PHÂN TÍCH KHẢ NĂNG KIỂM THỬ CÁC ĐƠN VỊ PHẦN MỀM

TESTABILITY ANALYSIS OF SOFTWARE COMPONENTS

NGUYỄN THANH BÌNH
Trường Đại học Bách khoa, Đại học Đà Nẵng

TÓM TẮT

Bài báo trình bày phương pháp phân tích khả năng kiểm thử các đơn vị phần mềm để đánh giá và tiên lượng những khó khăn gặp phải trong quá trình kiểm thử phần mềm. Để phân tích khả năng kiểm thử, chúng tôi đề xuất giải pháp sử dụng kỹ thuật biểu diễn gán đơn tĩnh SSA bằng cách chuyển các đơn vị kiểm thử sang một dạng biểu diễn luồng dữ liệu. Kết quả phân tích một mặt giúp người kiểm thử phân bổ tài nguyên kiểm thử hợp lí hơn, một mặt giúp người thiết kế khắc phục những sự cố mất mát thông tin và có thể sửa đổi lại chương trình nguồn.

ABSTRACT

This paper presents the result of testability analysis of software components to evaluate and predict some difficulties encountered in the testing phase. In our solution, we concentrate on testability analysis of software components. We propose the use of the Static Single Assignment (SSA) form to transform a software component into a data-flow representation, and testability analysis is based on this SSA form. This results helps designers during the specification phases of the components, and testers during the testing phases to evaluate and eventually modify the components.

1. Đặt vấn đề

Trong tiến trình phần mềm, giai đoạn *kiểm thử* (testing) đóng vai trò quan trọng. Phần mềm càng lớn và càng phức tạp, thủ tục kiểm thử càng đòi hỏi nhiều thời gian và công sức. Để nâng cao hiệu quả sản phẩm, những người tham gia xây dựng phần mềm cần có sớm thông tin về hệ thống chương trình cần kiểm thử. Những thông tin này có được nhờ *phân tích khả năng kiểm thử* (PTKNKT) phần mềm. Một mặt, kết quả phân tích cho phép phát hiện ra những phần chương trình nào vừa chứa đựng nhiều lỗi, vừa khó kiểm thử, những đơn vị chương trình nào thì nên đầu tư nhiều công sức. Mặt khác, PTKNKT giúp phân bổ tài nguyên kiểm thử tốt hơn, hoặc có thể yêu cầu người thiết kế sửa đổi lại nội dung thiết kế để cải thiện *khả năng kiểm thử* (KNKT). PTKNKT cũng đưa ra chỉ số chất lượng nhờ phép đo độ phức tạp khi kiểm thử phần mềm.

Trước đây, PTKNKT phần mềm mới chỉ là đánh giá độ phức tạp của các chương trình. McCabe [1] và Nejmeh [2] đánh giá độ phức tạp dựa trên số lượng dòng lệnh thực hiện và số lượng dữ liệu kiểm thử, nhưng chưa đề cập đến KNKT. Sau đó, Freedman [3] là người đầu tiên đưa vào khái niệm KNKT, bằng cách dựa trên *khả năng quan sát* (observability) và *khả năng điều khiển* (controllability). Phương pháp này đánh giá chương trình dựa trên miền dữ liệu vào và miền dữ liệu ra. Hai tác giả Voas và Miller [4] định nghĩa KNKT phần mềm là khả năng phần mềm bị sự cố (failure) do gặp lỗi khi kiểm thử và cũng đánh giá chương trình qua miền dữ liệu vào và miền dữ liệu ra. Trong phạm vi đánh giá KNKT các phần mềm giao tiếp, Petrenko cùng nhóm tác giả [5] sử dụng các ôtômat hữu hạn, Karoui và nhóm tác giả [6] sử dụng các đặc tả quan hệ, v.v...

Gần đây, Le Traon và Robach [7] phát triển phương pháp PTKNKT sử dụng công cụ SATAN (System's Automatic Testability Analysis). Các tác giả này xây dựng một mô hình

chức năng gồm các đồ thị có hướng phù hợp để biểu diễn truyền thông tin theo thiết kế *luồng dữ liệu* (data flow). Đó là các biểu đồ *đặc tả nhờ máy tính* (Computer Aided Specification) mà mỗi biểu đồ CAS là một tập hợp các đơn thể chương trình được kết nối với nhau nhờ các luồng dữ liệu. Dữ liệu đầu ra của đơn thể này là dữ liệu đầu vào của một đơn thể khác. Do chỉ áp dụng được cho thiết kế luồng dữ liệu, hạn chế của phương pháp này là không áp dụng được để PTKNKT các đơn thể chương trình được viết trong các ngôn ngữ mệnh lệnh, chẳng hạn như ngôn ngữ C.

Trong bài báo này, chúng tôi đề xuất phương pháp PTKNKT phần mềm sử dụng biểu đồ CAS. Các đơn thể chương trình được viết trong ngôn ngữ C, do công ty Thales Avionics cung cấp. Để áp dụng được công cụ SATAN, chúng tôi đề xuất sử dụng *phép gán đơn tĩnh* SSA (Static Single Assignment) [10], [11], [12] để chuyển đổi mã nguồn trong của các chương trình sang dạng biểu diễn luồng dữ liệu. Sử dụng kết quả PTKNKT, người thiết kế có thể sửa đổi lại nội dung thiết kế, người kiểm thử có thể có giải pháp phân bố tài nguyên kiểm thử và chọn lựa chiến lược kiểm thử tốt hơn.

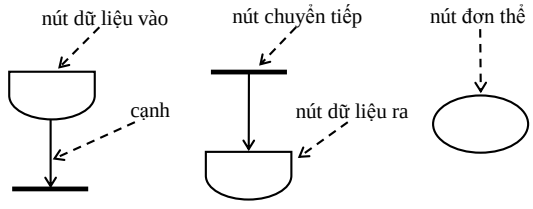
Nội dung chính của bài báo như sau: phần 2 giới thiệu ngắn gọn mô hình phân tích và cách đo KNKT sử dụng công cụ SATAN; phần 3 trình bày kỹ thuật gán đơn tĩnh SSA; phần 4 chỉ ra cách áp dụng công cụ SATAN để phân tích một đơn thể chương trình và đánh giá kết quả.

2. Phân tích khả năng kiểm thử một hệ thống phần mềm

Trước tiên, chúng tôi trình bày mô hình chức năng dựa trên thiết kế luồng dữ liệu [8] và sử dụng mô hình này để đo KNKT, đánh giá lưu lượng truyền tin, hay lưu chuyển luồng dữ liệu, trong các đồ thị có hướng của mô hình.

2.1. Mô hình chức năng

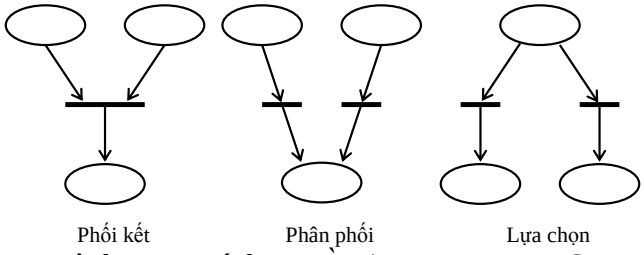
Nguyên tắc mô hình hóa truyền tin là biểu diễn đồng thời luồng điều khiển và luồng dữ liệu trong cùng một đồ thị, gọi là *đồ thị truyền tin ITG* (Information Transfer Graph). Đồ thị có hai loại nút là *nút thông tin* (place) và *nút chuyển tiếp* (transition). Các nút thông tin có thể là *các đơn thể* (module) biểu diễn các toán tử, đơn vị, câu lệnh..., các *nút dữ liệu vào* và các *nút dữ liệu ra* của hệ thống. Trong biểu diễn đồ thị, các đơn thể có hình e-líp; các nút dữ liệu vào/ra có dạng một nửa dưới hình e-líp; các nút chuyển tiếp là các đoạn thẳng nằm ngang (hình 1). Các cạnh nối liền các nút của ITG. Đồ thị được bắt đầu bởi các nút dữ liệu vào và được kết thúc bởi các nút dữ liệu ra.



Hình 1. Các loại nút của đồ thị truyền tin ITG.

Các nút chuyển tiếp phục vụ truyền tin giữa các nút thông tin. Có ba cách truyền tin (hình 2):

- *Phối kết*: nhiều nút nguồn chuyển tới một nút đích;
- *Phân phối:* chỉ một nút trong số các nút nguồn khác nhau chuyển tới một nút đích;
- *Lựa chọn*: một nút nguồn có thể đi đến nhiều nút đích khác nhau.

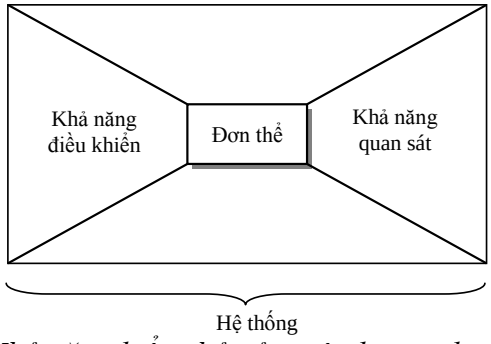


Hình 2. Ba cách truyền tin trong một ITG.

Đồ thị truyền tin ITG dùng để xác định các đường truyền tin, hay được gọi là *luồng dữ liệu* (dataflow). Mỗi luồng chứa tập hợp các nút (nút thông tin và nút chuyển tiếp) và các cạnh. Luồng xuất phát từ một vài nút dữ liệu vào, qua các nút đơn thể để đến một nút dữ liệu ra. Mỗi luồng có thể được xem như một đồ thị con, hay hệ thống con, hoạt động độc lập so với phần còn lại của hệ thống.

2.2. Đo khả năng kiểm thử

Phân tích cách truyền tin giữa các nút trong mỗi luồng của của đồ thị ITG, KNKT được đo dựa trên khả năng điều khiển và khả năng quan sát của mỗi đơn thể chương trình trong hệ thống đang xét. Khả năng điều khiển được đánh giá qua lượng thông tin đến từ đầu vào. Khả năng quan sát được đánh giá qua lượng thông tin nhận được ở đầu ra (xem hình 3). Từ đó, KNKT của một đơn thể là lượng thông tin thất lạc khi đi từ miền dữ liệu vào, qua đơn thể đến đầu ra của hệ thống trên mỗi luồng.



Hình 3. Khả năng kiểm thử của một đơn vị chương trình.

Gọi I_F là đầu vào và O_F là đầu ra của luồng F, I_M là đầu vào và O_M là đầu ra của đơn thể M, khả năng điều khiển của đơn thể M trong luồng F được tính bởi công thức:

$$CO_F(M) = \frac{T(I_F, I_M)}{C(I_M)} \quad (1)$$

trong đó, $T(I_F, I_M)$ là lượng thông tin lớn nhất có thể mà đơn thể M nhận được từ đầu vào I_F của luồng F và $C(I_M)$ là toàn bộ thông tin mà đơn thể M có thể nhận được nếu nó đứng độc lập.

Khả năng quan sát của đơn thể M trong luồng F được tính bởi công thức tương tự (1):

$$OB_F(M) = \frac{T(O_M, O_F)}{C(O_M)} \quad (2)$$

trong đó $T(O_F,, O_M)$ là lượng thông tin lớn nhất mà đầu ra O_F của luồng F có thể nhận được từ đầu ra O_M của đơn thể M và $C(I_M)$ là toàn bộ thông tin mà đơn thể M có thể sinh ra từ đầu ra O_M.

Khả năng điều khiển $CO_F(M)$ và khả năng quan sát $OB_F(M)$ có giá trị nằm trong đoạn [0,1], là tốt nhất nếu có giá trị 1, là kém nhất nếu có giá trị 0.

KNKT của đơn thể *M* trong luồng *F* là một hàm theo $CO_F(M)$ và $OB_F(M)$ (xem chi tiết ở [8]):

$$TE_F(M) = f(CO_F(M), OB_F(M)) \qquad (3)$$

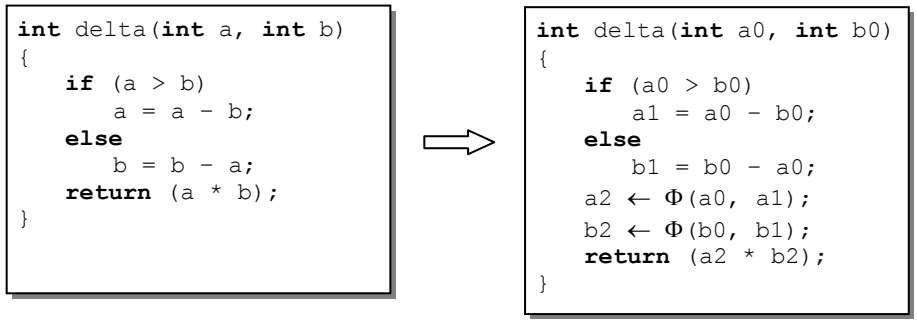
3. Kỹ thuật gán đơn tĩnh SSA

Kỹ thuật gán đơn tĩnh SSA là một dạng biểu diễn trung gian dùng để xây dựng các luồng dữ liệu cho đồ thị ITG. Kỹ thuật được SSA sử dụng chủ yếu cho các thuật toán tối ưu mã của các chương trình dịch viết trong một ngôn ngữ mệnh lệnh.

Một chương trình được chuyển sang dạng SSA qua hai bước. Trong bước thứ nhất, các hàm đặc biệt Φ được đưa vào tại mỗi *nút nối* (join node) trong đồ thị luồng điều khiển của chương trình. Một hàm Φ tại nút *X* có dạng $V \leftarrow \Phi(R, S, \ldots)$, trong đó *V*, *R*, *S*, … là các biến. Số lượng các biến này là số nhánh rẽ đứng trước nút *X*. Hàm Φ kết hợp các giá trị khác nhau của một biến để tạo ra một giá trị mới phụ thuộc vào nhánh rẽ. Trong bước thứ hai, các biến *R*, *S*, … được đổi tên thành V_1, V_2,... sao cho V_i chỉ xuất hiện trong một phép gán duy nhất trong toàn bộ chương trình.

Ví dụ, xét chương trình gốc là hàm *delta*, sau khi chuyển sang biểu diễn SSA, hai hàm Φ được chèn vào sau câu lệnh điều kiện để xác định giá trị của hai biến *a* và *b*, sau đó hai biến này được đổi tên thành các biến có chỉ số (hình 4).

Dạng biểu diễn SSA có đặc điểm là mỗi biến chỉ xuất hiện trong một định nghĩa hay trong một phép gán duy nhất. Cả chương trình gốc và dạng biểu diễn SSA đều có đồ thị luồng điều khiển giống nhau và có cùng ngữ nghĩa. Sử dụng dạng SSA để biểu diễn luồng dữ liệu của một chương trình tuần tự, ta có thể xây dựng được đồ thị truyền tin ITG để PTKNKT.



a) Chương trình gốc b) Kết quả dưới dạng SSA

Hình 4. Chuyển một hàm sang biểu diễn SSA

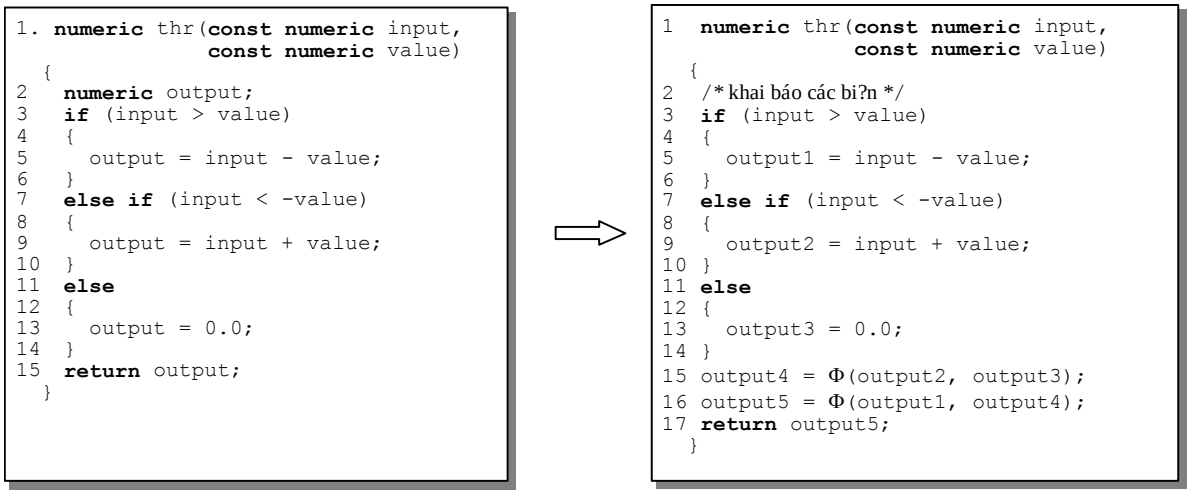
4. Ví dụ áp dụng

Công ty Hàng không châu Âu Thales Avionics đã cho phép chúng tôi sử dụng các biểu đồ CAS gồm các chương trình viết trong ngôn ngữ C. Phương pháp SSA được áp dụng cho các đơn vị THR để PTKNKT, gồm các bước sau:

- Sử dụng trình dịch GCC[1] (GNU Compiler Collection) để chuyển mã nguồn THR thành SSA.
- Xây dựng đồ thị truyền tin ITG từ SSA.
- Tính số lượng các luồng trong ITG.
- Đo KNKT cho mỗi đơn thể trong mỗi luồng.

Đơn vị THR dùng để tính giá trị ngưỡng của một giá trị đầu vào. Mã nguồn của THR sử dụng kiểu số thực (numeric). Hình 5 minh hoạ kết quả chuyển mã THR thành biểu diễn SSA sử dụng GCC.

Từ biểu diễn SSA, đồ thị ITG của THR được xây dựng như sau: Các toán tử và các biến được biểu diễn thành các nút đơn thể. Các toán tử gồm *sub*, *add*, *uminus*, *comp1* và *comp2*, trong đó *comp1* và *comp2* là các toán tử so sánh "$>$" và "$<$" tương ứng. Các câu lệnh điều kiện (if-then-else) được biểu diễn thành hai đơn thể là *then* và *else*. Trong một luồng, chỉ có một trong hai đơn thể điều kiện được chọn.



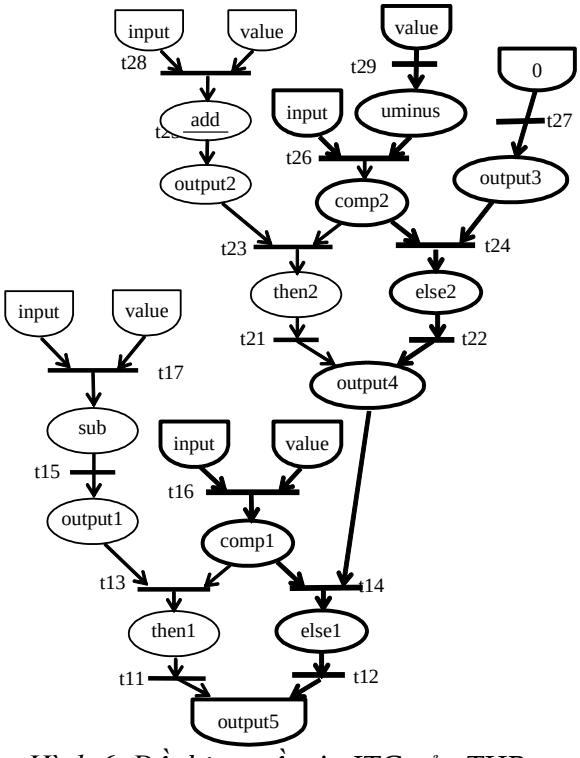
Hình 5. Chuyển mã THR thành biểu diễn SSA sử dụng GCC

Đồ thị có 3 luồng dữ liệu có cùng nút dữ liệu vào {*input, value*} và nút dữ liệu ra {*output5*}. Tập hợp các đơn thể F_i trong mỗi luồng được xác định như sau:

F_1 = {*then1, comp1, output1, sub*}.
F2 = {*else1, comp1, output4, then2, output2, comp2, add, uminus*}.
F3 = {*else1, comp1, output4, else2, comp2, output3, uminus*}.

Mỗi luồng tính giá trị đầu ra {*output5*} từ các giá trị đầu vào {*input, value*}. Chẳng hạn, luồng F_3 được tô đậm trong hình 6.

[1] GCC phiên bản 3.0 hoặc mới hơn.



Hình 6. Đồ thị truyền tin ITG của THR

Khả năng kiểm thử của mỗi đơn thể trong đồ thị ITG được tính theo luồng nhờ công cụ SATAN. Mỗi đơn thể có một cặp giá trị tương ứng với khả năng điều khiển và khả năng quan sát.

Bảng 1. Giá trị khả năng kiểm thử từ đồ thị ITG

luồng/ đơn thể	F_1	F_2	F_3
add			(1.0, 1.0)
comp1	(1.0, 1.0)	(1.0, 1.0)	(1.0, 1.0)
comp2		(1.0, 1.0)	(1.0, 1.0)
else1	(1.0, 1.0)	(0.1538, 1.0)	(1.0, 1.0)
else2		(1.0, 1.0)	
uminus		(1.0, 0.0833)	(1.0, 0.0833)
output1	(1.0, 1.0)		
output2			(1.0, 1.0)
output3		(1.0, 1.0)	
output4		(0.0833, 1.0)	(1.0, 1.0)
sub	(1.0, 1.0)		
then1	(1.0, 1.0)		
then2			(1.0, 1.0)

Từ kết quả cho trong bảng 1, ta xác định được những đơn thể có cặp giá trị khả năng điều khiển và khả năng quan sát thấp. Trong trường hợp này, đơn thể *uminus* luôn có giá trị khả năng quan sát rất thấp trong luồng F_2 và F_3. Điều này dễ hiểu, vì qua luồng F_2 và F_3 có sự mất

mát thông tin từ đầu vào *input* đến đơn thể *comp2*, dữ liệu đầu ra của *uminus* có kiểu số thực, còn dữ liệu đầu ra của đơn thể *comp2* có kiểu *boolean*.

Ta cũng nhận thấy rằng các giá trị KNKT còn cho phép xác định được những luồng hiệu quả hơn để quan sát một đơn thể. Ví dụ, giá trị khả năng điều khiển của *output4* là rất thấp trong luồng F_2, nhưng lại rất cao trong luồng F_3. Luồng F_3 nên được lựa chọn để quan sát đơn thể *output4*.

Phương pháp phân tích đưa ra hai chỉ số:

1) Số luồng để kiểm thử cho phép đánh giá độ phức tạp của đơn vị chương trình cần kiểm thử, bao gồm số lượng dữ liệu kiểm thử cần thiết
2) Các giá trị KNKT chỉ ra sự mất mát thông tin từ dữ liệu đầu vào đến đầu ra, giúp người kiểm thử nhận biết khó khăn phải xử lý khi quan sát và theo dõi kiểm thử. Chẳng hạn, phân tích cho thấy đơn vị THR chỉ có 3 luồng quan sát, mặc dù đơn giản nhưng các giá trị KNKT lại chỉ ra rằng có sự mất mát thông tin đáng kể, cho nên cần phải chú trọng quan sát khi kiểm thử.

Chúng tôi cũng đã phân tích các đơn vị phần mềm khác của Công ty Hàng không Thales Avionics. Kết quả PTKNKT đã giúp các kỹ sư kiểm thử của công ty phát hiện nhiều vấn đề kiểm thử cần giải quyết.

5. Kết luận

Giải pháp PTKNKT dựa trên đồ thị truyền tin ITG sử dụng công cụ SATAN đã được áp dụng cho các biểu đồ thiết kế luồng dữ liệu CAS. Ưu thế của giải pháp là khắc phục tính không tương thích về ngôn ngữ lập trình. Các đơn vị chương trình tham gia kiểm thử được đặc tả trong biểu đồ CAS thường không được viết trong các ngôn ngữ luồng dữ liệu (như Esterel hay Lustre), mà được viết trong các ngôn ngữ mệnh lệnh. Từ đó, chuyển đổi đơn vị kiểm thử sang biểu diễn SSA đã sử dụng được công cụ SATAN để PTKNKT.

Giải pháp của chúng tôi đã mang lại nhiều thông tin hữu ích cho người kiểm thử và người thiết kế phần mềm. Một mặt giúp người kiểm thử định hướng xây dựng tập hợp dữ liệu kiểm thử, mặt khác, giải pháp đã giúp họ phân bổ tài nguyên kiểm thử hợp lí hơn, khắc phục những sự cố mất mát thông tin. Phương pháp đã trình bày ở trên không phụ thuộc vào một phương pháp kiểm thử nào, nhưng đã cung cấp thông tin cần thiết trợ giúp người kiểm thử. Hơn nữa, sử dụng phương pháp này, người xây dựng phần mềm có thể sửa đổi lại chương trình nhằm cải thiện khả năng kiểm thử.

Chúng tôi tiếp tục áp dụng phương pháp này cho mã nguồn của các phần mềm khác nhau về ngôn ngữ lập trình. Chúng tôi cũng nhắm tới khả năng tự động hóa quá trình PTKNKT khi xây dựng đồ thị truyền tin ITG từ biểu diễn SSA.

TÀI LIỆU THAM KHẢO

[1] T. J. McCabe, A Complexity Measure, *IEEE Transactions on Software Engineering*, SE-2(4), 308-320, 1976.

[2] B. A. Nejmeh, NPATH: A Measure of Execution Path Complexity and Its Applications, *Communication of the ACM*, 31(2), 188-200, 1988.

[3] *R. S*. Freedman, Testability of Software Components, *IEEE Transactions on Software Engineering*, 17(6), 553-564, 1991.

[4] J. *M*. Voas and K.W. Miller, Semantic Metrics for Software Testability, *Journal of Systems and Software*, No. 20, 207-216, 1993.

[5] *R*. Petrenko, *R*. Dssouili and H. Koenig, On Evaluation of Testability of Protocol Structures, *Proceedings of the International Workshop on Protocol Test Systems*, Pau, France, 1993.

[6] K. Karoui and *R*. Dssouli, Testablity Analysis of the Communication Protocols Modeled by Relations, *Technical Report*, No. 1050, Département d'Informatique et de Recherche Opérationnelle, Faculté des Arts et des Sciences, Université de Montréal, 1996.

[7] Y. Le Traon, C. Robach, Testability Measurements for Data Flow Designs, *Proceedings of the Fourth International Software Metrics Symposium*, Albuquerque, New Mexico, USA, 91-98, 1997.

[8] Dammak, Etude des Mesures de Testabilité de Systèmes Logiques, Université de Paris-Sud, Centre d'Orsay, Paris, Thesis, 1985.

[9] *R*. Cytron, B. Rosen, *M*. Wegman, and *F*. Zadeck, Efficiently Computing Static Single Assignment Form and the Control Dependence Graph, *ACM Transactions on Programming Languages and Systems*, 13(4), 451-490, 1991.

[10] Nguyễn Thanh Bình, *M*. Delaunay, and C. Robach, Testability Analyis with Respect to Testing Criteria for Software Components, *Proceedings of the 6th ISATED International Conference on Software Engineering and Applications*, Cambridge, USA, 558-563, 2002.

[11] Nguyễn Thanh Bình, *M*. Delaunay, and C. Robach, Testability Analysis of Data-Flow Software, *Proceedings of International Workshop on Test and Analysis of Component Based Systems*, Barcelona, Spain, 193-203, 2004.

[12] Đỗ Huy Vũ, Nguyễn Thanh Bình, *M*. Delaunay, C. Robach, Analyse de la Testabilité des Logiciels Flots de Données Synchrones, Acts de la Première Conférence Internationale Associant Chercheurs Vietnamiens et Francophones, Hanoi, Vietnam, 223-232, 2003.