

Cheatsheet: Advanced Logic và Program Verification

1 Tổng quan về Hoare Logic

Hoare Triple: Biểu thức có dạng:

$$\{P\} C \{Q\}$$

Trong đó:

- P : Tiền điều kiện (*precondition*) — điều kiện phải thỏa mãn trước khi chương trình C được thực thi.
- C : Chương trình hoặc đoạn mã cần kiểm tra.
- Q : Hậu điều kiện (*postcondition*) — điều kiện cần thỏa mãn sau khi chương trình C hoàn tất.

2 Phân loại tính đúng đắn

2.1 Đúng đắn từng phần (Partial Correctness)

Chứng minh tính đúng đắn từng phần yêu cầu đảm bảo rằng nếu chương trình kết thúc, hậu điều kiện sẽ được thỏa mãn. Ký hiệu là:

$$\models_{\text{par}} \{P\} C \{Q\}$$

Trong đó \models_{par} biểu diễn tính đúng đắn từng phần.

2.2 Đúng đắn tổng thể (Total Correctness)

Chứng minh chương trình vừa đúng và vừa kết thúc. Điều này đảm bảo chương trình không chỉ thực hiện chính xác mà còn phải dừng lại. Ký hiệu là:

$$\models_{\text{tot}} \{P\} C \{Q\}$$

Trong đó \models_{tot} biểu diễn tính đúng đắn tổng thể.

3 Các quy tắc Hoare cơ bản

3.1 Quy tắc gán (Assignment Rule)

$$\{P[E/x]\}x = E\{P\}$$

Ví dụ: Nếu P là $y = 2$, thì sau lệnh $x = y$, ta có $x = 2$.

3.2 Quy tắc tuân tự (Composition Rule)

$$\{P\}C_1\{Q\}, \quad \{Q\}C_2\{R\} \implies \{P\}C_1; C_2\{R\}$$

Thực hiện tuân tự C_1 và C_2 sẽ biến P thành R nếu C_1 biến P thành Q , và C_2 biến Q thành R .

3.3 Quy tắc điều kiện (Conditional Rule)

$$\{P \wedge B\}C_1\{Q\}, \quad \{P \wedge \neg B\}C_2\{Q\} \implies \{P\}\text{if } B \text{ then } C_1 \text{ else } C_2\{Q\}$$

Nếu B đúng, thực thi C_1 ; nếu B sai, thực thi C_2 , và kết quả của cả hai thỏa mãn Q .

4 Bất biến vòng lặp (Loop Invariant)

4.1 Cách xác định bất biến vòng lặp

Bất biến vòng lặp là điều kiện luôn đúng ở đầu mỗi lần lặp của vòng lặp. Để xác định bất biến vòng lặp, thực hiện các bước sau: 1. **Xác định điều kiện khởi tạo**: Kiểm tra điều kiện nào đúng trước khi vòng lặp bắt đầu. 2. **Kiểm tra tính duy trì**: Chứng minh rằng nếu bất biến đúng ở đầu một lần lặp, nó vẫn sẽ đúng ở cuối lần lặp đó. 3. **Kiểm tra điều kiện kết thúc**: Khi vòng lặp kết thúc, sử dụng bất biến và điều kiện dừng của vòng lặp để chứng minh hậu điều kiện.

Ví dụ: Xét chương trình tính giai thừa:

```
1 y = 1;
2 z = 0;
3 bluewhile (z != x) {
4     z = z + 1;
5     y = y * z;
6 }
```

Bất biến vòng lặp là: $y = z!$, tức là tại mỗi bước lặp, y bằng giai thừa của z .

4.2 Cách chứng minh bất biến

Chứng minh bất biến vòng lặp yêu cầu thực hiện ba bước: 1. **Khởi tạo**: Chứng minh rằng bất biến đúng trước khi bắt đầu vòng lặp.

$$\text{Trước vòng lặp: } y = 1, z = 0 \implies y = z!$$

2. **Duy trì**: Chứng minh rằng nếu bất biến đúng tại đầu mỗi lần lặp, nó sẽ vẫn đúng ở cuối lần lặp.

$$\text{Cuối mỗi lần lặp: } y = z! \implies y = (z + 1)!$$

3. **Kết thúc**: Khi vòng lặp kết thúc, sử dụng bất biến để chứng minh hậu điều kiện.

$$z = x \Rightarrow y = x!$$

5 Hàm chặn (Bound Function)

Hàm chặn là hàm số nguyên không âm, đảm bảo rằng giá trị của nó giảm dần sau mỗi lần lặp và vòng lặp sẽ kết thúc khi giá trị này đạt đến 0.

5.1 Cách xác định hàm chặn

- **Xác định biến thay đổi trong vòng lặp**: Tìm một biến mà giá trị của nó giảm sau mỗi lần lặp.
- **Kiểm tra điều kiện không âm**: Hàm chặn không được âm trong quá trình vòng lặp.
- **Chứng minh hàm giảm dần**: Sau mỗi lần lặp, giá trị của hàm phải giảm.

Ví dụ: Với chương trình tính giao thừa:

$$E = x - z$$

Hàm chặn E giảm mỗi lần lặp do z tăng dần, và $E = x - z$ không thể âm vì $z \leq x$.

6 Tiền điều kiện yếu (Weak Precondition)

Tiền điều kiện yếu (*weak precondition*) là điều kiện ít chặt chẽ hơn nhưng đủ để đảm bảo hậu điều kiện vẫn đúng sau khi thực thi đoạn mã.

6.1 Cách xác định tiền điều kiện yếu

- **Xác định hậu điều kiện**: Định nghĩa điều kiện cần thỏa mãn sau khi đoạn mã chạy.
- **Tìm điều kiện yếu nhất trước khi thực thi**: Điều kiện này phải đảm bảo hậu điều kiện đúng sau khi đoạn mã thực thi.
- **Chứng minh tiền điều kiện**: Chứng minh rằng nếu tiền điều kiện yếu được thỏa mãn, hậu điều kiện cũng sẽ thỏa mãn.

Ví dụ: Xét đoạn mã:

```
1 y = y + 1;
```

Hậu điều kiện là $y < 4$. Tiền điều kiện yếu cần tìm là $y < 3$ để đảm bảo rằng sau khi thực thi $y = y + 1$, ta có $y < 4$.

7 Advanced Logic

7.1 Logic bậc cao (Higher-order Logic)

Logic bậc cao là một hệ thống logic mạnh mẽ cho phép lượng hóa không chỉ trên các biến, mà còn trên các vị từ và hàm. Nó cung cấp một công cụ biểu diễn mạnh mẽ cho các tính chất phức tạp của chương trình.

7.1.1 Đặc điểm của Logic bậc cao

- Lượng hóa trên hàm và vị từ:** Cho phép biểu diễn các khái niệm như "tồn tại một hàm f sao cho..."
- Lambda abstraction:** Cho phép định nghĩa các hàm ẩn danh, ví dụ: $\lambda x.x+1$
- Kiểu dữ liệu đa hình:** Hỗ trợ các kiểu dữ liệu tổng quát như $List<T>$
- Curry-Howard isomorphism:** Thiết lập mối quan hệ giữa các chương trình và chứng minh

7.1.2 Ví dụ Mở rộng

1. Định nghĩa tính chất "bị chặn trên" cho một hàm số thực:

$$\text{bounded_above} = \lambda f : \mathbb{R} \rightarrow \mathbb{R}. \exists M : \mathbb{R}. \forall x : \mathbb{R}. f(x) \leq M$$

2. Định nghĩa khái niệm "liên tục" cho hàm số thực:

$$\text{continuous} = \lambda f : \mathbb{R} \rightarrow \mathbb{R}. \forall \epsilon > 0. \forall x : \mathbb{R}. \exists \delta > 0. \forall y : \mathbb{R}. |x - y| < \delta \Rightarrow |f(x) - f(y)| < \epsilon$$

7.1.3 Ứng dụng Cụ thể trong Xác minh Chương trình

- **Xác minh thuật toán đê quy:** Sử dụng các vị từ bậc cao để định nghĩa các bất biến phức tạp.
- **Chứng minh tính đúng đắn của trình biên dịch:** Sử dụng các hàm bậc cao để biểu diễn các phép biến đổi chương trình.
- **Xác minh các thuộc tính bảo mật:** Sử dụng lượng hóa bậc cao để biểu diễn các chính sách bảo mật phức tạp.

7.2 Bất biến Suy diễn (Inductive Invariants)

Bất biến suy diễn là một công cụ quan trọng trong việc chứng minh tính đúng đắn của các vòng lặp và hệ thống phản ứng. Nó cung cấp một cách để "cắt" không gian trạng thái vô hạn thành một tập hữu hạn các trạng thái đại diện.

7.2.1 Định nghĩa Hình thức và Giải thích

Cho một hệ thống chuyển trạng thái (S, \rightarrow, I) với:

- S : tập trạng thái
- \rightarrow : quan hệ chuyển trạng thái
- I : $S \rightarrow S$: tập trạng thái khởi tạo

Một vị từ I trên S là một bất biến suy diễn nếu:

1. $s \in S : I(s)$ (Điều kiện khởi tạo)
2. $s, s' \in S : I(s) \wedge (s \rightarrow s') \rightarrow I(s')$ (Điều kiện bảo toàn)

Giải thích:

- Điều kiện 1 đảm bảo rằng I đúng cho mọi trạng thái khởi tạo.
- Điều kiện 2 đảm bảo rằng nếu I đúng tại một trạng thái, nó sẽ tiếp tục đúng sau mỗi bước chuyển trạng thái.

7.2.2 Ví dụ Mở rộng: Thuật toán Tìm kiếm Nhị phân

Xét thuật toán tìm kiếm nhị phân trên một mảng đã sắp xếp:

```
1 blueint binarySearch(blueint[] arr, blueint target) {  
2     blueint left = 0, right = arr.length - 1;  
3     bluewhile (left <= right) {  
4         blueint mid = (left + right) / 2;  
5         blueif (arr[mid] == target) bluereturn mid;  
6         blueif (arr[mid] < target) left = mid + 1;  
7         blueelse right = mid - 1;  
8     }  
9     bluereturn -1;  
10 }
```

Bất biến suy diễn cho vòng lặp này là:

$$I \equiv (0 \leq left \leq right + 1 \leq arr.length) \wedge (\forall i. 0 \leq i < left \Rightarrow arr[i] < target) \wedge (\forall i. right < i < arr.length \Rightarrow arr[i] > target)$$

Giải thích bất biến:

- Phần đầu tiên đảm bảo rằng các chỉ số left và right luôn hợp lệ.
- Phần thứ hai đảm bảo rằng tất cả các phần tử bên trái của left đều nhỏ hơn target.
- Phần thứ ba đảm bảo rằng tất cả các phần tử bên phải của right đều lớn hơn target.

7.2.3 Quy trình Chứng minh Tính Đúng đắn Chi tiết

1. **Chứng minh I đúng trước khi vòng lặp bắt đầu:** - Ban đầu: $left = 0, right = arr.length - 1$ - Điều này thỏa mãn phần đầu tiên của I - Các phần còn lại của I là đúng một cách tự nhiên vì không có phần tử nào ở bên trái của left hoặc bên phải của right

2. **Chứng minh I được bảo toàn qua mỗi lần lặp:** - Giả sử I đúng tại đầu một lần lặp - Xét ba trường hợp: $arr[mid] == target$, $arr[mid] < target$, $arr[mid] > target$ - Trong mỗi trường hợp, chứng minh rằng I vẫn đúng sau khi cập nhật left hoặc right

3. **Sử dụng I để chứng minh tính đúng đắn:** - Khi vòng lặp kết thúc, ta có $left > right$ - Kết hợp với I, ta có thể kết luận rằng nếu target có trong mảng, nó phải ở vị trí đã được trả về

7.3 Suy luận bằng Logic Quy nạp (Inductive Reasoning)

Suy luận quy nạp là một kỹ thuật chứng minh mạnh mẽ, đặc biệt hữu ích trong việc chứng minh tính chất của các chương trình đệ quy và các cấu trúc dữ liệu vô hạn.

7.3.1 Các Dạng Quy nạp

1. **Quy nạp Toán học Đơn giản:** Để chứng minh $P(n)$ đúng với mọi $n \in \mathbb{N}$:

1. Chứng minh $P(0)$ đúng (Trường hợp cơ sở)
2. Chứng minh $\forall k \in \mathbb{N} : P(k) \Rightarrow P(k + 1)$ (Bước quy nạp)

2. **Quy nạp Mạnh:** Để chứng minh $P(n)$ đúng với mọi $n \in \mathbb{N}$:

1. Chứng minh $P(0)$ đúng (Trường hợp cơ sở)
2. Chứng minh $\forall k \in \mathbb{N} : (\forall i < k, P(i)) \Rightarrow P(k)$ (Bước quy nạp mạnh)
3. **Quy nạp Cấu trúc:** Sử dụng cho các cấu trúc dữ liệu để quy như danh sách liên kết hoặc cây.
 1. Chứng minh P đúng cho cấu trúc cơ bản (ví dụ: danh sách rỗng)
 2. Chứng minh nếu P đúng cho các cấu trúc con, thì P đúng cho cấu trúc lớn hơn

7.3.2 Ví dụ Mở rộng: Chứng minh Tính Đúng đắn của Quicksort

Xét thuật toán Quicksort:

```
1 bluevoid quicksort(blueint[] arr, blueint low, blueint high) {  
2     blueif (low < high) {  
3         blueint pivot = partition(arr, low, high);  
4         quicksort(arr, low, pivot - 1);  
5         quicksort(arr, pivot + 1, high);  
6     }  
7 }
```

Mệnh đề cần chứng minh: Với mọi $n \geq 0$, $\text{quicksort}(arr, 0, n-1)$ sắp xếp mảng $arr[0..n-1]$.

Chứng minh bằng quy nạp theo độ dài của mảng:

1. **Trường hợp cơ sở ($n = 0$ hoặc 1):** Mảng rỗng hoặc có một phần tử đã được sắp xếp.
2. **Giả thuyết quy nạp:** Giả sử quicksort hoạt động đúng cho mọi mảng có độ dài $< n$.
3. **Bước quy nạp:** Chứng minh quicksort hoạt động đúng cho mảng độ dài n :
 - Sau bước partition, pivot ở đúng vị trí.
 - Áp dụng giả thuyết quy nạp cho hai lối gọi đệ quy:
 - $\text{quicksort}(arr, low, pivot - 1)$ sắp xếp đúng phần bên trái
 - $\text{quicksort}(arr, pivot + 1, high)$ sắp xếp đúng phần bên phải
 - Kết hợp ba phần này, ta có mảng đã được sắp xếp.

7.3.3 Ứng dụng Nâng cao trong Xác minh Chương trình

- **Chứng minh tính kết thúc của chương trình:** Sử dụng quy nạp trên "độ đo tiến triển" của chương trình.
- **Xác minh các thuộc tính an toàn:** Sử dụng quy nạp để chứng minh rằng các trạng thái không an toàn không thể đạt được.
- **Chứng minh tính đúng đắn của các giao thức mạng:** Sử dụng quy nạp trên số lượng bước giao tiếp.
- **Xác minh tính đúng đắn của các trình biên dịch:** Sử dụng quy nạp cấu trúc trên cây cú pháp của chương trình nguồn.