một hàm  $f(n)$  là hàm ước tính chi phí nhỏ nhất cho đường đi qua  $n$ . Trong đó,  $f(n) = g(n) + h(n)$ , với  $g(n)$  là chi phí từ Node Start đến Node  $n$ , và  $h(n)$  là chi phí ước tính nhỏ nhất của Node  $n$  đến Node Goal. Vì vậy, nếu chúng ta muốn tìm đường đi có chi phí nhỏ nhất, thì điều đầu tiên chúng ta phải làm là tìm  $g(n)$  và  $h(n)$  nhỏ nhất.

### **Tính hoàn thiện – Complete**

Giả sử có một đường đi từ điểm bắt đầu  $S$  đến mục tiêu  $G$ . Ta sẽ chứng minh rằng A\* sẽ tìm được mục tiêu nếu một đường đi tồn tại.

Trước hết, vì không gian trạng thái là hữu hạn, nên số lượng các Node có thể mở rộng là hữu hạn.

Với Mỗi lần mở rộng một nút  $n$ ,  $A^*$  sẽ thêm các nút con của  $n$  vào hàng đợi (fringe) nếu chúng chưa từng được mở rộng trước đó. Nếu một Node con đã nằm trong fringe nhưng có giá trị  $g(n)$  thấp hơn so với lần thêm trước, thì Node đó sẽ được cập nhật với giá trị  $g(n)$  mới, đảm bảo rằng đường đi ngắn nhất tới các Node con luôn được duyệt.

Với hàm đánh giá có thể chấp nhận được – admissible heuristics, nên nó không bao giờ đánh giá vượt quá chi phí thực tế. Do đó,  $A^*$  sẽ không bỏ qua một đường đi ngắn hơn hoặc tối ưu hơn để đến mục tiêu.

**Giả sử tồn tại một đường đi tới G.** Vì  $A^*$  liên tục mở rộng các nút có giá trị  $f(n)$  thấp nhất trước, nên khi  $G$  được chọn để mở rộng, đó là đường đi ngắn nhất tới  $G$ , và thuật toán sẽ dừng lại.

Trường hợp còn lại, nếu không có đường đi từ  $S$  đến  $G$ , thuật toán sẽ mở rộng tất cả các Node có thể còn lại và cuối cùng sẽ kết luận rằng không có đường đi tới  $G$  (khi mà hàng đợi không còn Node nào khác).

Vì vậy, ta luôn đảm bảo rằng  $A^*$  có tính hoàn thiện chỉ khi mà hàm  $h(n)$  của nó có thể chấp nhận được và trong một không gian trạng thái hữu hạn. Tức là số Node luôn đếm được, và chỉ cần như vậy thì sẽ luôn có một đường đi từ Node Start đến Node Goal, và  $A^*$  chắc chắn tìm ra đường đi đó.

### Tính Tối Ưu - Optimal

Cho một đường đi có các thuộc tính như sau:

☉  $S_k$  - Một đường đi con (sub-path) có độ dài  $k$  thuộc đường đi đến mục tiêu không tối ưu  $G_{sub}$

- $S_k$  đại diện cho một phần của đường đi không tối ưu đến mục tiêu. Đường đi này không phải là tốt nhất (có thể có chi phí lớn hơn đường đi tối ưu).

☉  $O_k$  - Một đường đi con có độ dài  $k$  thuộc đường đi tối ưu đến mục tiêu  $G_o$ :

- $O_k$  là một phần của đường đi tốt nhất (tối ưu) để đến mục tiêu. Đường đi tối ưu này có chi phí thấp nhất so với các đường đi khác từ điểm xuất phát đến mục tiêu.

⊙ **C(P)** - Tổng chi phí của đường đi P:

⊙ **h(P)** - Một hàm heuristic admissible ước lượng chi phí từ nút cuối cùng trong đường đi P đến mục tiêu:

⊙ **f(P) = C(P) + h(P)** - Là ước lượng tổng chi phí của đường đi P.

Ta có:

. Vì h là hàm Admissible  $\Rightarrow \forall j : f(O_j) \leq C(G_o)$ :

Nghĩa là chi phí ước tính của bất kỳ đường đi nào luôn nhỏ hơn hoặc bằng tổng chi phí của đường đi tối ưu

.  $C(G_{sub}) > C(G_o) \Rightarrow \exists m : C(S_m) > C(G_o)$

Nghĩa là đường đi tối ưu con có chi phí cao hơn đường đi tối ưu cha, hay tồn tại một vài đường đi con  $S_m$  của  $G_{sub}$  (ví dụ là  $S_m$ ) sao cho tổng chi phí của  $S_m$  luôn lớn hơn tổng chi phí của đường đi tối ưu. Nói cách khác, ở đâu đó trên đường đi đường đi tối ưu con sẽ tích lũy nhiều chi phí.

Kết hợp lại, ta được:

Với các đường đi con  $S_m$  của bất kỳ đường đi tối ưu con  $G_{sub}$ , ta được:

$$[C(S_m) > C(G_o) \geq f(O_j)] \Rightarrow [C(S_m) > f(O_j)] \Rightarrow [f(S_m) > f(O_j)]$$

Hay thuật toán luôn chọn đường đi ít chi phí nhất.

Nghĩa là tính tối ưu của  $A^*$  được đảm bảo vì các đường đi không tối ưu sẽ luôn "bão hòa" trước khi đạt đến nút mục tiêu và thuật toán sẽ tìm kiếm ở nơi khác. Vì vậy trường hợp xấu nhất có thể là: Tất cả các đường đi không tối ưu đều được khám phá đến điểm "bão hòa" của chúng trước khi cuối cùng phát hiện ra đường đi tối ưu đến mục tiêu.

Kết luận, với điều kiện là hàm heuristich(n) thỏa mãn các điều kiện nhất định,  $A^*$  Research vừa hoàn thiện vừa tối ưu.

## Time Complexity – Độ phức tạp thời gian

Khác với các thuật toán tìm kiếm Best First Search, Thuật toán A Star đã phát sinh thêm một hàm  $h(n)$  gọi là hàm Heuristic. Vì vậy, hàm Heuristic này ảnh hưởng đến độ phức tạp không gian và thời gian của thuật toán tìm đường đi A\*.

Một hàm  $h(n)$  tốt là một hàm heuristic có thể giúp giảm thiểu số lượng Node cần mở rộng trong quá trình tìm kiếm, giúp A\* tìm ra đường đi tối ưu nhanh hơn. Cụ thể, heuristic tốt có các đặc điểm sau:

**Admissible (chấp nhận được):** Heuristic này không bao giờ ước lượng vượt quá chi phí thực tế từ Node hiện tại đến mục tiêu. Điều này đảm bảo rằng A\* sẽ tìm được lời giải tối ưu. Tính chấp nhận được của heuristic được xem là điều kiện cần để A\* đảm bảo tính tối ưu.

**Consistent (nhất quán):** Heuristic này thỏa mãn điều kiện nhất quán, nghĩa là giá trị heuristic từ một Node đến mục tiêu không lớn hơn chi phí để đi đến nút kế cộng với giá trị heuristic từ nút kế đó đến mục tiêu. Điều này giúp tránh việc A\* phải mở rộng lại các nút đã đi qua, giúp thuật toán hoạt động hiệu quả hơn. Consistency thường được yêu cầu trong tìm kiếm trên đồ thị.

Một hàm heuristic tốt sẽ giúp A\* loại bỏ (prune) nhiều Node không cần thiết so với tìm kiếm không thông tin. Điều này làm giảm số lượng nút cần mở rộng trong không gian tìm kiếm.

Ta gọi chất lượng của heuristic có thể được biểu thị qua hệ số phân nhánh hiệu quả  $b^*$  có thể xác định bằng cách đo số lượng Node được sinh ra bởi quá trình mở rộng  $N$  và độ sâu của lời giải  $d$ . Một heuristic tốt có hệ số phân nhánh hiệu quả  $b^*$  thấp, lý tưởng nhất là  $b^* = 1$ . Điều này có nghĩa là số lượng Node mà A\* cần mở rộng giảm đáng kể so với tìm kiếm không thông tin, giúp tiết kiệm thời gian và tài nguyên tính toán:

Với  $N$  tổng số Node đã được mở rộng trong quá trình tìm kiếm cho đến khi tìm thấy giải pháp, và lập tổng  $N$  với Node gốc, ta được:

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

Trong đó  $(b^*)^k$  là các Node ở độ sâu  $k$ .

Phương trình tương đương  $N + 1 \sim O(b^{\varepsilon d})$

Đối với graph là 1 tree, thì độ phức tạp đa thức sẽ là:  $|h(n) - h^*(n)| = O(\log h^*(n))$ .

**Sai số nhỏ so với heuristic tối ưu  $h^*$ :** Để đạt hiệu quả cao nhất, sai số của heuristic (độ chênh lệch giữa  $h$  và  $h^*$ ) không nên tăng nhanh hơn logarit của  $h^*$ . Điều này sẽ giúp  $A^*$  đạt độ phức tạp thời gian đa thức thay vì hàm mũ, giúp quá trình tìm kiếm diễn ra nhanh hơn trong không gian lớn.

### Space Complexity – Độ phức tạp không gian

Độ phức tạp về không gian của  $A^*$  gần giống như của tất cả các thuật toán tìm kiếm đồ thị khác, vì nó giữ tất cả các Node được tạo trong bộ nhớ. Độ phức tạp không gian của  $A^*$  thường là  $O(b^{ed})$  tương tự như độ phức tạp thời gian trong trường hợp tệ nhất, trong đó  $b$  là hệ số nhánh (số lượng Node con) và  $d$  là độ sâu của giải pháp.  $A^*$  cần lưu trữ tất cả các Node đã được mở rộng cho đến khi tìm ra giải pháp, vì thuật toán có thể cần quay lại các nút này trong quá trình tìm kiếm.

$A^*$  lưu trữ tất cả các Node trong một hàng đợi mở (open list) và một hàng đợi đóng (closed list):

- **Hàng đợi mở (Open List):** chứa các Node đã được tạo ra nhưng chưa được mở rộng. Điều này bao gồm cả các Node đã được tính toán chi phí, vì  $A^*$  sẽ chọn Node có chi phí thấp nhất để mở rộng tiếp theo.
- **Hàng đợi đóng (Closed List):** chứa các Node đã được mở rộng và không cần phải xem xét lại.

Nếu hàm heuristic tốt (ví dụ, gần chính xác với chi phí thực tế để đến mục tiêu), thì  $A^*$  có thể loại bỏ nhiều nút không cần thiết, do đó giảm số lượng nút mà nó cần lưu trữ. Điều này dẫn đến độ phức tạp không gian có thể thấp hơn trong thực tế.



Câu 3:

Chứng minh mọi công thức trên logic mệnh đề đều có thể đưa về dạng chuẩn tắc.

**Dạng chuẩn tắc là gì:**

Dạng chuẩn tắc của một đối tượng toán học là một cách tiêu chuẩn để trình bày đối tượng đó dưới dạng một biểu thức toán học. Thông thường, nó là cách cung cấp cách biểu diễn đơn giản nhất của một đối tượng và cho phép nó được xác định theo một cách duy nhất.

Trong logic Boolean, các công thức, mệnh đề, đều được biểu hiện dưới dạng  $\{0, 1, \wedge, \vee\}$

Dạng chuẩn tắc được chia làm hai loại

1. Disjunctive Normal Forms (DNF) hay còn gọi là Sum of Products: dạng chuẩn tắc phân biệt.

Ví dụ:  $p \wedge q \wedge r \vee (p \wedge \sim q \wedge \sim r) \vee (\sim p \wedge \sim q \wedge r)$ .

2. Conjunctive Normal Forms (CNF) hay còn gọi là Products of Sums: Dạng chuẩn liên hợp.

Ví dụ:  $(P \vee Q) \wedge (Q \vee R) \wedge (\sim P \vee Q \vee \sim R)$

Vì vậy, ta sẽ chứng minh mọi công thức trên logic mệnh đề đều có thể đưa về CNF và DNF. Qua đó nó sẽ được đưa về dạng chuẩn tắc.

I. Chứng minh mọi công thức trên logic mệnh đề đều có thể đưa về dạng CNF.

**Giả sử chúng ta có một tập hợp các hằng tử  $A=\{a_1, \dots, a_n\}$**  (có thể hiểu là các mệnh đề đúng hoặc sai, hoặc các công thức đúng hoặc sai, ví dụ như “ $3 > 4$ ”).

Ta cần chứng minh rằng: Với bất kỳ mệnh đề  $p$  nào (được tạo thành từ các phép nối logic, các hằng tử và các mệnh đề khác), thì  $p$  có thể được chuyển đổi thành một phép lặp kết hợp dạng  $p_1 \vee \dots \vee p_k$ , trong đó mỗi  $p_i = a_1^{\epsilon_1} \wedge \dots \wedge a_n^{\epsilon_n}$ ,  $a^1 = a$  và  $a^{-1} = \neg a$ .

Ta sử dụng kiến thức về “bảng giá trị chân lý”, lưu ý rằng một hàng trong bảng giá trị chân lý chỉ chứa các hằng tử là một sự phân bố chân lý  $t: A \rightarrow \{0, 1\}$ . Điều này nghĩa là ta cần xem xét các hàng hay các sự phân bố chân lý mà làm cho mệnh đề  $p$  đúng và sử dụng nó vào  $p_i$ .

Liệt kê tất cả các phân bố chân lý  $t_1, \dots, t_k$

Xét phân bố chân lý  $t_j$

Với  $p_i = a_1^{\epsilon_1} \wedge \dots \wedge a_n^{\epsilon_n}$ ,  $p_i$  đúng khi  $t_j$  đúng hay  $t_j(p_i) = 1$

Đặt  $\epsilon_i = 1$  nếu  $t_j(p_i) = 1$  và  $\epsilon_i = -1$  nếu  $t_j(p_i) = 0$ . Điều này chỉ ra rằng  $p_i$  là 1 với hàng hiện tại của bảng giá trị chân lý mà chúng ta đang xem xét.

Vì  $p$  là một phép lặp kết hợp (OR) nên phải có  $t_j(p) = 1$  vì  $t_j(p_i) = 1$ . Do đó,  $t(p) = 1$  hay  $t = t_j$ . Vì vậy  $\epsilon_i = 2t(a_i) - 1 = 1$

Nên  $p_i = a_1^{\epsilon_1} \wedge \dots \wedge a_n^{\epsilon_n} = a_1^{2t(a_1)-1} \wedge \dots \wedge a_n^{2t(a_n)-1}$

Chứng minh hoàn tất.

II. Chứng minh mọi công thức trên logic mệnh đề đều có thể đưa về dạng CNF tương tự như DNF.

Vì vậy, mọi công thức trên logic mệnh đề đều có thể đưa về CNF và DNF. Qua đó nó sẽ được đưa về dạng chuẩn tắc.

Bài làm của em đã xong, cảm ơn cô đã đọc và chấm. Em xin chúc cô một ngày mới thật vui ạ.

