Tạp chí Tin học và Điều khiển học, T. 15, S. 3 (1999), 40–45

CÂY CHỈ THỊ NHỊ PHÂN BIỂU DIỄN KHÔNG GIAN TRẠNG THÁI CHẨN ĐOÁN VI RÚT TIN HỌC

NGUYỄN THANH THỦY, TRƯƠNG MINH NHẬT QUANG

Abstract. Applying problem solving methods and knowledge processing techniques in Artificial Intelligence, we have designed a searching space which helps the inference motor to identify new virus in the form of binary instruction tree. This database structure specifies execution processing of the central processing unit 80×86. It's made to be suitable with installing the searching algorithm. A knowledge based algorithm is combined with some heuristic to speed up the process of searching virus in the tree and increase the confidence of the inference motor.

1. CHẨN ĐOÁN VI RUS VÀ CÁC PHƯƠNG PHÁP GIẢI QUYẾT VẤN ĐỀ

"Về một phương diện nào đó, khả năng trí tuệ chính là khả năng học tập và khả năng giải quyết vấn đề" (W. Beranrd).

Một trong những khó khăn lớn nhất của việc chẩn đoán virus là cách tiếp cận các phương pháp giải quyết vấn đề (GQVĐ). Cần lựa chọn phương pháp GQVĐ nào, tổ chức cơ sở tri thức với không gian biểu diễn ra sao, xây dựng thuật giải tương ứng... Các yếu tố này rất quan trọng, nếu không nói là có tính chiến lược của AntiVirus. Nó quyết định tính khả thi, độ tin cậy của quá trình chẩn đoán và ảnh hưởng đến tính bền vững của cả hệ. Hệ chẩn đoán mà chúng ta xây dựng phải tái hiện một cách trung thực các qui trình công nghệ chẩn đoán virus của các chuyên gia trong thế giới thực. Nói cách khác, hệ phải có khả năng GQVĐ trên cơ sở lý thuyết của trí tuệ nhân tạo.

Để giải quyết vấn đề, điều trước tiên là cần biểu diễn vấn đề theo các yêu cầu [4]:

- Sự hiểu vấn đề thống nhất giữa người đặt vấn đề và người GQVĐ.
- Sự nhất trí về hình thức biểu đạt cách GQVĐ.
- Xác định tiền đề (cơ sở dữ liệu ban đầu) cho cách giải.
- Xác định tiêu chuẩn đánh giá chất lượng cách giải.

Qua phân tích các yêu cầu của giai đoạn biểu diễn vấn đề nhằm cố gắng tìm một phương pháp tiếp cận khả dĩ cho vấn đề chẩn đoán virus, chúng ta thấy rằng có thể phát biểu vấn đề này dưới dạng "từ một trạng thái xuất phát hãy tìm đường dẫn đến một trạng thái kết thúc mong muốn". Quá trình GQVĐ bao gồm:

- Chọn không gian tìm kiếm thích hợp.
- Tiến hành tìm kiếm có hệ thống và hiệu quả trong không gian tìm kiếm.
- Sử dụng triệt để các nguồn tri thức liên quan trong quá trình tìm kiếm tương ứng với miền đối tượng cụ thể.

Không gian tìm kiếm của một vấn đề giải trên máy tính thường được biểu diễn bởi đồ thị, đôi khi có thể một dạng đặc biệt của đồ thị là cây, trong đó:

- Mỗi đỉnh là một giai đoạn của quá trình giải (trạng thái).
- Mỗi cung là tác động biến đổi quá trình từ giai đoạn này sang giai đoạn khác.

Việc GQVĐ rút cục lại là tìm đường đi từ trạng thái ban đầu tới trạng thái mong muốn được biểu diễn qua hai đỉnh nào đó của đồ thị hoặc cây tìm kiếm (vì vậy trong một số trường hợp, *không gian tìm kiếm* cũng được gọi là *không gian trạng thái*). Vấn đề còn lại là xây dựng các thủ tục tìm kiếm thích hợp trên không gian trạng thái một cách tối ưu, hiệu quả và nhanh nhất.

2. TỔ CHỨC KHÔNG GIAN TRẠNG THÁI CHO MÔ-TƠ SUY DIỄN

Tất cả các chương trình thi hành trên máy PC dùng bộ xử lý 8088, 80×86 (gọi tắt là VXL 80×86) dù được viết bằng ngôn ngữ nào, cũng được trình biên dịch của nó dịch thành các tập tin thực thi chứa các chỉ thị máy của bộ xử lý. Lợi dụng đặc điểm này, người viết virus sử dụng các trình biên dịch Assembly để thiết kế virus và tìm cách đính nó vào các chương trình thực thi. Vì thế, chúng luôn "hòa hợp" với các chương trình thực thi khác mà không phụ thuộc vào ngôn ngữ ban đầu của chủ thể. Như vậy, để phát hiện một hành vi (bao gồm nhiều chỉ thị theo một cấu trúc nào đó), mô tơ phải thao tác trên tập hợp E các mã lệnh của bộ xử lý. Tập hợp này được tổ chức như mảng một chiều trên bộ nhớ của máy tính (RAM, đĩa từ). Cách lưu trữ đơn giản này không phù hợp với các phương pháp tìm kiếm đã nêu. Bản thân cấu trúc dữ liệu mảng đã bộc lộ quan điểm xem các phần tử trên E như là một tập dữ liệu đơn thuần (các AntiVirus cổ điển cũng quan niệm như vậy). Quan điểm này chỉ đúng đối với trường hợp tập E đang "ngủ yên" trên thiết bị lưu trữ, nhưng khi chúng được tải vào vùng nhớ và nhận được quyền thi hành, thì chúng không còn là một chuỗi các byte vô nghĩa như trước mà trở thành một chương trình linh hoạt, hoạt động theo kế hoạch đã lập sẵn, thực hiện các *hành vi* nội tại... Như vậy, nếu quan niệm E là một tập các *hành vi có cấu trúc* thì vấn đề sẽ sáng tỏ hơn. Điều này có thể giải quyết bằng việc tìm hiểu cơ chế thực hiện lệnh VXL 80×86. Chúng ta sẽ không đi sâu vào phân tích chi tiết các kiến trúc vật lý của họ máy PC mà chỉ cần tìm hiểu cơ chế thực hiện lệnh với mục đích tái hiện tiến trình xử lý dưới dạng giải thuật.

2.1. Các thanh ghi của VXL 80×86 [1]

VXL 80×86 được thiết kế để thực hiện và tiến hành các phép toán số học và luận lý cùng lúc với việc nó nhận lệnh và chuyển dữ liệu vào/ra bộ nhớ. Để làm điều này, 80×86 dùng các thanh ghi 16-bit:

• Bốn thanh ghi chung AX, BX, CX, DX, được dùng tùy theo công dụng riêng:

- AX (Accumulator): được dùng để tích lũy, thực hiện các phép tính số học.
- BX (Base): được dùng chứa địa chỉ con trỏ theo bảng, hoặc phần địa chỉ offset của một địa chỉ theo đoạn.
- CX (Count): được dùng làm bộ đếm để điều khiển vòng lặp, các lệnh chuyển dời dữ liệu..
- DX (Data): được dùng để cất dữ liệu 16 bit cho các mục đích chung.

Mỗi thanh ghi này có thể được chia làm hai thanh ghi 8-bit riêng biệt, đó là AH, AL, BH, BL, CH, CL, DH, DL.

• Bốn thanh ghi đoạn CS, DS, ES, SS dùng để đánh các địa chỉ theo đoạn 64 KB của bộ nhớ:

- CS (Code Segment): xác định đoạn lệnh nơi chương trình đang thi hành.
- DS (Data Segment): xác định đoạn dữ liệu của chương trình.
- ES (Extra Segment): xác định đoạn thêm, bổ sung cho đoạn dữ liệu.
- SS (Stack Segment): xác định đoạn ngăn xếp, dùng theo dõi các tham số và các địa chỉ đang được chương trình hiện hành sử dụng.

• Năm thanh ghi offset IP, SP, BP, SI, DI dùng định chính xác một byte/word trong một đoạn cụ thể:

- IP (Instruction Pointer): giữ vị trí của lệnh hiện hành trong đoạn lệnh, dùng chung với CS tạo thành một địa chỉ 20 bit dạng CS:IP.
- SP (Stack Ponter): con trỏ ngăn xếp, giữ vị trí hiện hành của đỉnh ngăn xếp.
- BP (Basic Pointer): dùng truy nhập vào ngăn xếp, định các tham số...
- SI (Source Index): con trỏ định vị địa chỉ nguồn.
- DI (Destination Inedx): con trỏ định vị địa chỉ đích.

• Thanh ghi cờ chứa 9 bit điều khiển biểu diễn thông tin trạng thái của bộ VXL, bao gồm 6 cờ trạng thái:

- CF (Carry Flag): cờ nhớ, dùng cho các phép toán số học có nhớ.
- OF (Overtflow Flag): cờ tràn, dùng cho các phép toán số học bị tràn.
- ZF (Zero Flag): cờ zero, báo cáo kết quả bằng 0, so sánh bằng.
- SF (Sign Flag): cờ dấu, báo cáo kết quả, so sánh âm.
- PF (Parity Flag): cờ chẵn lẻ, số chẵn các bit 1.
- AF (Auxiliary): cờ phụ.

Thanh ghi này còn chứa ba cờ điều khiển:

- DF (Direction Flag): cờ định hướng tiến lùi.
- IF (Interrup Flag): cờ ngắt.
- TF (Trap Flag): cờ bẫy.

2.2. cơ chế thực hiện lệnh của VXL 80 × 86 [1]

Quá trình thực hiện lệnh của VXL 80 × 86 không giới hạn trong phạm vi 64 KB của đoạn lệnh do CS quản lý mà có thể trải dài trên vùng nhớ 20 bit địa chỉ. Chúng ta hãy xem xét giải thuật thực hiện các chỉ thị của VXL 80×86 như sau:

Bước 1 – Khởi tạo giá trị ban đầu cho các thanh ghi.
– CS:IP giữ địa chỉ ban đầu của điểm vào lệnh đầu tiên.

Bước 2 – Thực hiện lệnh hiện tại, ghi nhận trạng thái vào thanh ghi cờ.
– Tính độ dời lệnh trong từng trường hợp. Nếu lệnh hiện tại là:

+ Lệnh gọi, bao gồm:

. CALL ⟨ thủ tục ⟩:

- Tăng IP 3 byte (IP trỏ đến lệnh kế, sau lệnh CALL).
- Lưu địa chỉ trở về: đẩy IP vào địa chỉ SS:SP, giảm SP 2 byte.
- Giảm SP 2 byte.
- Tính toán địa chỉ offset Thủ tục, gán cho IP.
- Đến bước 2.

. CALL FAR ⟨Thủ tục⟩:

- Tăng IP 3 byte (IP trỏ đến lệnh kế, sau lệnh CALL).
- Lưu địa chỉ trở về: đẩy CS,IP vào địa chỉ SS:SP, giảm SP 4 byte.
- Tính toán địa chỉ thủ tục (dạng CS:IP).
- Định vị lại CS:IP mới.
- Đến bước 2.

. INT ⟨X⟩

- Tăng IP 2 byte (IP trỏ đến lệnh kế, sau lệnh INT)
- Đẩy thanh ghi cờ vào SS: SP, giảm SP 2 byte
- Lưu địa chỉ trở về: đẩy CS,IP vào địa chỉ SS:SP, giảm SP 4 byte.
- Tính toán địa chỉ ngắt X (dạng CS:IP) từ bảng vector ngắt.
- Định vị lại CS:IP mới.
- Đến bước 2.

+ Lệnh trở về:

. RET:

- Lấy giá trị 1 word tại SS:SP gán cho IP.
- Tăng SP 2 byte.
- Đến bước 2.

. RETF:
- Lấy giá trị 2 word tại SS:SP gán cho CS:IP.
- Tăng SP 4 byte.
- Đến bước 2.

. IRET:
- Lấy giá trị 1 word tại SS:SP vào thanh ghi cờ.
- Tăng SP 2 byte.
- Lấy giá trị 2 word tại SS:SP gán cho CS:IP.
- Tăng SP 4 byte.
- Đến bước 2.

+ Lệnh nhảy không điều kiện:
. Nhảy gần: định hướng nhảy, độ rời lệnh.
. Nhảy xa: * Định lại CS:IP mới.
* Đến bước 2.

+ Lệnh tuần tự: tính độ dài lệnh hiện tại.

+ Lệnh nhảy có điều kiện: dạư vào trạng thái thanh ghi cờ (điều kiện nhảy), định hướng nhảy, độ dời lệnh.

Bước 3 – Cộng giá trị dời lệnh vào IP, (do đó IP sẽ trỏ đện địa chỉ lệnh kế tiếp),
– Đến bước 2.

Trên cơ sở của giải thuật này chúng ta có thể xem quá trình thực hiện một chương trình trên máy PC sử dụng VXL 80 × 86 chỉ chứa hai lệnh cơ bản:

i) Các lệnh tuần tự:
Phép tính IP mới sau các lệnh này dựa theo công thức:

$$IP = IP + \text{Độ dài lệnh.} \quad (1)$$

Đại lượng *Độ dài lệnh* là một hằng số được qui định bởi VXL 80×86.

Ví dụ: + Các lệnh thay đổi giá trị thanh ghi, biến...
+ Các phép toán số học, luận lý.
+ Các lệnh gọi chương trình con.
+ ...

ii) Các lệnh rẽ nhánh:
Phép tính IP mới sau này phụ thuộc dựa theo công thức:

$$IP = IP + \text{Độ dời lệnh} \quad (2)$$

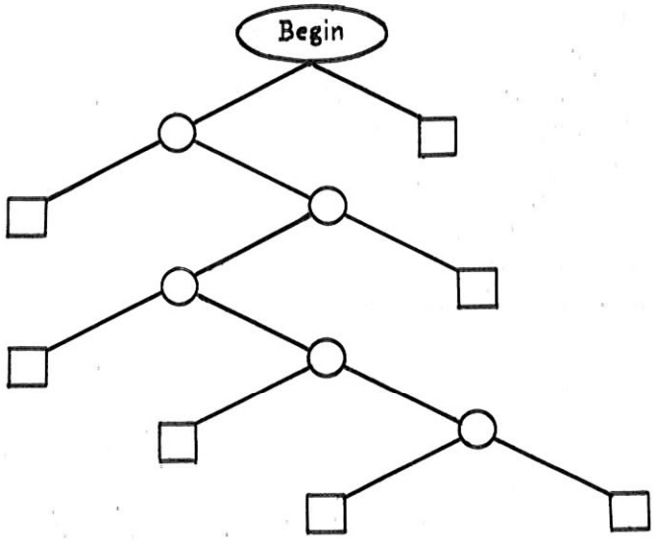
Đại lượng *Độ dời lệnh* là một biến số được qui định bởi cơ chế tính toán địa chỉ và sinh mã lệnh của trình biên dịch. Giá trị của nó sẽ được tính toán lúc HĐH nạp mã lệnh vào bộ nhớ (đối với tập EXE) hoặc lúc chạy chương trình E.

Ví dụ: + Các lệnh nhảy có điều kiện, không điều kiện, gần hoặc xa.
+ Các vòng lặp.
+ Một số thủ thuật thao tác ngăn xếp cho kết quả tương tự lệnh nhảy.

Lưu ý rằng với cách phân loại trên, các lệnh nhảy sẽ rẽ hướng chương trình theo *một trong hai* hướng thích hợp tùy theo điều kiện, trạng thái của thanh ghi cờ. Lúc này địa chỉ nhảy đến được tính toán và "mạch" của chương trình sẽ thay đổi.

Lệnh CALL, INT được xem là lệnh tuần tự, vì nó tuân theo công thức (1) và không làm thay đổi mạch của chương trình. Ngược lại các lệnh vòng lặp không được xem là tuần tự vì bản thân chúng thường chứa các lệnh nhảy có điều kiện, hoặc các lệnh điều khiển rẽ nhánh mặc định (dạng

LOOP if CX <> 0). Vì vậy, một cách hình thức có thể biểu diễn mô hình xử lý lệnh của VXL 80×86 như một cây *nhị phân*.



Cây chỉ thị nhị phân của chương trình E

Trong đó:

- Gốc biểu diễn điểm vào lệnh đầu tiên.
- Các nhánh biểu diễn các lệnh tuần tự.
- Các nút biểu diễn một lệnh nhảy.
- Nút lá đánh dấu điểm kết thúc của chương trình.

Với cách biểu diễn này, môtơ suy diễn sẽ dễ dàng áp dụng các chiến lược tìm kiếm trên không gian trạng thái đã được đồ thị hóa dưới dạng *cây chỉ thị nhị phân*. Như vậy, vấn đề tổ chức không gian trạng thái đã được giải quyết. Vấn đề thứ hai là *xây dựng cơ chế tìm kiếm* trên không gian này.

3. XÂY DỰNG CƠ CHẾ TÌM KIẾM TRÊN CÂY CHỈ THỊ NHỊ PHÂN

Các thủ tục tìm kiếm điển hình bao gồm [5]:

① Tìm kiếm rộng: đường đi được tìm theo mọi hướng có thể ở mỗi bước.

② Tìm kiếm sâu: đường đi sâu mãi theo một hướng đến khi nào không tiếp tục đi nữa mới chuyển sang hướng khác.

③ Tìm kiếm sâu dần: tìm kiếm sâu ở mức n cho trước rồi tìm kiếm rộng ứng với mức đó.

④ Tìm kiếm cực tiểu hóa giá thành: mỗi cung của cây/đồ thị được gán giá thành, hướng tìm kiếm được xác định bởi việc cực tiểu hóa giá thành đường đi.

⑤ Tìm kiếm với tri thức bổ sung: hướng tìm kiếm được xác định với tri thức bổ sung ở mỗi bước.

Đối với chúng ta, phương pháp tìm kiếm nào tỏ ra hiệu quả nhất trên cây chỉ thị nhị phân đã xây dựng? Các phương pháp ①, ②, ③ cho phép vét cạn cây tìm kiếm nhưng tốc độ khá chậm, nhất là đối với trường hợp tập E đặc tả các file EXE dài hàng MByte. Vả lại, việc nhận dạng các hành vi cơ bản của virus trên E cũng không cần thiết phải vét cạn không gian tìm kiếm. Phương pháp ④ đòi hỏi phải định giá thành cho mỗi nhánh trên cây, nhưng ta lại không có một cơ sở nào để ước

lượng được khả năng xuất hiện các hành vi của virus trên một nhánh nào đó. Nếu cố định chọn giải pháp này, chúng ta phải xây dựng thêm một bộ nhận dạng *tiền hành vi* cho mỗi nhánh trước khi nhận dạng hành vi trên nhánh đó, kết quả chắc chắn sẽ suy biến đến vô cực!.

Phương pháp ⑤ (tìm kiếm với *tri thức bổ sung* tại mỗi nút) có thể được áp dụng để giải quyết vấn đề. Nhưng lượng *tri thức bổ sung* được cập nhật từ đâu và bổ sung như thế nào trong quá trình tìm kiếm? Để giải quyết vấn đề này, chúng ta sử dụng giải pháp heuristic như sau:

Khi thực hiện tìm kiếm đến một nút (đặc tả lệnh nhảy) trên cây, vấn đề mấu chốt là *phải biết nhảy theo hướng nào*. Như trên đã phân tích, trước khi thực hiện lệnh nhảy, VXL 80 × 86 sẽ kiểm tra trạng thái thanh ghi cờ để quyết định hướng nhảy. Giá trị của thanh ghi cờ tại mỗi nút chính là điều kiện nhảy, như vậy chúng sẽ lợi dụng cơ chế này để nhận được *tri thức bổ sung* tại từng nút cho cây nhị phân của mình. Đến đây thì sự việc đã quá rõ ràng, chúng ta chỉ cần xây dựng một cơ chế tìm kiếm dựa theo giải thuật thi hành lệnh của VXL 80 × 86. Để thu thập tri thức bổ sung, hàm Flag(n) có nhiệm vụ phân tích trạng thái biến nhớ 1 word đặc tả thanh ghi cờ. Hàm này sẽ được gọi định kỳ khi quá trình tìm kiếm đạt đến nút n nào đó, kết quả trả về của hàm sẽ hỗ trợ tri thức cho việc quyết định rẽ hướng trên cây.

Như vậy, để giải quyết bài toán nhận dạng virus, chúng ta sẽ xây dựng một không gian trạng thái với cây chỉ thị nhị phân cho V, các nút lá của cây V chính là điểm dừng khi môtơ đã đạt đến *số nút tới hạn*, hoặc đạt được kết quả nhờ quá trình tổng hợp hành vi (được ghi nhận từ quá trình tìm kiếm ở các nút cha). Áp dụng phương pháp tìm kiếm với tri thức bổ sung trên cây để hạn chế miền tìm kiếm. Trong một số trường hợp, các giải thuật quay lui, heuristics sẽ được ứng dụng. Ví dụ nếu không gian trạng thái không lớn, có thể gia tăng giá trị *số nút tới hạn* để mở rộng miền tìm kiếm. Như vậy môtơ suy diễn của chúng ta phải có khả năng nhận dạng từng chỉ thị, độ dài chỉ thị (để tính địa chỉ, nội dung của chỉ thị kế), biết cách tính địa chỉ tương đối của các lệnh nhảy (để xác định các nút), và nhất là biết khai thác tri thức bổ sung từ kết quả của hàm Flag(n) cung cấp cho nó.

TÀI LIỆU THAM KHẢO

[1] Peter Norton, *PC programmer handbook*, Bản dịch của Nguyễn Minh Sơn và Đỗ Phúc, NXB Đại học và Giáo dục chuyên nghiệp, 1992.

[2] Arne Shaper, *Tubo Pascal 5.5: Concepts, Analyses, Tip and Tricks*, Eddison-Wesley Publishing Co. Inc 1990, Bản dịch của Trương Văn Chú, Văn Đại Hưng, NXB Khoa học và Kỹ thuật, 1997.

[3] Niklau Wirth, *Program = Data Structure + Algorithm*, Bản dịch của Hồ Thuần, NXB Thống kê, 1982.

[4] Bạch Hưng Khang, Hoàng Kiếm, *Trí tuệ nhân tạo: Các phương pháp và ứng dụng*, NXB Khoa học và Kỹ thuật, 1989.

[5] Nguyễn Thanh Thủy, *Trí tuệ nhân tạo: Các phương pháp giải quyết vấn đề và xử lý tri thức*, NXB Giáo dục, 1995.

[6] Nguyễn Thanh Thủy, Trương Minh Nhật Quang, Các giải pháp cho phần mềm chống virus thông minh, *Tạp chí Tin học và Điều khiển học* **13** (3) (1997) 123-132.

[7] Nguyễn Thanh Thủy, Trương Minh Nhật Quang, Các cơ chế chẩn đoán virus tin học thông minh dựa trên tri thức, *Tạp chí Tin học và Điều khiển học* **12** (2) (1996) 42-52.

Nhận bài ngày 8 - 11 - 1997

Nguyễn Thanh Thủy, Đại học Bách khoa Hà Nội.

Trương Minh Nhật Quang, Viện Tin học tiếng Pháp.