Ban đầu, hệ thống sẽ cố gắng tìm một hàm có khả năng đưa ra kết quả là $log(x^2)$ từ yêu cầu ban đầu là $x \geq 0$. Sử dụng cơ chế tính toán tương tự giữa các đặc tả như mô tả trong Phần 3.1, bộ lập kế hoạch xác định được hàm phù hợp nhất là f_3. Để có thể tạo ra kết quả là $log(x^2)$, hàm f_3 yêu cầu ngõ nhập phải thoả điều kiện $x<0$. Như vậy bộ lập kế hoạch cần tìm một hàm thoả mãn *subgoal* : $\{x \geq 0 \Rightarrow x<0\}$. Một lần nữa, việc tính toán độ tương tự giữa các hàm sẽ xác định hàm phù hợp nhất cho subgoal này là f_2[2]. Đến đây, một lời giải tổng hợp đã được hình thành đầy đủ như sau: $f_3(f_2(x))$.

3.3 Bộ tích hợp hàm mới vào thư viện đặc tả

Chức năng này đóng vai trò giúp cho thư viện đặc tả các hàm của chúng ta phong phú thêm, giúp cho những lần sau khi gặp những yêu cầu tương tự, hệ thống sẽ có sẵn các hàm/ chuỗi các hàm đáp ứng yêu cầu, giúp tối ưu hiệu suất của hệ thống.

Kết quả đầu ra của khung thức này là một đoạn mã hoặc một nguyên mẫu phần mềm (prototype) trong đó có chứa các lời gọi hàm cần thiết theo yêu cầu ban đầu của người sử dụng. Ví dụ, sau khi tìm ra lời giải tổng hợp cho yêu cầu được mô tả trong Phần 3.2, đặc tả và lời giải này lại được tiếp tục lưu trữ vào thư viện như một hàm mới (hàm f_4) để phục vụ cho những yêu cầu tìm kiếm và tổng hợp lời giải mới sau này.

4 MỘT SỐ BÀI TOÁN MINH HOẠ

Sau đây là một số ví dụ minh họa việc sử dụng đặc tả JML để đặc tả cho các yêu cầu tìm kiếm và gọi hàm tự động từ một số trường hợp thực tế.

Trường hợp 1: Tìm ngay được một hàm phù hợp, đáp ứng yêu cầu.

Yêu cầu 1: "Cho một hình lập phương, cạnh có độ dài đại số là *a*. Tính thể tích của hình lập phương đó".

Chúng ta đặc tả yêu cầu trên như sau:

```
//@ requires a > 0
/*@ ensures \result > 0 &&
 @JMLDouble.approximatelyEqualTo
  @     (a^3, \result, 0.001);
 */
```

Bằng cách phân tích độ tương thích giữa hậu điều kiện (***ensure***) của yêu cầu với hậu điều kiện của các hàm có trong thư viện thì chúng ta nhận được một hàm phù hợp ngay, đó là hàm p*ow()* của lớp *java.lang.Math*. Hàm *pow()* này có đặc tả JML như sau[3]:

```
/*@ ensures
 @JMLDouble.approximatelyEqualTo
 @    (a^b, \result, 0.000001);
*/
public static double pow
            (double a, double b);
```

Tiền điều kiện của hàm *pow()* được bỏ qua nghĩa là tiền điều kiện luôn đúng. Đó đó, nó thỏa mãn với tiền điều kiện của yêu cầu. Như vậy *pow(a,3)* là hàm cần gọi.

Yêu cầu 2: "Chèn một phần tử *x* vào đầu Stack *S*".

Đặc tả JML cho yêu cầu:

```
//@ requires S.size<S.maxSize-1;
/*@ ensures
S.size==\old(S.size+1)
  @         &&
S.firstElement()==x;
 */
```

Với phương pháp tương tự như yêu cầu 1, khung thức cũng dễ dàng tìm được 1 hàm thỏa mãn yêu cầu trên là hàm *push()* của lớp Stack, với đặc tả như sau:

```
/*@ requires size < maxSize - 1;
 @ assignable size;
 @ ensures size = \old(size + 1)
          && S.firstElement()==x;
 @ ensures_redundantly (\forall
 @       int i; 0≤i && i<size-1;
 @element(i)==\old(element(i)));
 @*/
public void push(Object x);
```

Trường hợp 2: Cần tinh chế đặc tả mới có thể tìm ra lời gọi hàm.

Yêu cầu 3: "Trả về vị trí của phân tử *x* trong mảng *a*. Nếu phần tử *x* không tồn tại trong mảng *a* thì trả về -1".

Đặc tả JML cho yêu cầu trên như sau:

[2] Để tính toán được rằng đặc tả của f_2 ($x \Rightarrow -x$) là tương đương với đặc tả của subgoal, chúng ta sẽ cần đến một *công cụ chứng minh* (prover). Đã có rất nhiều prover được phát triển trong cộng đồng nghiên cứu, nhưng trong khuôn khổ của bài báo này, chúng tôi không thảo luận đến các prover.

[3] Đặc tả này đã có trong thư viện về đặc tả của các hàm thư viện của Java.