**SP800-192: PHƯƠNG PHÁP THỬ NGHIỆM, XÁC MINH CHO CÁC MÔ HÌNH, CHÍNH SÁCH ĐIỀU KHIỂN TRUY CẬP**

# Tổng quan

## Phạm vi và mục đích của tài liệu

Mục đích của tài liệu này là cung cấp cho các cơ quan thông tin kỹ thuật để hỗ trợ họ thiết kế, xác minh và thử nghiệm các mô hình điều khiển truy cập (AC). Tài liệu này thảo luận về các nguyên tắc, yêu cầu, lý thuyết và sơ đồ xác minh và thử nghiệm các mô hình AC được thiết kế để đảm bảo chính sách AC có thể được thực hiện theo cơ chế.

Chính sách AC được chỉ định để điều khiển quyền truy cập tài nguyên hệ thống, cơ chế AC điều khiển người dùng hoặc quy trình nào có quyền truy cập vào tài nguyên nào trong hệ thống. Tuy nhiên, việc thực thi chính xác các chính sách AC bằng các cơ chế AC là những vấn đề rất thách thức. Do đó, các mô hình AC thường được viết để thu hẹp khoảng cách trừu tượng giữa các chính sách và cơ chế để nắm bắt chính thức và chính xác các yêu cầu an toàn mà các hệ thống AC cần tuân thủ.

Trong tài liệu này, xem xét các phương pháp xác minh cho các mô hình AC và thử nghiệm triển khai mô hình bằng cách xác định trước các cấu trúc tiêu chuẩn cho các mô hình AC. Sau đó, trình bày các biểu thức của mô hình AC và các yêu cầu an toàn bằng ngôn ngữ đặc tả của trình kiểm tra mô hình để sử dụng trình kiểm tra mô hình white-box và black-box để xác minh tính toàn vẹn, phạm vi và giới hạn của các yêu cầu an toàn được chỉ định đối với các mô hình.

## Đối tượng

Tài liệu này nhằm cung cấp hướng dẫn thực tế và khái niệm cho người quản lý bảo mật, quản trị viên và người kiểm tra hệ thống có chuyên môn liên quan đến AC.

## Tổ chức tài liệu

Tài liệu này được chia thành bảy phần và hai phụ lục.

* Phần 1 nêu rõ phạm vi, mục đích, đối tượng và giả định của tài liệu này.
* Phần 2 giới thiệu khái niệm chung về chính sách và mô hình AC.
* Phần 3 giải thích các yếu tố an toàn và lỗi của AC.
* Phần 4 trình bày các nguyên tắc để xác minh mô hình AC.
* Phần 5 cung cấp một số phương pháp chính được sử dụng để thử nghiệm và xác minh mô hình AC
* Phần 6: Trình bày cách triển khai công cụ thử nghiệm và xác minh AC của NIST: NIST’s Access Control Policy Tool (ACPT)
* Phần 7: Trình bày kết luận cho tài liệu.

# Chính sách và mô hình điều khiển truy cập

Mục tiêu của hệ thống điều khiển truy cập (AC) là vệ tài nguyên hệ thống trước sự truy cập của người dùng không phù hợp hoặc không mong muốn. Từ góc độ triển khai, mục tiêu này cũng có thể được mô tả theo cách chia sẻ thông tin tối ưu cho người dùng và ứng dụng. Tuy nhiên, mức độ chia sẻ lớn hơn có thể cản trở việc bảo vệ tài nguyên, do đó, chính sách AC đủ chi tiết sẽ cho phép chia sẻ thông tin có chọn lọc trong trường hợp không có, việc chia sẻ có thể được coi là quá rủi ro.

Việc thực thi chính sách AC chính xác dựa trên tiền đề rằng các đặc tả chính sách là chính xác mà không có các quy tắc ẩn hoặc xung đột gây rò rỉ hoặc chặn truy cập vào các đối tượng. Thông số kỹ thuật chính sách AC phải trải qua xác minh và xác thực nghiêm ngặt thông qua thử nghiệm có hệ thống để đảm bảo rằng các thông số kỹ thuật chính sách thực sự gói gọn những mong muốn của người lập chính sách. Kiểm thử và xác minh các chính sách AC giống như kiểm thử phần mềm ứng dụng chung, nhưng cũng có những khác biệt. Để nắm bắt chính xác các yêu cầu an toàn mà hệ thống AC cần tuân thủ, các mô hình thường được viết để thu hẹp khoảng cách khá rộng về sự trừu tượng giữa chính sách và cơ chế. Do đó, một mô hình AC cung cấp biểu thức rõ ràng và chính xác cũng như tham chiếu cho thiết kế và thực hiện các yêu cầu an toàn.

## Chính sách điều khiển truy cập

Chính sách AC là các yêu cầu chỉ định cách quản lý quyền truy cập. Ai, trong hoàn cảnh nào, có thể truy cập thông tin nào. Mặc dù các chính sách AC có thể dành riêng cho ứng dụng và do nhà cung cấp ứng dụng xem xét, các chính sách cũng có khả năng liên quan đến hành động của người dùng trong bối cảnh của một đơn vị tổ chức hoặc qua các ranh giới tổ chức.

Việc lập danh sách các chính sách AC chung là không thực tế, vì mục tiêu kinh doanh, chấp nhận rủi ro, văn hóa doanh nghiệp và trách nhiệm pháp lý ảnh hưởng đến chính sách khác nhau giữa doanh nghiệp với doanh nghiệp và thậm chí từ đơn vị tổ chức này sang đơn vị tổ chức khác. Hơn nữa, các chính sách AC có bản chất động, theo đó chúng có thể thay đổi theo thời gian để phản ánh các yếu tố kinh doanh, quy định của chính phủ và điều kiện môi trường. Các chính sách AC có thể được phân loại thành tùy quyền hoặc không tùy quyền.

* **Điều khiển truy cập tùy quyền (Discretionary Access Control)**

AC tùy quyền (DAC) định nghĩa những chính sách điều khiển truy cập cơ bản đối với các đối tượng trong một hệ thống tập tin (*filesystem*) của chủ sở hữu đối tượng hoặc bất kỳ ai khác được ủy quyền điều khiển quyền truy cập của đối tượng.

Ví dụ, DAC thường được sử dụng để giới hạn quyền truy cập của người dùng vào một tệp; chủ sở hữu tệp điều khiển quyền truy cập của người dùng khác vào tệp. Chỉ những người dùng được chỉ định bởi chủ sở hữu mới có thể có một số quyền kết hợp đọc, ghi, thực thi, v.v. đối với tệp. Chính sách của DAC có xu hướng rất linh hoạt và được sử dụng rộng rãi trong các lĩnh vực thương mại và chính phủ.

Tuy nhiên, DAC được biết là vốn yếu vì hai lý do.

* Đầu tiên, cấp quyền truy cập đọc là bắc cầu; ví dụ: khi Ann cấp cho Bob quyền truy cập vào một tệp, không có gì ngăn Bob sao chép nội dung của tệp Ann vào một đối tượng mà Bob điều khiển. Bây giờ Bob có thể cấp cho bất kỳ người dùng nào khác quyền truy cập vào bản sao của tệp Ann mà Ann không biết.
* Thứ hai, chính sách của DAC dễ bị tấn công bởi Trojan. Vì các chương trình kế thừa danh tính của người dùng yêu cầu, chẳng hạn, Bob có thể viết một chương trình cho Ann, trên bề mặt, thực hiện một số chức năng hữu ích, đồng thời phá hủy nội dung của các tệp Ann.

Do đó, về mặt hình thức, các nhược điểm của DAC như sau:

* Thông tin có thể được sao chép từ đối tượng này sang đối tượng khác; do đó, không có sự đảm bảo thực sự về luồng thông tin trong một hệ thống.
* Không hạn chế áp dụng cho việc sử dụng thông tin khi người dùng đã nhận được.
* Các đặc quyền để truy cập các đối tượng được quyết định bởi chủ sở hữu của đối tượng, thay vì thông qua chính sách toàn hệ thống phản ánh các yêu cầu an toàn của tổ chức.

ACL và chủ sở hữu / nhóm / cơ chế AC khác cho đến nay là các cơ chế phổ biến nhất để thực hiện các chính sách DAC. Các cơ chế khác, mặc dù không được thiết kế với DAC, có thể có khả năng thực hiện chính sách DAC.

* **Điều khiển truy cập không tùy quyền**

Nói chung, tất cả các chính sách AC khác với DAC được nhóm thuộc danh mục AC không tùy quyền (NDAC). Như tên của nó, các chính sách trong danh mục này có các quy tắc không được thiết lập theo quyết định của người dùng. Chính sách không tùy quyền thiết lập các điều khiển mà người dùng không thể thay đổi mà chỉ thông qua hành động quản trị.

Ví dụ, NDAC tĩnh là multi-level security( MLS), ABAC và RBAC và NDAC động như chính sách phân chia nhiệm vụ (SoD) có thể được sử dụng để thực thi các ràng buộc đối với việc gán người dùng cho vai trò hoặc nhiệm vụ. Một ví dụ về một ràng buộc là yêu cầu hai vai trò loại trừ lẫn nhau.

Ví dụ: nếu một vai trò yêu cầu chi tiêu và một vai trò khác chấp thuận chúng, tổ chức có thể cấm cùng một người dùng được gán cho cả hai vai trò. Vì vậy, tư cách thành viên trong một vai trò có thể ngăn người dùng trở thành thành viên của một hoặc nhiều vai trò khác, tùy thuộc vào quy tắc SOD, chẳng hạn như Work Flow và RBAC. Một ví dụ khác về NDAC là chính sách SOD dựa trên lịch sử quy định. ví dụ: liệu cùng một vai trò có thể truy cập cùng một đối tượng cho một số lần thay đổi hay không.

Do thực tế là các chính sách của DAC không có các biến được xác định rõ về định nghĩa mô hình, nên nhìn chung chúng khó chính thức hóa và xác minh hiệu quả với các mô hình đơn hàng đầu tiên, mà hầu hết các công cụ xác minh White-box và Back-box đều dựa trên. Tài liệu này không bao gồm các chính sách của DAC.

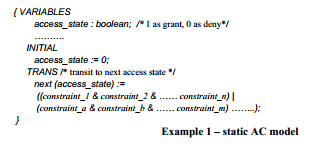
## Mô hình AC

Thay vì cố gắng đánh giá và phân tích các chính sách AC duy nhất ở cấp độ cơ chế, các mô hình bảo mật thường được viết để mô tả các thuộc tính bảo mật của chính sách AC. Mô hình AC là bản trình bày chính thức về chính sách AC được thi hành bởi cơ chế và rất hữu ích trong việc chứng minh các giới hạn lý thuyết của hệ thống để các cơ chế AC có thể được thiết kế để tuân thủ các thuộc tính của mô hình. Người dùng xem mô hình AC là một biểu thức rõ ràng và chính xác của các yêu cầu. Các nhà cung cấp và nhà phát triển hệ thống xem các mô hình AC là đặc điểm kỹ thuật của các yêu cầu thiết kế và triển khai. Một mô hình AC có thể cứng nhắc trong việc thực hiện một chính sách duy nhất. Mặt khác, một mô hình AC sẽ cho phép thể hiện rất nhiều chính sách và các lớp chính sách. Nói chung, các chính sách AC không tùy quyền có thể được mô hình hóa bằng các mô hình Finite State Machine (máy trạng thái đơn định) (FSM) từ một trong ba lớp: **Static, Dynamic, Historical**

### Lớp chính sách Static

Các chính sách trong Lớp chính sách Static quy định quyền truy cập theo trạng thái hoặc điều kiện hệ thống tĩnh như quy tắc, thuộc tính và môi trường hệ thống (thời gian và vị trí để truy cập). Các chính sách AC phổ biến với các loại thuộc tính này bao gồm MLS và RBAC [NIST-IR-7316]. Các loại chính sách này có thể được chỉ định bởi các biểu thức đặc tả không đồng bộ (**asynchronous** ) hoặc trực tiếp (direct) của mô hình FSM. Mối quan hệ chuyển tiếp của các trạng thái ủy quyền được chỉ định trực tiếp như một công thức mệnh đề theo các giá trị hiện tại và tiếp theo của các biến trạng thái. Bất kỳ cặp trạng thái hiện tại / trạng thái tiếp theo nào đều nằm trong mối quan hệ chuyển tiếp khi và chỉ khi nó thỏa mãn công thức

Ví dụ 1

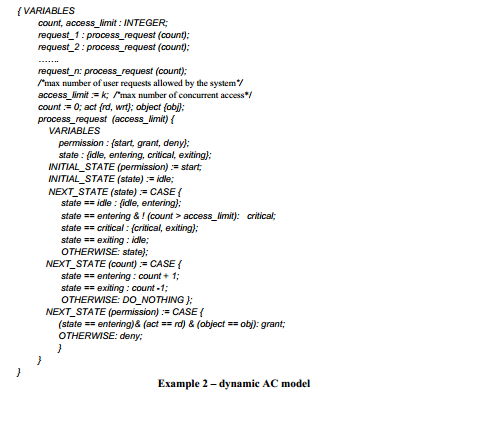


Trạng thái ủy quyền truy cập hệ thống được khởi tạo là trạng thái **deny** và được chuyển sang trạng thái **grant** cho bất kỳ yêu cầu truy cập nào tuân thủ các ràng buộc của quy tắc tương ứng với từng vị từ ràng buộc (ví dụ: *constraint* *\_1*… & *constraint\_n*) trong quy tắc và vẫn deny nếu không tuân thủ yêu cầu nào.

### Lớp chính sách Dynamic

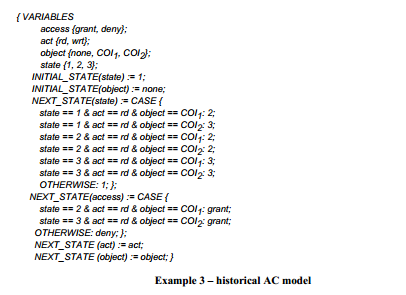
Các chính sách trong Lớp chính sách Dynamic có thể bao gồm các ràng buộc tạm thời điều chỉnh quyền truy cập theo các trạng thái hoặc điều kiện hệ thống động như các sự kiện được chỉ rõ hoặc bộ đếm hệ thống hoặc chính sách AC của N-person. Một mô hình AC với các loại thuộc tính này chỉ định rằng các truy cập chỉ được cho phép bởi một đối tượng nhất định đối với một đối tượng nhất định với các giới hạn nhất định (ví dụ: đối tượng x có thể truy cập không quá i lần cùng lúc bởi nhóm người dùng y).

Ví dụ: nếu vai trò của người dùng là một nhân viên thu ngân, anh ta không thể làm kế toán cùng một lúc khi xử lý hóa đơn của khách hàng. Loại chính sách này có thể được chỉ định bằng các biểu thức đặc tả không đồng bộ ( **asynchronous )** hoặc trực tiếp ( **direct specification )** của mô hình FSM, sử dụng một biến tín hiệu để thể hiện các thuộc tính động của quy trình quyết định ủy quyền. Một ví dụ khác về các trạng thái ràng buộc động là thực thi một số lượng hạn chế các truy cập đồng thời vào một đối tượng. Do đó, quy trình ủy quyền cho một người dùng có bốn trạng thái: ***idle, entering, critical***, ***exiting***. Một người dùng thường ở trạng thái ***idle***. Người dùng được chuyển sang trạng thái ***entering*** khi người dùng muốn truy cập vào đối tượng quan trọng. Nếu không chuyển trạng thái được do số lần truy cập bị giới hạn, người dùng sẽ được chuyển sang trạng thái ***critical*** và số lượng truy cập hiện tại được tăng thêm 1. Khi người dùng kết thúc truy cập vào đối tượng quan trọng, người dùng sẽ được chuyển sang trạng thái ***exiting*** , và số lượng truy cập hiện tại giảm đi 1. Sau đó, người dùng được chuyển từ trạng thái thoát sang trạng thái idle. Quá trình ủy quyền có thể được mô hình hóa:



### Lớp chính sách Historical

Các chính sách trong Lớp chính sách Historical quy định quyền truy cập theo lịch sử trạng thái truy cập hoặc chuỗi sự kiện được ghi lại và xác định trước. Các chính sách AC đại diện cho loại chính sách AC này bao gồm các chính sách AC của Chinese Wall và Workflow. Lớp chính sách này có thể được mô tả tốt nhất bằng các biểu thức đặc tả đồng bộ hoặc trực tiếp của mô hình FSM. Ví dụ, đặc tả FSM đồng bộ trong ví dụ dưới dây chỉ rõ 1 chính sách AC Chinese Wall trong đó có hai xung đột nhóm lợi ích COI1, COI2 của các đối tượng:

****

Lưu ý rằng trong thực tế, các chính sách AC giống nhau có thể được biểu thị bằng nhiều mô hình AC khác nhau hoặc được thể hiện bằng một mô hình duy nhất ngoài các quy tắc ràng buộc bổ sung bên ngoài mô hình. Xác minh sự phù hợp của các chính sách và mô hình AC là một nhiệm vụ không quan trọng và quan trọng. Một khía cạnh quan trọng của xác minh đó là kiểm tra sự không nhất quán và không đầy đủ của mô hình và để xác minh các yêu cầu an toàn, bởi vì mô hình AC không nhất thiết phải thể hiện chính sách một cách rõ ràng, cũng có thể được nhúng ngầm bằng cách trộn với các ràng buộc truy cập trực tiếp hoặc các mô hình AC khác.

# An toàn và các lỗi điều khiển truy cập

Sự bảo mật của hệ thống bị xâm phạm do các mô hình AC và cơ chế của các chính sách AC bị lỗi nhiều hơn là do sự thất bại của các nguyên tắc hoặc giao thức mã hóa. Nhược điểm của AC bao gồm rò rỉ đặc quyền, chặn truy cập được ủy quyền và không đạt được kết quả chính xác trong các quyết định grant /deny. Các lỗi như vậy có thể dẫn đến các lỗ hổng nghiêm trọng, đặc biệt là khi các mô hình AC và quy tắc ràng buộc khác nhau được kết hợp thành một mô hình. Vấn đề này ngày càng trở nên nghiêm trọng khi các hệ thống trở nên phức tạp hơn và được triển khai để quản lý một lượng lớn thông tin và tài nguyên nhạy cảm được tổ chức thành các cấu trúc tinh vi.

## An toàn

An toàn là tính chất cơ bản của hệ thống AC, đảm bảo rằng mô hình AC sẽ không dẫn đến rò rỉ quyền đối với thủ phạm trái phép. Do đó, một mô hình AC được cho là an toàn nếu không có đặc quyền nào có thể được chuyển đến các thủ phạm trái phép hoặc ngoài ý muốn. Nhưng các đặc quyền chính xác luôn có thể truy cập được cho các đối tượng được ủy quyền. An toàn được chỉ định thông qua việc sử dụng các mô hình AC bị hạn chế có thể được chứng minh chung cho mô hình đó mô tả các yêu cầu an toàn của bất kỳ cấu hình nào.

Trong số tất cả các tính năng an toàn, Phân chia nhiệm vụ (SoD) (xem Phụ lục B) năng động hơn các tính năng khác. SoD đề cập đến nguyên tắc rằng không người dùng nào được cung cấp đủ đặc quyền để tự mình lạm dụng hệ thống. Ví dụ, người phê chuẩn tiền lương không nên là người có chuẩn bị chúng. Các SoD có thể được thi hành tĩnh (bằng cách xác định các vai trò xung đột, tức là, các vai trò không thể được thực thi bởi cùng một người dùng) hoặc động (bằng cách thực thi điều khiển tại thời điểm truy cập).

## Các nhược điểm AC

Các nhược điểm AC làm tổn hại đến sự an toàn của hệ thống. Lỗi AC thường được gây ra bởi sự thể hiện sai hoặc không hiệu quả của các quyền sở hữu AC hoặc thuật toán cấp phép. Ở cấp độ cú pháp, lỗi AC chỉ đơn giản là do lỗi thực thi trong cơ chế AC như lỗi mã hóa hoặc cấu hình sai của hệ thống AC. Nói chung, lỗi AC có thể được phân loại thành các lớp sau.

**Rò rỉ đặc quyền**

Rò rỉ đặc quyền đề cập đến các tình huống trong đó một chủ thể có thể truy cập các tài nguyên bị cấm bằng các điều kiện an toàn. Rò rỉ như vậy có thể gây ra sự leo thang đặc quyền từ một miền tài nguyên hoặc lớp đối với các vùng bị cấm như rò rỉ từ cấp thấp hơn đến cấp cao hơn của chính sách MLS hoặc rò rỉ đặc quyền như từ vai trò này sang vai trò bị cấm khác của chính sách RBAC. Rò rỉ đặc quyền có thể được gây ra bởi việc gán đặc quyền nhầm lẫn trực tiếp hoặc thừa kế đặc quyền bất cẩn một cách gián tiếp.

**Chặn đặc quyền**

Đối lập với rò rỉ đặc quyền, lỗi chặn đặc quyền chặn quyền truy cập hợp pháp vào các tài nguyên hợp pháp. Chặn đặc quyền cũng có thể xảy ra khi các thuộc tính của chính sách AC không thể đưa ra quyết định *grant* hoặc *deny*, hoặc không có logic có sẵn trong thuật toán chính sách AC để đánh giá yêu cầu truy cập. Chặn đặc quyền cũng có thể là kết quả của sự chồng chéo các quy tắc truy cập trong đó quy tắc có sự phụ thuộc vào (các) quy tắc khác, cuối cùng phụ thuộc vào chính quy tắc đó, do đó yêu cầu của chủ thể sẽ không bao giờ đưa ra quyết định vì tham chiếu theo chu kỳ .

**Kế thừa theo chu kỳ**

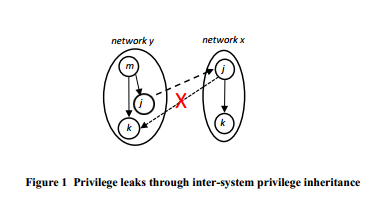
Lỗi kế thừa theo chu kỳ đề cập đến vấn đề kế thừa đặc quyền từ những người dùng (nhóm) khác, cũng thuộc chuỗi quan hệ kế thừa trở lại đặc quyền của người dùng (nhóm). Ví dụ: người dùng x kế thừa đặc quyền từ người dùng y, người dùng y kế thừa đặc quyền từ người dùng z, do đó người dùng z thừa hưởng đặc quyền từ người dùng x. Kế thừa theo chu kỳ dẫn đến quá trình đánh giá truy cập không thể xác định hoặc vô hạn.

**Xung đột đặc quyền**

Không giống như logic lập trình thông thường rằng việc gán giá trị sau của một biến sẽ ghi đè lên giá trị được gán trước đó của cùng một biến, các quy tắc của chính sách AC thường không được xem xét ưu tiên trong đánh giá quyền. Nói cách khác, các quy tắc AC sẽ không bị ghi đè bởi các quy tắc khác trừ khi được cho phép cụ thể. Do đó, xung đột đặc quyền xuất hiện khi thông số kỹ thuật của hai hoặc nhiều quy tắc truy cập dẫn đến các quyết định xung đột về việc cho phép các đối tượng truy cập yêu cầu bằng cách gán quyền truy cập trực tiếp hoặc gián tiếp (kế thừa). Ngoài ra, khi nhiều chính sách được gọi để cấp phép, các quyết định mâu thuẫn giữa các chính sách có thể xảy ra.

## Multi-policies considerations

Trong môi trường doanh nghiệp, có thể phải có các chính sách AC được chỉ định độc lập bởi các hệ thống hợp tác hoặc kết nối mạng khác nhau trong doanh nghiệp. Do đó, yêu cầu truy cập giữa các hệ thống có thể được đánh giá bởi nhiều hơn một chính sách mà chủ thể yêu cầu được điều chỉnh theo. Do đó, quyền tự chủ chính sách AC cũng cần được bảo đảm để truy cập an toàn giữa các hệ thống. Duy trì quyền tự chủ của tất cả các hệ thống hợp tác là một yêu cầu chính của chính sách đối với hoạt động. Nguyên tắc tự chủ tuyên bố rằng nếu một hệ thống riêng lẻ được phép truy cập thì nó cũng phải được cho phép dưới quyền truy cập liên hệ thống an toàn. Nguyên tắc bảo mật tuyên bố rằng nếu một hệ thống riêng lẻ bị từ chối truy cập thì nó cũng phải bị từ chối dưới quyền truy cập liên hệ thống an toàn. Trong một hệ thống hợp tác, có thể gây ra vi phạm quyền truy cập giữa các hệ thống an toàn bằng cách thêm quan hệ thừa kế đặc quyền giữa các hệ thống.



ví dụ, Hình 1 cho thấy đặc quyền k thừa hưởng đặc quyền j thông qua thừa kế đặc quyền liên hệ thống hợp pháp (vì cả hai đều có cùng đặc quyền cấp j), được cấp trong mạng x nhưng bị từ chối trong mạng y. Những loại vi phạm này có thể được phát hiện bằng cách kiểm tra tính kế thừa theo chu kỳ, rò rỉ đặc quyền và vi phạm SoD. Do đó, cả bảo mật và quyền tự chủ có thể được mô tả là các yêu cầu an toàn của chính sách AC đa chính sách, cần được bảo đảm trong quá trình hợp tác. Chính sách meta là một chính sách thường được áp dụng để điều hòa xung đột chính sách hoặc để xử lý các ưu tiên của các quyết định truy cập được đưa ra từ nhiều hơn một chính sách. Do đó, ngoài các yêu cầu tự chủ, yêu cầu an toàn AC có thể bao gồm một mô hình ưu tiên trong chính sách meta.

## Cấu trúc dễ bị tổn thương trong Cấu hình điều khiển truy cập

Nó đã được chứng minh rằng có một mối quan hệ phân cấp giữa các lỗ hổng trong các hệ thống kiểm soát truy cập, sao cho các điều kiện cho phép khai thác một lỗ hổng đủ để kích hoạt các lỗ hổng khác ở cấp thấp hơn trong hệ thống phân cấp. Mặc dù số lượng lỗ hổng tiềm năng dẫn đến lỗ hổng là rất lớn, cấu trúc của các quy tắc điều khiển truy cập dẫn đến một hệ thống phân cấp cho các loại lỗ hổng nhất định. Phần này giải thích các phân cấp này và cách chúng có thể được sử dụng để giảm số lượng thử nghiệm cần thiết.

**Cấu trúc quy tắc điều khiển truy cập**

Chính sách kiểm soát truy cập P được triển khai theo một bộ quy tắc R. Lỗ hổng có thể được định nghĩa là một điều kiện theo đó quyết định từ quy tắc R khác với kết quả dự định được chỉ định bởi chính sách P, nghĩa là R ≠ P. Ở cấp độ cao, các lỗ hổng có thể được nhóm thành các nhóm truy cập trái phép, từ chối dịch vụ (có thể là một phần) hoặc kết hợp cả hai, tùy thuộc vào các điều kiện theo đó quyết định là *grant/deny.* Đối với các quy tắc trong đó quyết định là *grant,* có ba khả năng dễ gây ra lỗ hổng, trong đó Rc = *grant*  trong chính sách đã triển khai và Pc = *grant* trong chính sách dự kiến:

1. Rc ⇒ Pc: từ chối dịch vụ (có thể một phần) (vì Rc ⊂ Pc).

2. Rc ⇐ Pc: truy cập trái phép (vì Rc ⊃ Pc)

3. khác: có thể kết hợp truy cập trái phép hoặc từ chối dịch vụ

Trong trường hợp quyết định bị từ chối, tình huống là hình ảnh phản chiếu của quy tắc *grant:*

1. Rc ⇒ Pc: truy cập trái phép (vì Rc ⊂Pc)

2. Rc ⇐ Pc: từ chối dịch vụ (có thể một phần) (vì Rc ⊃ Pc).

3. khác: có thể kết hợp truy cập trái phép hoặc từ chối dịch vụ

Cấu trúc quy tắc chính sách kiểm soát truy cập được định nghĩa ở đây là cấu hình của các toán tử logic, các điều khoản chính sách và các quyết định, được phân loại theo cách các quy tắc được xây dựng. Mỗi quy tắc có một điều kiện và một quyết định. Các điều kiện có thể bao gồm các điều kiện khác được kết nối bởi các toán tử logic, thường có cấu trúc tiêu chuẩn. Các quyết định thường là 2 giá trị, nhưng cũng có thể có ba hoặc nhiều giá trị (ví dụ*: grant, deny, defer,…*). Các phương pháp kết hợp quy tắc cũng có thể có một mô hình. Trong nhiều các trường hợp, một tập hợp các quy tắc có thể dẫn đến quyết định *grant* được theo sau bởi một trường hợp mặc định *“else deny*”. Ngoài ra, các quy tắc với các quyết định *grant* hoặc *deny* có thể được trộn lẫn, theo sau là một trường hợp mặc định. Do đó, một cách để phân loại cấu trúc quy tắc chính sách kiểm soát truy cập là chỉ định định dạng điều kiện, số lượng giá trị quyết định và định dạng kết hợp quy tắc.

Một nguyên tắc phân loại đã được xác định trong [Kuhn11] với các thuộc tính và giá trị cấu trúc quy tắc sau:

• Định dạng điều kiện: *con* - kết hợp các điều kiện; *dis* - không phân biệt các điều kiện; *cnf* - dạng kết hợp tiêu chuẩn của điều kiện.

• Số lượng giá trị quyết định: binary hoặc *n-ary*.

• Kết hợp quy tắc: *singular* (tất cả các quyết định quy tắc cùng loại, ví dụ: *grant* hoặc *deny*, theo sau là mặc định) hoặc *mixed* trong đó đề cập đến các quyết định *grant* và *deny* hỗn hợp trong quy tắc.

Các cấu trúc sau đó có thể được phân loại theo định dạng [định dạng điều kiện] / [số giá trị quyết định] / [phương thức kết hợp quy tắc]. Một số cấu trúc có thể được xác định bằng cách sử dụng phân loại này được thảo luận dưới đây.

A*. con / 2 / singular*: Các quyết định được xác định bởi sự kết hợp của các điều kiện theo đó quyền truy cập được cấp. Nếu không có quyết định “grant” nào phù hợp với cấu hình đầu vào, quyền truy cập sẽ bị deny.

B. *con / 2 / mixed*: Các quyết định được xác định bởi sự kết hợp của các điều kiện theo đó quyền truy cập được cấp hoặc từ chối. Nếu không có quyết định grant phù hợp với cấu hình đầu vào, quyền truy cập sẽ bị từ chối. Bộ quy tắc được xác định cho *con / 2 / singular* ngoại trừ các quyết định trên mặc định có thể là grant hoặc deny.

C. *dis / 2 / singular*: Các quyết định được xác định bởi sự phân biệt các điều kiện theo đó quyền truy cập được cấp. Nếu không có quyết định “grant” nào phù hợp với cấu hình đầu vào, quyền truy cập sẽ bị deny.

D. *dis / 2 / mixed*: Các quyết định được xác định bởi sự phân biệt các điều kiện theo đó quyền truy cập được cấp hoặc từ chối. Nếu không có quyết định grant phù hợp với cấu hình đầu vào, quyền truy cập sẽ là deny. Bộ quy tắc được xác định cho *disj / 2 / singular* ngoại trừ các quyết định trên mặc định có thể là grant hoặc deny.

E. *cnf / 2 / singular*: Các quyết định được xác định bởi các điều kiện ở dạng tiêu chuẩn kết hợp (CNF) theo đó quyền truy cập được cấp. Nếu không có quyết định “grant” nào phù hợp với cấu hình đầu vào, quyền truy cập sẽ bị từ chối. Lớp quy tắc này được bao hàm bởi vì nó tương đối phổ biến trong các vấn đề kiểm soát truy cập trong thực tế. Tiêu chuẩn XACML là một khung được sử dụng rộng rãi như vậy để thực hiện các chính sách kiểm soát truy cập.

F. *cnf / 2 / mixed* : Các quyết định được xác định bởi sự kết hợp của các điều kiện theo đó quyền truy cập được cấp hoặc từ chối. Nếu không có quyết định grant phù hợp với cấu hình đầu vào, quyền truy cập sẽ bị từ chối. Bộ quy tắc được xác định cho *cnf / 2 / singular* ngoại trừ các quyết định trên mặc định có thể là cấp hoặc từ chối.

G*. [điều kiện] / n / [kết hợp]* (n-ary quy tắc quyết định) Khi các quyết định có thể nhiều hơn cấp hoặc từ chối, cấu trúc quy tắc được xác định như trên với sửa đổi quyết định grant

**Các lớp dễ bị tổn thương**

Phần này xác định các phân cấp của lỗ hổng và xác định hệ thống phân cấp lỗ hổng cho các cấu trúc chính sách kiểm soát truy cập được xác định trước đó. Một số lượng lớn các lỗ hổng có thể được xác định cho các quy tắc kiểm soát truy cập, nhưng một bộ hợp lý có thể bao gồm các điều sau đây. Lưu ý rằng biểu mẫu chính xác sẽ thay đổi theo loại cấu trúc quy tắc truy cập như được xác định trong phân loại ở trên.

1. Thêm điều kiện: Một điều kiện không được chỉ định đã được thêm vào thực hiện

2. Xóa điều kiện: Một điều kiện cụ thể bị thiếu trong quá trình thực hiện

3. Điều kiện thay thế: Một điều kiện được thay thế bằng một điều kiện khác, không tương đương

4. Bị mắc kẹt ở điều kiện đúng: Một điều kiện luôn luôn đúng

5. Bị mắc kẹt ở điều kiện sai: Một điều kiện luôn luôn là sai

6. Điều kiện phủ định: Một điều kiện bị phủ định

7. Quyết định phủ định: Kết quả được chỉ định cho một điều kiện trái ngược với dự định. Đột biến này chỉ có thể áp dụng khi có hai quyết định có thể, thường là Grant hoặc Deny:

8. Xóa quy tắc: Một quy tắc được chỉ định đã bị bỏ qua khỏi quá trình thực hiện.

Trong triển khai thực tế, một số lỗ hổng này có thể phát sinh từ lỗi quản trị viên, chẳng hạn như vô tình xóa hoặc bỏ qua một điều kiện hoặc quy tắc và những lỗi khác có thể xảy ra do lỗi phần mềm, chẳng hạn như mô-đun nhằm xác minh thuộc tính và trả về đúng hoặc sai , nhưng luôn trả về đúng.

**Hệ thống phân cấp dễ bị tổn thương**

Đối với tám lớp dễ bị tổn thương được giới thiệu ở trên, các mối quan hệ phân cấp tồn tại cho các cấu trúc quy tắc kiểm soát truy cập trong phân loại. Trong các cấu trúc bên dưới, các lỗ hổng được viết tắt như sau:

1. ac: thêm điều kiện

2. dc: xóa điều kiện

3. rc: thay thế điều kiện

4. tc: điều kiện đúng

5. fc: điều kiện sai

6. nc: điều kiện phủ định

7. nd: quyết định phủ định

8. dr: xóa quy tắc

Lưu ý rằng bộ lỗ hổng này không được tuyên bố là hoàn chỉnh. Nó có thể được mở rộng, ví dụ, với các lỗ hổng như quy tắc thêm vào quy tắc, hay thay thế quy tắc. Mối quan hệ phân cấp giữa các điều kiện phát hiện cho các lớp dễ bị tổn thương khác nhau có thể được xác định bằng cách kiểm tra hàm ý giữa các cặp điều kiện phát hiện. Đó là, đối với tất cả các cặp lỗ hổng Pi, Pj, i ≠ j, tính P ⨁ Pi ⇒ P ⨁ Pj, trong đó Pi và Pj là các quy tắc truy cập cho các lỗi cụ thể dẫn đến các lỗ hổng.

Các cấu trúc phân cấp có thể được hiển thị tồn tại cho các chính sách có nhiều quy tắc (hai hoặc nhiều) với các quyết định Grant, trong đó mỗi quy tắc chứa hai hoặc nhiều điều kiện có thể được coi là các điều khoản Boolean và điều kiện tương tự bị ảnh hưởng trong chính sách đúng và sai. Phân tích dưới đây dành cho trường hợp phổ biến của các chính sách chứa một loạt quy tắc với các quyết định Grant, theo sau là quyết định Deny mặc định, trong đó mỗi quy tắc có một biến hoặc điều kiện duy nhất hoặc kết hợp hai hoặc nhiều điều kiện, như được hiển thị bên dưới.

if (c11 · . . . · c1n1) then grant;

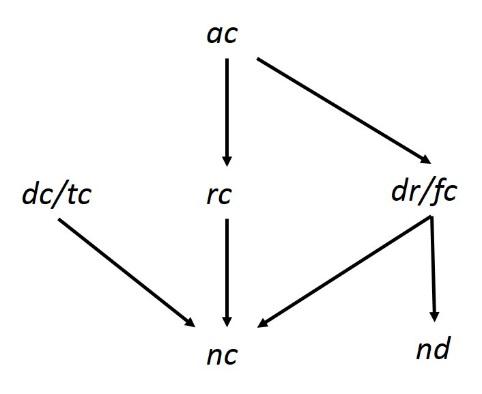
if (c21 · . . . · c2n2) then grant;

. . .

if (ck1 · . . . · ckn3) then grant;

else deny;

Các hệ thống phân cấp tương tự có thể được xây dựng cho các mẫu quy tắc khác trong phân loại được xác định ở trên, nhưng không được hiển thị ở đây.



**Figure 2** **Example Vulnerability Hierarchy**

Hình 2 cho thấy một ví dụ về phân cấp lỗ hổng, lưu ý rằng đối với các chính sách này, dc và tc là tương đương nhau, bởi vì đặt một thuật ngữ thành true trong một tập hợp các thuật ngữ trong một kết hợp tương đương với việc xóa thuật ngữ, ví dụ: đặt a thành đúng, abc = 1bc = bc. Tương tự, đặt thuật ngữ thành false cho lớp quy tắc này tương đương với việc xóa toàn bộ quy tắc, ví dụ: đặt a thành false, abc → G trở thành 0bc → G = 1 + G = 1.

**Ý nghĩa của việc thử nghiệm**

Hệ thống phân cấp cho thấy các thử nghiệm có khả năng xác định một số lỗi nhất định cũng sẽ xác định các loại lỗi khác được xác định trước bởi hệ thống phân cấp. Do đó, chúng tôi có thể xác định rằng một số thử nghiệm là dự phòng nếu chúng được thiết kế để kiểm tra hai loại có liên quan trong cấu trúc phân cấp. Tương tự, nếu kiểm tra sự thay đổi đang được sử dụng, hệ thống phân cấp cho chúng ta biết trước các thay đổi nào sẽ tương đương, mà không tạo thay đổi và áp dụng trình kiểm tra mô hình hoặc công cụ khác để phân tích logic.

Những khái niệm này có thể được làm rõ với một ví dụ. Lưu ý rằng trong hệ thống phân cấp tc, bao gồm một nc. Một chính sách giả định có thể được định nghĩa như sau:

 if (*a* & *b* & *c*) then *grant*; if (*d* & *e*) then *grant*; else *deny*

Mô hình

P = ((a b c) → G) ((d e) → G) • (~ (a b c) • ~ (d e) → ~ G)

Giả sử chính sách được thực thi không chính xác, bỏ qua điều kiện a:

Ptc = ((b c) → G) ((d e) → G) • (~ (b c) • ~ (d e) → ~ G)

Các điều kiện phát hiện cho loại lỗ hổng này trong Ptc là:

P ⨁ Ptc = a̅ b c d̅ + a̅ b c e̅

Nghĩa là, đối với các đầu vào của a̅ b c d̅ hoặc a̅ b c e̅, Ptc chính sách tạo ra kết quả không chính xác, do đó, thử nghiệm với một trong hai đầu vào này sẽ phát hiện lỗi. Một chính sách bị lỗi khác có thể thay thế điều kiện trong quy tắc 1 bằng phủ định của nó:

Pnc = ((~ a b c) → G) ((d e) → G) (~ (~ a b c) • ~ (d e) → ~ G)

Đối với Pnc thực hiện, các điều kiện phát hiện là:

P ⨁ Pnc = b c d̅ + b c e̅

Do đó, một thử nghiệm với b c d̅ + b c e̅ (và a hoặc a̅) sẽ phát hiện lỗi. Lưu ý rằng P ⨁ Ptc tức là, các điều kiện phát hiện cho Ptc bao gồm các điều kiện cho Pnc, do đó, một thử nghiệm phát hiện chính sách Ptc bị lỗi, với điều kiện thực, cũng sẽ phát hiện các chính sách có cùng điều kiện bị phủ định trong cùng quy tắc. Ví dụ: kiểm tra a̅ b c d̅ có thể phát hiện cả hai lỗ hổng bằng cách đánh giá G trong cả hai triển khai bị lỗi, thay vì kết quả đúng G̅.

# Các yếu tố xác minh chính sách

Xác định sự khác biệt giữa chính sách, mô hình và triển khai AC là rất quan trọng bởi vì việc thực thi chính sách dựa trên tiền đề rằng các đặc tả và thực thi chính sách là chính xác, do đó, việc xác định và thực thi chính sách phải được xác minh và kiểm tra nghiêm ngặt thông qua kiểm tra và xác minh có hệ thống để đảm bảo thực sự bao gồm đầy đủ các thuộc tính AC mong muốn từ người tạo ra chính sách. Lưu ý rằng không giống như các câu lệnh của ngôn ngữ lập trình chung, trong đó việc gán giá trị sau có thể ghi đè lên các biến trước đó của cùng một biến trong chuỗi logic lập trình, gán đặc quyền trong quy tắc AC không có thứ tự ưu tiên, nghĩa là gán đặc quyền không được ghi đè bởi những người sau này. Do đó, mục tiêu cơ bản của chính sách AC và xác minh thực hiện là phát hiện các quy tắc mâu thuẫn hoặc thiếu (nghĩa là các phát biểu của chính sách) bằng cách xác minh mô hình chính sách AC và thử nghiệm chính sách. Để đạt được điều này, có thể sử dụng các phương pháp semantically và syntactically với các kỹ thuật kiểm tra black-box hoặc white-box.

**Các yêu cầu an toàn**

Xác minh chính sách AC phải kiểm tra xem các yêu cầu an toàn của chính sách AC có được kết hợp trong mô hình được thể hiện hay không, đây sẽ là kế hoạch chi tiết để thực hiện hệ thống AC. Thông số kỹ thuật của các yêu cầu an toàn có thể là thuộc tính AC, yêu cầu nghiệp vụ, thông số kỹ thuật của các tính năng bảo mật hệ thống dự kiến ​​/ không mong muốn hoặc sự biểu diễn trực tiếp các tính năng chính sách. Các yêu cầu an toàn cũng có thể bao gồm kế thừa đặc quyền, ví dụ để xác minh phân chia nhiệm vụ (SoD), yêu cầu an toàn có thể chỉ định rằng

1) chủ thể x và y loại trừ lẫn nhau nếu không ai kế thừa trực tiếp hoặc gián tiếp

2) Nếu chủ thể x và y là loại trừ lẫn nhau, sau đó không có chủ thể nào khác được thừa hưởng đặc quyền từ cả hai. Tương tự như SoD, SoD động (DSoD) có yêu cầu an toàn

3) Nếu SoD tiến hành, thì DSoD được duy trì. Do đó, 1) và 2) phải được đảm bảo. (Antonios)

Lưu ý rằng chính sách AC không nhất thiết phải được thể hiện rõ ràng bằng một mô hình duy nhất; nó cũng có thể được nhúng hoàn toàn bằng cách trộn với các ràng buộc truy cập trực tiếp hoặc các mô hình AC khác. Do đó, chính sách AC có thể được thể hiện bằng cách kết hợp nhiều mô hình AC (ví dụ: kết hợp các chính sách) hoặc các ràng buộc bổ sung bên ngoài mô hình thành một mô hình kết hợp. Về nguyên tắc, đảm bảo sự phù hợp của một mô hình với chính sách là phát hiện chính thức các lỗi không thống nhất và không đầy đủ như được mô tả trong Phần 3. Ví dụ, trong trường hợp trước, yêu cầu truy cập có thể được chấp nhận và từ chối, trong trường hợp sau, yêu cầu truy cập không được chấp nhận cũng không bị từ chối theo mô hình.

## Xác minh mô hình

Cách tiếp cận chung để kiểm tra đặc điểm kỹ thuật chính xác của mô hình AC là sử dụng các phương pháp black-box để xác minh mô hình AC theo các yêu cầu an toàn. Và vì sự tin tưởng vào tính chính xác của mô hình phụ thuộc vào chất lượng của các yêu cầu an toàn, phương pháp đánh giá white-box trên các thực thể trong mô hình và các yêu cầu an toàn là cần thiết để đánh giá tính đầy đủ của sự an toàn, bao quát và hạn chế của mô hình.

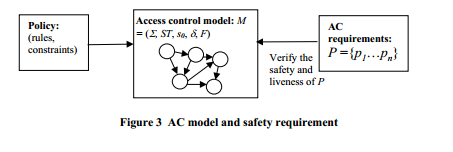
Về các thuộc tính AC, định nghĩa chính thức của mô hình AC có thể được minh họa bằng bộ chuyển đổi trạng thái hữu hạn xác định tương ứng với Máy trạng thái đơn định (FSM) với M = (Σ, ST, s0, δ , F), trong đó : Σ là điều kiện đầu vào đại diện cho các thuộc tính được liên kết với các chủ đề, hành động, đối tượng và điều kiện môi trường. ST là tập hợp hữu hạn, không rỗng của các trạng thái và quyền của hệ thống AC được ghi. s0 là trạng thái ban đầu, δ là hàm chuyển trạng thái, trong đó δ: ST × Σ → ST, F là tập hợp các trạng thái cuối cùng bao gồm Grant, Deny làm đầu ra.

Đối với các mô hình AC tĩnh như được mô tả trong Phần 2.2.1, FSM *Mstatic* không yêu cầu trạng thái bên trong st để đạt đến trạng thái cho phép, do đó F = ST = {Grant, Deny}, nghĩa là *Mstatic* chỉ là mô hình FSM đơn giản không có trạng thái chuyển tiếp. Đối với mô hình AC động như được mô tả trong phần 2.2.2 bảng chữ cái đầu vào của FSM *Mdynamic* là Σdynamic = {gCond1, …, gCondn}, trong đó gCondi là chỉ số ngưỡng của giới hạn truy cập, chẳng hạn như số người phải truy cập cùng lúc trong chính sách kiểm soát N-Person hoặc số lượng truy cập tối đa được phép cho chính sách Limited\_Number\_of\_Access. Đối với các mô hình AC lịch sử như được mô tả trong Phần 2.2.3, các điều kiện đầu vào của FSM *Mhistorical* là Σhistorical = - {gCond1, …, gCondn}, trong đó sCondi, aCondi và oCondi đóng góp vào bản ghi lịch sử được sử dụng như xác định các yếu tố cho quyết định cho phép tiếp theo. Lưu ý rằng các loại mô hình AC khác nhau có thể kết hợp thành một mô hình sao cho Mcombine = {Mstatic ∪ Mdynamic ∪ Mhistorical}

Yêu cầu an toàn AC p được biểu thị bằng mệnh đề p: ST × ST của FSM, có thể được dịch chung theo các công thức logic sao cho p = (*si*\**sCond1\*…\*sCondn\*aCond1\*…\* aCondn\* oCond1\** …*\* oCondn\*gCond1\* …\*gCondn*) → *d*, trong đó pP là tập hợp các yêu cầu an toàn và \* là toán tử Boolean theo các công thức logic của logic thời gian như logic thời gian phân nhánh (CTL) và logic thời gian (LTL). Mục đích của việc kiểm tra mô hình là để xác minh tập hợp ST trong M trong đó p là đúng theo tìm kiếm không gian trạng thái toàn diện. Ngoài ra, bằng cách xác minh tập hợp các trạng thái trong đó phủ định của p là đúng, chúng ta có thể có được tập hợp các mẫu phản để xác nhận rằng p là đúng. Sự hài lòng của mô hình AC M đối với yêu cầu an toàn AC P bằng cách kiểm tra mô hình bao gồm hai yêu cầu:

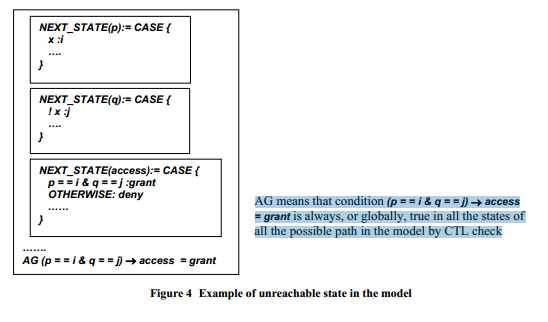
(1) An toàn, trong đó M thỏa mãn P. Nghĩa là không có vi phạm quy tắc nào đối với logic được chỉ định trong P và chắc chắn rằng M cuối cùng sẽ ở trạng thái mong muốn sau khi thực hiện hành động tuân thủ yêu cầu truy cập của người dùng .

(2) Tính động, trong đó M sẽ không có sự bế tắc trong đó hệ thống chờ mãi cho các sự kiện hệ thống, cũng không phải là livelock trong đó mô hình liên tục thực hiện các hoạt động tương tự mãi mãi. Hình 3 cho thấy mối quan hệ giữa M và P trong khung kiểm tra mô hình.



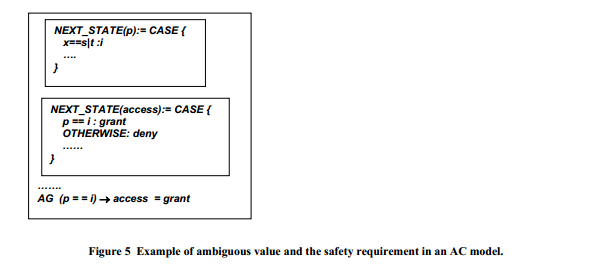
Do đó, các quy tắc AC xác định các hành vi hệ thống có chức năng như quan hệ chuyển tiếp δ trong M. Sau đó, khi yêu cầu an toàn AC được biểu diễn bằng công thức logic tạm thời p, chúng ta có thể biểu diễn khẳng định rằng mô hình M thỏa mãn p bởi M | = Ab → AXp , trong đó bộ lượng tử hóa logic tạm thời A đại diện cho “always”, và bộ định lượng logic X đại diện cho “is true next state”. Mục đích của xác minh an toàn và xác minh mức độ sử dụng bằng cách kiểm tra mô hình là để xác định xem các xác nhận này có đúng hay không và để xác định trạng thái trong đó các xác nhận không đúng như một ví dụ cho các xác nhận. Do hành vi của cơ chế AC có thể được biểu diễn bằng FSM M và các yêu cầu an toàn mà M phải đáp ứng có thể được biểu diễn bằng các công thức logic tạm thời, chúng ta có thể định nghĩa chính xác hơn vì mô hình có thể được dẫn từ mọi trạng thái có thể có thể truy cập từ trạng thái ban đầu đến trạng thái cuối cùng được xác định trong khi tuân thủ các yêu cầu an toàn.

Mặc dù được kiểm tra bằng thử nghiệm hộp đen như mô tả ở trên, mô hình không phải là chứng minh lỗi vì logic tạm thời trong mô hình có thể không kỹ lưỡng trong việc bao gồm tất cả các giá trị có thể của tất cả các quy tắc hoặc tất cả các điều kiện trong quy tắc. Ví dụ, hai trạng thái được xác định bởi các phép gán ngược nhau của cùng một biến Boolean được nhúng trong các mô đun trạng thái phụ khác nhau, trong đó trạng thái thứ ba chỉ được kích hoạt khi các ràng buộc của hai trạng thái được thỏa mãn. Như được minh họa trong Hình 4, hai quy tắc sẽ không bao giờ đồng ý do sự tự phủ định cho cùng một ràng buộc. Trong trường hợp này, trạng thái thứ ba sẽ không bao giờ được thỏa mãn, nhưng đã được chứng minh là đúng nếu không có phản ứng mẫu thông qua hộp đen có sẵn

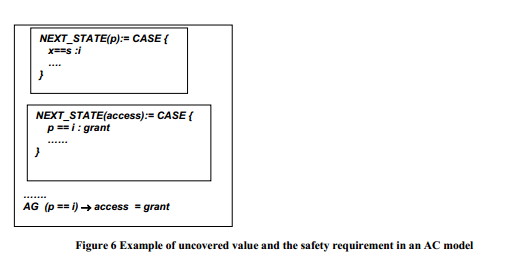


Để phát hiện loại lỗi ngữ nghĩa này, nên áp dụng thử nghiệm hộp trắng dựa trên phân tích mã sao cho kết quả các phiên bản đột biến được sử dụng để phát hiện lỗi của mô hình. Kiểm tra các đột biến [MX07] đảm bảo tất cả các đường dẫn của một phần của mã mô hình được bao phủ bằng cách đặt các biến mục tiêu liên quan thành tất cả các giá trị có thể làm đầu vào và kiểm tra xem liệu có kết quả khác với các thay đổi hay không. Nếu không có, thì mã đã bị đột biến sẽ không bao giờ được thực thi hoặc biến không thể xác định được lỗi. Như trong Hình 4, nếu chúng ta thay đổi mô-đun trường hợp đầu tiên để thay đổi x thành !x, kết quả trạng thái truy cập sẽ được cấp mà không bị ảnh hưởng. (Cái đó hoạt động tương tự cho mô-đun trường hợp thứ hai). Lỗi này chứng tỏ rằng có một sự dư thừa trong mô hình, không vi phạm logic thời gian của mô hình. Nghiên cứu sâu hơn để kiểm tra mô hình liên quan đến biến sẽ cho thấy rằng **(p = = i & q = = j) → access = = grant** yêu cầu an toàn sẽ không bao giờ xảy ra. Lưu ý rằng lỗi này có thể bị bắt nếu có thêm một yêu cầu an toàn ! (p = = i & q = = j) (có nghĩa là tồn tại một số đường dẫn mà cuối cùng trong tương lai sẽ thỏa mãn! (p = = i & q = = j) trong kiểm tra mô hình CTL) được chỉ định. Do đó, không phải tất cả các yêu cầu an toàn đều được quy định hoàn hảo ngay từ đầu. Do đó, kiểm tra hộp trắng có thể được sử dụng như một tuyến phòng thủ thứ hai chống lại các lỗi sẽ không bị phát hiện khi kiểm tra hộp đen.

Hầu hết các lỗi trong mô hình AC là kết quả của tính tự động không xác định của các trạng thái FSM, ví dụ, trong Hình 5, kiểm tra hộp trắng sẽ phát hiện ra rằng giá trị x sẽ dẫn đến việc cấp quyền truy cập khi nó là s hoặc t. Điều này không vi phạm các yêu cầu an toàn; tuy nhiên, thuộc tính an toàn sẽ không được đáp ứng nếu yêu cầu an toàn nghiêm ngặt hơn yêu cầu chỉ có một giá trị của thuộc tính x được yêu cầu từ chính sách.

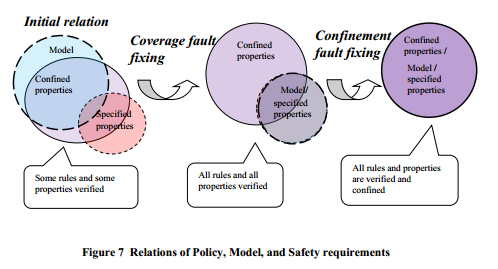


Một ví dụ khác cho thấy sự chuyển đổi sang trạng thái không xác định cho một phạm vi giá trị dữ liệu nhất định như trong Hình 6. Không có cách nào để trình kiểm tra hộp đen xác định giá trị truy cập khi giá trị x khác với s trừ khi chúng ta kiểm tra độ an toàn yêu cầu **AG! (p = = i) → access = deny**. Giá trị chưa được khám phá này có thể được phát hiện bằng cách kiểm tra hộp trắng khi các giá trị khác nhau được gán cho x, không khớp với bất kỳ điều kiện trường hợp dự kiến nào và dẫn đến cùng một quyền truy cập. Do đó, xác minh yêu cầu an toàn thông báo cho người dùng những quy tắc không được quy định trong yêu cầu an toàn hiện có để người dùng có thể thêm các thuộc tính mới để bao quát các điều chưa được khám phá



## Xác minh phạm vi và hạn chế

Mỗi quy tắc trong chính sách, mô hình và các yêu cầu an toàn có thể mô tả không gian điều kiện cấp phép của riêng họ và có thể không đồng nhất trong một không gian như các ví dụ minh họa quan hệ ban đầu trong Hình 7. Kiểm tra độ an toàn và tính động chỉ có thể đảm bảo tính toàn vẹn logic một số quy tắc ngược lại một số yêu cầu an toàn. Sự hài lòng hoàn toàn của một mô hình đối với chính sách của nó đòi hỏi phải sửa chữa phạm vi và lỗi hạn chế nếu có bất kỳ vi phạm nào được phát hiện bởi Kiểm tra phạm vi và hạn chế (CCC), tuyến phòng thủ thứ hai chống lại các lỗi ngữ nghĩa đó.

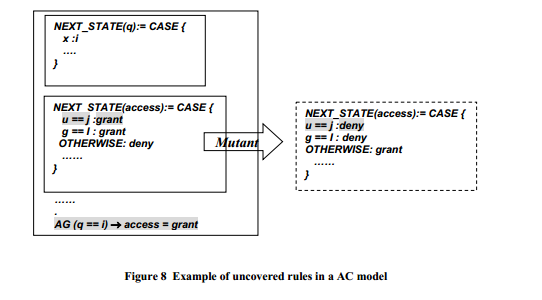


CCC yêu cầu các phiên bản đột biến của mô hình và các thuộc tính được sửa đổi bổ sung để kiểm tra mô hình bổ sung. Như minh họa trong Hình 7, mục tiêu của CCC là đảm bảo rằng các quy tắc trong yêu cầu an toàn được mô hình hoàn toàn bao phủ và để xác nhận rằng không có quyền truy cập đặc biệt nào được cấp trừ khi được cho phép có chủ ý. Bước đầu tiên của CCC là phát hiện các quy tắc được thông qua các đặc điểm kỹ thuật của yêu cầu an toàn bằng cách áp dụng kiểm tra hộp trắng trên các phiên bản đột biến của mô hình. Bước thứ hai là phát hiện sự cho phép truy cập bất ngờ có thể không phải là ý định của tác giả chính sách, bằng cách áp dụng kiểm tra mô hình trên các quy tắc đã sửa đổi được trích xuất từ ​​các quy tắc ban đầu.

**Kiểm tra phạm vi quy tắc**

Khái niệm chính của kiểm tra phạm vi quy tắc là tổng hợp một phiên bản của mô hình đã cho theo cách cho phép các quy tắc của nó bị thay đổi sao cho quy tắc r được thay đổi thành ~ r. Nếu các yêu cầu an toàn được thỏa mãn bởi cả mô hình đột biến và mô hình gốc thông qua kiểm tra mô hình, thì một số quy tắc và đột biến của chúng sẽ không bao giờ được áp dụng cho các yêu cầu an toàn; nói cách khác, các yêu cầu an toàn không bao gồm tất cả các quy tắc trong mô hình.

Như một ví dụ trong Hình 8, kiểm tra độ an toàn và độ bền chứng minh rằng mô hình tuân thủ yêu cầu an toàn AG (q = = i) → access = Grant mà không có phản mẫu; tuy nhiên, áp dụng CCC bằng cách thay đổi quy tắc u == j: grant và u == j: deny cho kiểm tra phạm vi, kết quả cho thấy rằng yêu cầu an toàn cũng thỏa mãn các quy tắc bị đột biến (không có phản mẫu), chỉ ra rằng biến u không bao giờ được áp dụng cho yêu cầu an toàn **AG *(q = = i)* → *access = grant*.**

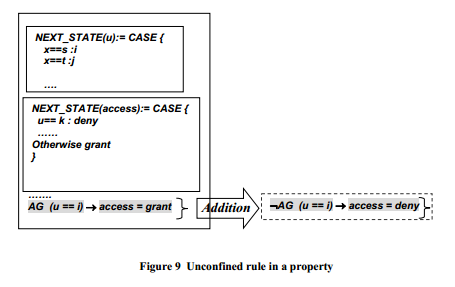
****

Kết quả này cho thấy quy tắc u == j: Grant không được xác minh với thuộc tính AG (q == i) → access = Grant. Một cách để giải quyết sự thiếu sót này là bằng cách thêm một thuộc tính mới mô tả sự kiểm soát thích hợp của u. Lưu ý rằng cần phải kiểm tra mọi quy tắc trong mô hình dựa trên tất cả các yêu cầu an toàn để đạt được xác minh kỹ lưỡng.

**Kiểm tra hạn chế**

Kiểm tra hạn chế tài nguyên đảm bảo rằng không có sự cho phép đặc biệt nào được phép ngoài yêu cầu an toàn được chỉ định; kiểm tra này đòi hỏi một yêu cầu an toàn sửa đổi phải được thêm vào cho lần chạy tiếp theo của kiểm tra mô hình. Kiểm tra hạn chế sẽ phát hiện ra sự khác biệt của yêu cầu an toàn được chỉ định và yêu cầu an toàn mà tác giả chính sách AC dự định. Lý do là nếu mô hình không đáp ứng yêu cầu an toàn đã sửa đổi, thì có các quyền truy cập đặc biệt rò rỉ thông qua yêu cầu an toàn.

Hình 9 cho thấy sự chuyển đổi sang trạng thái không xác định cho một phạm vi giá trị dữ liệu nhất định cho phép quyền đặc biệt không được bảo vệ bởi một yêu cầu an toàn được chỉ định vì giá trị truy cập khi giá trị u khác với i (chẳng hạn như u = j) cũng cấp quyền truy cập theo quy tắc khác ban cho. Lỗi này có thể được bắt gặp bởi một ví dụ mẫu AG (u == j) → access = Grant khi kiểm tra mô hình dựa trên thuộc tính hạn chế bổ sung ¬AG (u == i) → access = deny có nguồn gốc từ tài nguyên gốc AG (u == i ) → access = grant. Việc kiểm tra mô hình bổ sung để xác minh hạn chế thông báo cho các tác giả chính sách AC về yêu cầu an toàn không bị giới hạn để tác giả chính sách AC có thể thêm các quy tắc mới để thực thi sự an toàn của mô hình. Như trong trường hợp này, thay đổi quy tắc **otherwire:grant** bởi **otherwise:deny** và thêm tất cả các quy tắc được cấp trong trạng thái sẽ khắc phục vấn đề.

****

Lưu ý rằng có thể tác giả chính sách AC đã cố ý cho phép ngoại lệ cho yêu cầu an toàn và cần kiểm tra mọi yêu cầu an toàn đối với bộ quy tắc trong mô hình để đạt được xác minh kỹ lưỡng.

## Kiểm tra thực hiện

Kiểm tra mô hình hộp đen và kiểm tra đột biến hộp trắng cung cấp các phương pháp để xác minh biểu diễn mô hình chính xác của chính sách. Khi một mô hình được xác minh, cơ chế AC có thể được thực hiện dựa trên thiết kế của mô hình và các ràng buộc bổ sung nếu cần. Thông thường các cơ chế AC là mã được phát triển bằng ngôn ngữ mà hệ thống AC hỗ trợ, ví dụ ngôn ngữ AC chuyên dụng như XACML [XACML] thường được sử dụng để triển khai mã AC. AC thực hiện có thể dễ bị lỗi. Do mô hình AC được triển khai trực tiếp bằng thuật toán, các lỗi thường do lỗi cú pháp, chẳng hạn như thay đổi nhầm dấu + thành dấu - hoặc gõ chữ O thay vì 0.

Việc thực hiện chính xác của chính sách cần phải được kiểm tra. Để đạt được điều đó, cần phải có một thử nghiệm (bao gồm các trường hợp về tất cả các kết quả có thể có của yêu cầu an toàn AC, bởi vì lỗi thực thi là không thể đoán trước mà không có dấu vết logic để phát hiện. Do đó, tất cả các kết hợp các biến trong các yêu cầu an toàn cần phải được Ví dụ, yêu cầu về an toàn: Ví x read y Grant, trong đó x có 3 giá trị khác nhau và y có 5 giá trị khác nhau thì sẽ có 3\* 2 \*5 \* 2 = 60 trường hợp thử nghiệm (giả sử rằng hành động AC có hai giá trị: đọc và ghi và quyền chỉ có hai giá trị: cấp và từ chối). Hệ thống AC được triển khai sau đó sẽ chạy các trường hợp kiểm tra này để xác minh xem các đầu ra thử nghiệm thực tế có giống với đầu ra dự kiến ​​hay không.

Không có gì lạ khi một bài kiểm tra xác minh bao gồm hàng trăm yêu cầu an toàn; mỗi biến chứa hàng chục biến, trong trường hợp đó, số lượng trường hợp thử nghiệm trong một lời tiên tri cho thử nghiệm thực hiện là quá lớn để được thực hiện một cách hiệu quả, do đó, các kỹ thuật bổ sung [ACTS] [HKX08] để giảm kích thước trường hợp thử nghiệm mà không làm mất đi khả năng có thể được yêu cầu cho bài kiểm tra. Phần 5.6 mô tả các phương pháp tổ hợp để thực hiện các vấn đề lớn như vậy.

# Kỹ thật xác minh và kiểm thử

Để xác minh tính chính xác của mô hình AC, mô hình và các yêu cầu an toàn đã được xác minh được chỉ định trong một công thức logic chính thức (bao gồm bằng chứng logic đầu tiên và cao hơn). Ví dụ, các yêu cầu an toàn đối với mô hình được xác minh bằng trình kiểm tra mô hình. Trong quá trình này, sự tin cậy vào tính chính xác của mô hình phụ thuộc vào chất lượng của các yêu cầu an toàn được chỉ định. Do đó, phân tích hộp trắng có thể được áp dụng để kiểm tra xem các thực thể (tức là các quy tắc) của mô hình có được phạm vi đầy đủ và giới hạn bởi các yêu cầu an toàn, cũng như chất lượng của các yêu cầu an toàn nhất định hay không. Cuối cùng, các trường hợp thử nghiệm được tạo từ các biến trong mô hình AC và các yêu cầu an toàn được chỉ định. Các đầu vào thử nghiệm này được đưa vào triển khai AC thực của mô hình đã cho để xác minh xem đầu ra thử nghiệm thực tế có giống với đầu ra dự kiến ​​hay không. (IJEKE)

## Trình kiểm tra mô hình

Trình kiểm tra mô hình cung cấp các mô tả phân cấp mô-đun và định nghĩa các thành phần có thể sử dụng lại của đặc điểm kỹ thuật của Máy trạng thái đơn định (FSM) bao gồm từ đồng bộ đến không đồng bộ và từ chi tiết đến trừu tượng. Các kiểu dữ liệu duy nhất trong đặc tả là Booleans hữu hạn, vô hướng, mảng cố định và các kiểu dữ liệu tĩnh có thể được xây dựng. Mô hình được chỉ định trong một FSM mô tả các trạng thái chuyển tiếp của FSM. Nói chung, bất kỳ biểu thức nào trong phép tính mệnh đề đều có thể được sử dụng để xác định mối quan hệ chuyển tiếp của các trạng thái; tuy nhiên, tính linh hoạt của biểu thức đi kèm với rủi ro mâu thuẫn logic, điều này làm cho các thông số kỹ thuật trở nên chân thực hoặc làm cho hệ thống không thể thực hiện được. Về cơ bản, có ba loại cơ bản (như được mô tả trong Phần 2.2) của các biểu thức FSM để chỉ định các mô hình AC theo trình tự của các chuyển đổi trạng thái:

* Đồng bộ. Một mô hình AC được biểu thị bằng cách xác định giá trị của các ràng buộc AC ở trạng thái tiếp theo (sau mỗi lần chuyển đổi), được đưa ra bằng giá trị của các ràng buộc trong các trạng thái hiện tại (trước khi chuyển đổi).
* Không đồng bộ. Một mô hình AC được thể hiện bằng một tập hợp các trạng thái đồng thời. Loại mô hình này dành cho các hệ thống AC có quyết định ủy quyền được kích hoạt từ nhiều vùng như loại trừ lẫn nhau, giao thức truyền thông và mạch không đồng bộ.
* Chỉ định trực tiếp. Một mô hình AC được chỉ định trực tiếp theo các công thức mệnh đề. Tập hợp các trạng thái ban đầu có thể được chỉ định làm công thức trong các biến trạng thái hiện tại. Một trạng thái là ban đầu nếu nó thỏa mãn công thức. Mối quan hệ chuyển tiếp được chỉ định trực tiếp như một công thức mệnh đề theo các giá trị hiện tại và tiếp theo của biến trạng thái. Bất kỳ cặp trạng thái / trạng thái tiếp theo hiện tại đều nằm trong mối quan hệ chuyển tiếp khi và chỉ khi nó thỏa mãn công thức.

**Kiểm tra mô hình cơ bản FSM**

Dựa trên loại trạng thái chuyển tiếp (như mô tả ở trên) của đặc tả FSM, trình kiểm tra mô hình xác minh xem một mô hình có đáp ứng một tập hợp các yêu cầu an toàn mong muốn được chỉ định bởi người dùng hay không. Các thông số kỹ thuật có thể được kiểm tra bằng các logic thời gian như Logic thời gian phân nhánh (CTL) và Logic thời gian tuyến tính (LTL) được mở rộng với các toán tử quá khứ. Công thức CTL và TTL bao gồm công thức logic mệnh đề , cộng với kết nối thời gian. Các công thức logic mệnh đề là các biểu thức về trạng thái của hệ thống. Các kết nối tạm thời là các biểu thức về các đường dẫn đến trạng thái của hệ thống có thể theo. Và trình kiểm tra mô hình xây dựng và in một ví dụ mẫu, tức là một dấu vết của FSM làm sai lệch tài nguyên.

CTL là một logic thời gian phân nhánh có tính đến sự tiến hóa phân nhánh, không xác định của một FSM. Chính xác hơn, sự tiến hóa của một FSM sao cho từ một trạng thái nhất định, sự tiến hóa của FSM có thể được mô tả như một cây vô hạn, trong đó các nút là các trạng thái và sự phân nhánh đại diện cho sự chuyển tiếp không xác định của các trạng thái. Các đường dẫn trong cây từ trạng thái này sang trạng thái khác là sự phát triển có thể có của FSM từ trạng thái đó. Trong CTL, người ta có thể biểu thị các thuộc tính nên giữ cho tất cả các đường dẫn bắt đầu ở trạng thái, cũng như đối với các thuộc tính chỉ giữ cho một số đường dẫn.

LTL chia sẻ một giao điểm quan trọng bao gồm hầu hết các thuộc tính phổ biến được sử dụng trong CTL; nó đặc trưng cho từng đường dẫn tuyến tính gây ra bởi FSM (cách tiếp cận thời gian tuyến tính). Sự khác biệt chính giữa các công thức LTL và CTL là sự vắng mặt của các toán tử phổ biến và phổ quát, do đó các toán tử tạm thời LTL không có các bộ định lượng đường dẫn. Trong thực tế, các công thức LTL được đánh giá trên các đường tuyến tính và một công thức được coi là đúng trong một trạng thái nhất định nếu nó đúng với tất cả các đường dẫn bắt đầu ở trạng thái đó, phản ánh mô hình thời gian tuyến tính. Các thuộc tính LTL cũng có thể bao gồm các toán tử tạm thời trong quá khứ.

Các toán tử CTL và LTL có thể được lồng trong một cách tùy ý và có thể được kết hợp bằng các toán tử logic. Các ví dụ điển hình của công thức CTL là AG! P (Điều kiện p không có trong tất cả các điều kiện), AGEF p (Luôn luôn có thể đạt đến trạng thái nơi p có thể áp dụng) và AG (p → AF q) (mỗi lần xuất hiện điều kiện p được theo sau bởi sự xuất hiện của điều kiện q). Và ví dụ điển hình của LTL là G p, nói rằng điều kiện p giữ trong tất cả các yếu tố thời gian trong tương lai và F p, nói rằng điều kiện p giữ ở một trong những thời điểm tương lai.

Trình kiểm tra mô hình cũng có thể tìm thấy một ví dụ mẫu có chiều dài tăng dần và dừng ngay lập tức khi thành công, tuyên bố rằng công thức là sai. Số lần lặp tối đa có thể được chỉ định sao cho nếu đạt được số lần lặp tối đa và không tìm thấy mẫu phản hồi, thì việc kiểm tra được chấm dứt mà không có kết luận rằng công thức là đúng, ngoại trừ bất kỳ mẫu phản ứng nào phải dài hơn độ dài tối đa.

Nói chung, Kiểm tra mô hình giới hạn (BMC) có thể tìm thấy hai loại phản mẫu, tùy thuộc vào tài sản được phân tích. Đối với các yêu cầu an toàn với số lần chuyển tiếp giới hạn, mẫu phản là một chuỗi chuyển tiếp hữu hạn qua các trạng thái khác nhau. Đối với các thuộc tính sinh động, các mẫu phản là các chuỗi vô hạn nhưng định kỳ và có thể được biểu diễn trong một thiết lập giới hạn dưới dạng tiền tố hữu hạn theo sau là một vòng lặp, tức là một chuỗi hữu hạn các trạng thái kết thúc bằng một vòng lặp trở lại trạng thái trước đó. Vì vậy, một ví dụ mẫu chứng minh sự giả dối rằng một yêu cầu an toàn không thể là một chuỗi chuyển tiếp hữu hạn.

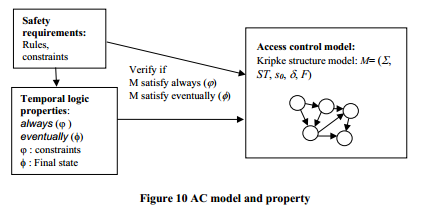
**Đặc tả mô hình AC trong Trình kiểm tra mô hình**

Từ quan điểm kiểm tra mô hình, sự hài lòng của mô hình AC đến các yêu cầu an toàn bao gồm hai yêu cầu. Những yêu cầu này có thể được chỉ định như sau trong các điều khoản kiểm tra mô hình.

(1) An toàn có nghĩa là mô hình AC đáp ứng các yêu cầu an toàn đã chỉ định. Ẩn ý trong mô tả về an toàn này là không có vi phạm các ràng buộc được chỉ định trong các yêu cầu an toàn và được đảm bảo rằng mô hình cuối cùng sẽ ở trong tình huống mong muốn sau khi nó thực hiện các hành động tuân thủ mô hình.

(2) Tính động có nghĩa là mô hình AC sẽ không có bế tắc trong đó mô hình chờ mãi cho các sự kiện AC, cũng không phải là một livelock trong đó mô hình liên tục thực hiện các hoạt động tương tự mãi mãi.

Hình 10 mô tả mối quan hệ giữa mô hình AC, các yêu cầu an toàn và khung kiểm tra mô hình. Mô hình cấu trúc được định nghĩa trong cấu trúc Kripke M = (, ST, s0, δ, F) như được mô tả trong Phần 4.1 và các yêu cầu an toàn được mô tả bởi một bộ quy tắc truy cập và các thực thể hệ thống. Chính sách này sau đó sẽ được dịch thành một tập hợp các mệnh đề logic có giá trị thật là đúng trong các tình huống. Các đề xuất nguyên tử này có thể đại diện cho việc gán các tham số trong mỗi tình huống.



Quy tắc AC xác định hành vi mô hình có chức năng như mối quan hệ chuyển tiếp M = (Σ, ST, s0, δ, F). Sau đó, khi ràng buộc được biểu diễn bằng công thức CTL, chúng ta có thể biểu diễn khẳng định (yêu cầu an toàn) mà mô hình M luôn thỏa mãn ϕ bởi M | = A bằng cách sử dụng toán tử logic tạm thời A để biểu diễn “always”. Tương tự, khi trạng thái cuối cùng được biểu thị bằng công thức CTL, khẳng định (yêu cầu an toàn) mà mô hình M cuối cùng dẫn đến trạng thái thỏa mãn có thể được đại diện bởi M | = Eφ bằng cách sử dụng toán tử logic tạm thời E để đại diện cho cuối cùng là Nero. Mục đích của xác minh và xác nhận bằng cách sử dụng kiểm tra mô hình là để xác định xem các xác nhận này (yêu cầu an toàn) có đúng không và để xác định trạng thái trong đó các xác nhận (yêu cầu an toàn) không đúng như một ví dụ mẫu. Do hành vi của mô hình có thể được biểu diễn bằng cấu trúc Kripke, nên yêu cầu an toàn mà mô hình phải đáp ứng có thể được biểu diễn bằng các công thức logic tạm thời, như trong Hình 10. Sử dụng định nghĩa trước, chúng ta có thể xác định tính hợp lệ của các yêu cầu an toàn chính xác hơn vì yêu cầu an toàn có hiệu lực đối với mô hình nếu mô hình có thể được dẫn từ mọi trạng thái có thể tiếp cận từ trạng thái ban đầu đến trạng thái cuối cùng được xác định trong khi tuân thủ quy tắc vận hành và không vi phạm các ràng buộc nhất định [KTAK07].

Sự chuyển đổi trạng thái của các mô hình AC có thể được phân loại theo ba loại ràng buộc, đó là các ràng buộc tĩnh, động và lịch sử. Những ràng buộc này có thể được thể hiện trực tiếp bằng ba loại mô hình FSM cơ bản. Do đó, một mô hình AC và các yêu cầu an toàn của nó có thể được chỉ định trong môi trường kiểm tra mô hình như các ví dụ dưới đây:

Đối với mô hình tĩnh Ví dụ 1 trong Mục 2.2, các thông số kỹ thuật của các ràng buộc tĩnh được xác minh thông qua ví dụ về các yêu cầu an toàn sau đây được thể hiện trong công thức logic tạm thời:

*AG (constraint\_1 & constraint\_2 & …. constraint\_n)*  *AX (access\_state = 1)  
AG (constraint\_a & constraint\_b & …. constraint\_m)*  *AX (access\_state = 1) ……  
AG ! ((constraint\_1 & ….constraint\_n) | (constraint\_a & …. constraint\_m) |… ) AX (access\_state = 0)*

điều đó chỉ có nghĩa là tất cả các yêu cầu truy cập tuân thủ các ràng buộc đã chỉ định phải được cấp và tất cả các yêu cầu không tuân thủ sẽ bị từ chối. Biểu mẫu của AG (b) → AX (d) cho biết rằng đối với tất cả các đường dẫn (chữ A trong chữ AG ) cho tất cả các trạng thái (chữ G), nếu b giữ thì (chữ →) các đường dẫn, ở trạng thái tiếp theo (các phiên bản X trong AX) d sẽ giữ.

Đối với mô hình động Ví dụ 2 trong Mục 2.2, các thông số kỹ thuật của các ràng buộc động được xác minh thông qua ví dụ về các yêu cầu an toàn sau đây được thể hiện trong công thức logic tạm thời:

*AG (state == entering) & (act == rd) & (object == obj* *AX (access = grant)  
AG (state == idle | state == critical | state == exiting)* *AX (access = deny)*

trong đó công thức logic tạm thời AG (b) AX (d) chỉ ra rằng nếu trạng thái đúng trong điều kiện b, điều kiện d luôn đúng ở trạng thái muộn hơn trạng thái.

Đối với mô hình lịch sử Ví dụ 3 trong Mục 2.2, các thông số kỹ thuật của các ràng buộc lịch sử được xác minh thông qua ví dụ về các yêu cầu an toàn sau đây được thể hiện trong công thức logic tạm thời:

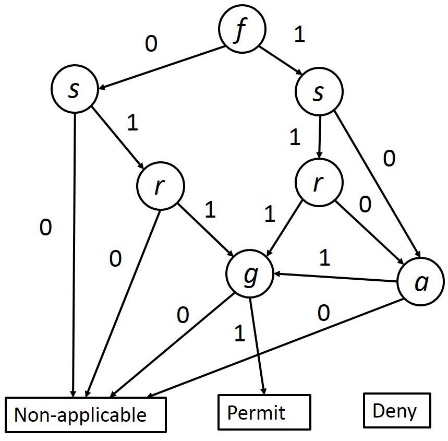
*AG ((u\_state == 2 & act == rd & o\_state == COI1) | (u\_state == 3 & act == rd & o\_state == COI2))* *AX (access = grant)  
AG ! ((u\_state == 2 & act == rd & o\_state == COI1) | (u\_state == 3 & act == rd & o\_state == COI2))*  *AX (access = deny)*

logic thời gian AG (b) AX (d) chỉ ra rằng sự kiện truy cập d được viện dẫn bởi các sự kiện lịch sử trong b.

Các phương pháp kiểm tra mô hình được triển khai trong NuSMV [NuSMV-M] được sử dụng bởi một số công cụ xác minh chính sách AC của NIST như ACPT và MOHAWK [MOHAWK].

## Sơ đồ quyết định nhị phân đa đầu cuối (MTBDD)

Đại diện cho các chính sách kiểm soát truy cập, Sơ đồ quyết định nhị phân đa đầu cuối (MTBDD) là một dạng sơ đồ quyết định ánh xạ các vectơ bit qua một tập hợp các biến đến một tập hợp hữu hạn của kết quả. Hình 11 cho thấy một ví dụ về một chính sách MTBDD đại diện cho một chính sách bảo mật đơn giản trong đó giảng viên (f) có thể gán (a) điểm (g) và sinh viên có thể nhận (r) điểm (g). Chính sách MTBDD có năm biến (f, s, r, a và g). Mỗi kết hợp các giá trị Boolean trên các biến này ánh xạ tới một trong ba kết quả chính sách (cho phép, từ chối hoặc không áp dụng); các kết quả được biểu thị bằng các thiết bị đầu cuối của chính sách MTBDD. Đưa ra một phép gán các giá trị Boolean cho các biến, đi qua một chính sách MTBDD từ gốc đến đầu cuối theo các giá trị biến cho biết kết quả của chính sách theo phép gán đó.

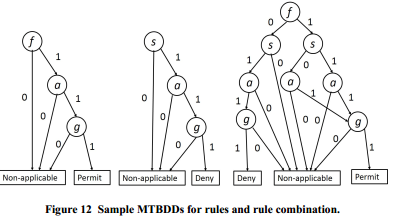


Các MTBD có ba đặc điểm xác định, có giá trị để xác minh hộp đen mô hình AC. Đầu tiên, chúng được xây dựng liên quan đến một số thứ tự cố định trên các biến (đọc từ gốc đến đầu cuối): một MTBDD không cho phép hai cây con kiểm tra các biến theo các thứ tự khác nhau. Hạn chế này làm cho các MTBDD trở thành một đại diện chính tắc cho đến thứ tự biến đã chọn. Thứ hai, các MTBDD chia sẻ tối đa các cây con, trong đó nhiều nhất là một bản sao của cây con xuất hiện ở bất cứ đâu trong sơ đồ quyết định; ví dụ, ba đường dẫn đến cùng một nút trong Hình 11. Thứ ba, các MTBDs thu gọn các biến không liên quan, nghĩa là nếu cả hai giá trị của cùng một biến đều tham chiếu đến cùng một cây con, thì nút cho biến đó sẽ bị xóa và tất cả các tham chiếu đến nó đều được chuyển hướng đến cây con chung. Hiệu ứng kết hợp của hai đặc điểm sau trên MTBDD trong Hình 11 cho thấy các biến của khoa và sinh viên là sai: thay vì liệt kê các biến còn lại của các giá trị, các giá trị trực tiếp đề cập đến thiết bị đầu cuối không áp dụng.

Mặc dù trong trường hợp xấu nhất, số lượng nút trong một MTBDD là số mũ theo số mũ, trong thực tế, số lượng nút thường là đa thức hoặc thậm chí tuyến tính. Đại diện này do đó tăng đáng kể so với DNF ngây thơ trong thực tế. Các hoạt động kết hợp các MTBD có xu hướng hiệu quả trong thực tế; hầu hết các nhu cầu xác minh chính sách đều mang lại các MTBDD với nhiều nhất là sản phẩm của số lượng nút của bản gốc.

Được phát triển theo sơ đồ Racket (PLT chính thức), Margrave [MARGRAVE] là bộ công cụ phần mềm được viết trong Đề án PLT để xác minh các yêu cầu an toàn đối với các chính sách AC được viết bằng XACML. Margrave đại diện cho các chính sách XACML dưới dạng các mô hình MTBDD, nó cho phép người dùng chỉ định các hình thức yêu cầu an toàn khác nhau trong ngôn ngữ lập trình Scheme. Margrave sử dụng một biến cho mỗi cặp giá trị thuộc tính trong chính sách XACML. Các biến cho chính sách được hiển thị trong Hình 11 sẽ tương ứng với vai trò = khoa, vai trò = sinh viên, hành động = nhận, hành động = gán và cuối cùng là tài nguyên = lớp (liên kết giữa các thuộc tính, như vai trò, với các giá trị, chẳng hạn như khoa, được nêu trong chính sách XACML). Margrave tạo các mô hình MTBDD cho các quy tắc chính sách riêng lẻ, sau đó kết hợp những thuật toán này với các thuật toán kết hợp MTBDD thực hiện các thuật toán kết hợp chính sách và quy tắc XACML.

Margrave xem các hằng số chính sách cho phép và từ chối như các quy tắc; một hoạt động được gọi là augment-rule lấy điều kiện Boolean trên các biến và quy tắc và ràng buộc quy tắc cũng yêu cầu điều kiện đã cho. Ví dụ, hai mô hình MTBDD bên trái trong Hình 12, các mô hình MTBDD cho các quy tắc nói rằng giảng viên cố gắng gán điểm sẽ mang lại giấy phép và sinh viên cố gắng gán điểm sẽ xuất hiện từ chối. Mô hình MTBDD thứ ba thể hiện kết quả của việc kết hợp hai quy tắc này bằng các thuật toán kết hợp quy tắc.



Margrave cho phép người dùng chỉ định các dạng yêu cầu an toàn khác nhau trong ngôn ngữ lập trình Scheme. API Margrave có thể xác minh các yêu cầu an toàn này và nếu có tồn tại bất kỳ mẫu phản ứng nào (là yêu cầu cụ thể) vi phạm các yêu cầu an toàn đã chỉ định, các mẫu phản ứng này được tạo ra. Margrave cũng cung cấp phân tích thay đổi toàn diện dựa trên sự khác biệt về ngữ nghĩa giữa các mô hình MTBDD đại diện cho các chính sách.

Tính mới của phương pháp tiếp cận hiện tại của Margrave nằm ở việc nắm bắt sự phong phú của logic vị ngữ bậc nhất đầy đủ để chỉ định các chính sách, hệ thống và truy vấn. Margrave tương tác với người dùng để tạo ra các kịch bản cụ thể để đáp ứng với các truy vấn của người dùng. Các truy vấn không chỉ bao gồm các yêu cầu an toàn theo kiểu xác minh thông thường, mà còn là những gì nếu các câu hỏi theo kiểu phong cách về tác động ngữ nghĩa của các chỉnh sửa. Margrave cũng cho phép người dùng chỉ định các quy tắc chính sách và yêu cầu an toàn. Sau đó, nó xử lý các truy vấn do người dùng xác định về các quy tắc chính sách và đưa ra các kịch bản dưới dạng đầu ra. Các tình huống, một cách không chính thức, các ảnh chụp nhanh của hệ thống, được điều chỉnh bởi các quy tắc chính sách, trong đó truy vấn giữ.

Margrave hỗ trợ xác minh dựa trên truy vấn và cung cấp các chế độ xem dựa trên truy vấn bằng cách tính toán các tập hợp kịch bản đầy đủ mang lại kết quả khác nhau bao gồm phân tích tác động thay đổi để so sánh một cặp chính sách. Margrave cung cấp các lợi ích của xác minh tĩnh mà không yêu cầu các tác giả viết các thuộc tính chính thức; sức mạnh của nó đến từ việc chọn một mô hình chính sách phù hợp theo logic thứ nhất và nắm lấy cả lý do tìm kiếm kịch bản và lý luận chính sách đa cấp. Nói chung, Margrave tự động xác định các công thức tương ứng với nhiều vấn đề phân tích tường lửa phổ biến, do đó cung cấp phân tích toàn diện cho các chính sách và truy vấn phong phú hơn.

## Phương pháp chính thức

Phương pháp chính thức để xác nhận các chính sách kiểm soát truy cập liên quan đến các công cụ toán học và bằng chứng cũng đã được ủng hộ. Rémi Delmas và Thomas Polacsek [AF08] đã đề xuất một khung mô hình logic để tìm ra sự không nhất quán và không đầy đủ trong các chính sách kiểm soát truy cập. Cung cấp một cơ chế để phát hiện hai thuộc tính này, họ đã giới thiệu hai thuộc tính mới: tính ứng dụng và mức tối thiểu; kỹ thuật đề xuất của họ có khả năng xác minh hai thuộc tính này. Bằng cách sử dụng các khái niệm về chữ ký, công thức và vị ngữ, họ đã xác định một số quy tắc cho khung logic, hoạt động cho dữ liệu hạn chế hoặc hữu hạn để quy tắc của chúng cũng có thể áp dụng cho dữ liệu hữu hạn. Họ cung cấp một cơ chế để phát hiện sự không nhất quán, không đầy đủ, tính ứng dụng và tính tối thiểu. [AS15] Họ cũng đề cập rằng công thức MSFOL (logic thứ tự đầu tiên được sắp xếp nhiều thứ tự) nên được chuyển đổi thành công thức logic giả Boolean để phân tích nó. Công cụ được đề xuất là một quy trình gồm ba bước trong đó một thao tác nối đất đưa ra công thức nối đất trong bước đầu tiên được chuyển đổi thành biểu thức vectơ bit bằng cách sử dụng mã hóa vectơ bit trong bước thứ hai của quy trình này. Trong bước cuối cùng của quy trình này, các biểu thức vectơ bit được chuyển đổi thành các mệnh đề ở dạng giả Boolean và cung cấp cho chúng ta công thức giả Boolean.

Z [PST96] dựa trên lý thuyết tập hợp tiên đề và logic vị ngữ thứ tự đầu tiên, có thể được sử dụng để mô tả và mô hình hóa các chính sách AC [Hu02]. Ký hiệu Z sử dụng lý thuyết tập hợp tạo cơ sở thích hợp để xây dựng mô hình AC, cho phép kiểm tra cú pháp và kiểu, mở rộng lược đồ, tính toán tiền điều kiện, kiểm tra miền và định lý chung chứng minh cho xác minh mô hình bằng cách kiểm tra miền. Nhiều nghĩa vụ chứng minh dễ dàng được chứng minh. Trong các trường hợp khó khăn hơn, việc tạo ra nghĩa vụ chứng minh thường là một trợ giúp đáng kể trong việc xác định liệu một đặc tả trong mô hình có ý nghĩa hay không.

## Kiểm tra đột biến

Các phương pháp kiểm tra hộp đen (như được mô tả trong Phần 5.1 đến 5.3) có thể không đủ để bảo vệ chống lại một số hành vi không mong muốn được nhúng trong mô hình AC như được trình bày trong 4.2. Để giải quyết điều đó, thử nghiệm đột biến của phương pháp thử nghiệm hộp trắng tạo ra các đầu vào thử nghiệm bổ sung để bao gồm các thực thể liên quan đến chính sách không được bao phủ bởi các phương thức thử nghiệm hộp đen.

Dưới đây minh họa xác minh đột biến của yêu cầu an toàn thông qua một ví dụ đơn giản. Chính sách ABAC này chính thức hóa mô hình AC của trường đại học về phân công và truy cập các lớp. Nó có hai đối tượng: giảng viên và sinh viên, hai tài nguyên: Internal\_grades và external\_grades, và ba hành động: gán, xem và nhận. Trong ví dụ này, chúng tôi hy vọng các yêu cầu an toàn sau sẽ được giữ:

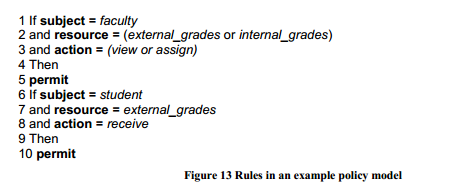
S1 - Không tồn tại thành viên của học sinh có thể gán external\_grades.

S2 - Tất cả các thành viên của khoa có thể chỉ định cả Internal\_grades và external\_grades.

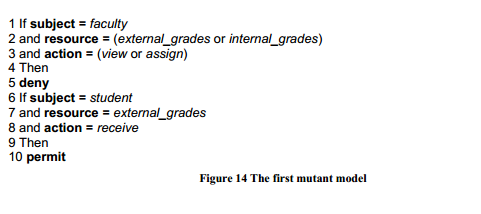
S3 - Không tồn tại sự kết hợp của các chủ đề sao cho người dùng với các chủ thể đó có thể vừa nhận và gán tài nguyên bên ngoài.

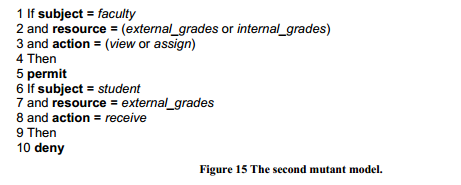
 Yêu cầu an toàn S1 là trực quan vì chúng tôi chắc chắn không muốn học sinh chỉ định điểm. Yêu cầu an toàn S2 là đảm bảo rằng các giảng viên thực sự có thể chỉ định điểm. S3 là một ví dụ về phân chia nhiệm vụ vì chúng tôi không muốn bất kỳ ai chỉ định điểm số của riêng mình, một xung đột lợi ích rõ ràng.

Hình 13 cho thấy mô hình chính sách AC mẫu. Để giữ cho ví dụ có thể đọc và súc tích, mô hình được viết dưới dạng các câu lệnh If Then đơn giản, không đáp ứng ngay ba yêu cầu an toàn này. Đây thực chất là một ràng buộc tách rời nhiệm vụ (SOD) nhằm hạn chế mọi yêu cầu có cả giảng viên và sinh viên. Tất cả ba yêu cầu an toàn giữ trong mô hình.



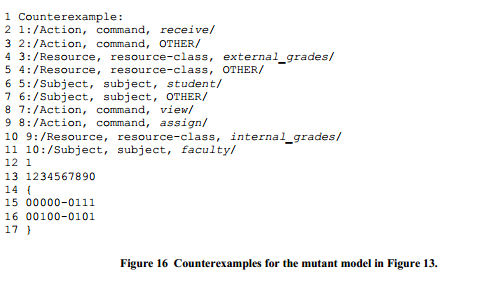
Bước đầu tiên của xác minh đột biến là tạo ra các mô hình đột biến bằng cách sử dụng toán tử đột biến chỉ đơn giản là đảo ngược từng hiệu ứng quy tắc bằng cách thay đổi giấy phép để từ chối hoặc từ chối cấp phép (mỗi lần một mô hình cho mỗi mô hình đột biến). Số lượng mô hình đột biến được tạo bởi toán tử này bằng với số quy tắc trong mô hình. Mô hình ví dụ chỉ có hai quy tắc và do đó chỉ có hai mô hình đột biến được tạo. Hình 14 và 15 cho thấy hai mô hình đột biến này.





Bước thứ hai của xác minh đột biến là xác định các yêu cầu an toàn nào cho mô hình ban đầu và từng mô hình đột biến. Kết quả thay đổi được cho là bị giết bởi yêu cầu an toàn nếu yêu cầu an toàn chứa mô hình ban đầu nhưng không chứa cho mô hình đột biến. Nói cách khác, yêu cầu an toàn cho thấy lỗi tồn tại trong mô hình đột biến. Số lượng đột biến giảm càng nhiều, mô hình ban đầu càng toàn diện trong việc đáp ứng các yêu cầu an toàn.

Mô hình đột biến đầu tiên trong Hình 14 không thỏa mãn S2 và do đó, đột biến đầu tiên không tồn tại. Nhớ lại S2 tìm cách đảm bảo chính sách mà tất cả các giảng viên có thể chỉ định điểm. Vì lỗi trong Hình 14 chính xác là quy tắc cấp quyền truy cập này, nên yêu cầu an toàn rõ ràng đã bị vi phạm. Hình 16 minh họa đầu ra từ xác minh mô hình hộp đen bằng cách sử dụng Margrave (Phần 5.2) trên mô hình đột biến đầu tiên. Mỗi ví dụ mẫu (tức là, yêu cầu) được biểu diễn dưới dạng mặt nạ bit trong đó mỗi bit tương ứng với id thuộc tính cụ thể trên Dòng 2 Biệt11. Nếu bit bằng 0, thì giá trị thuộc tính tương ứng không xuất hiện trong khi nếu bit là 1 thì giá trị thuộc tính tương ứng sẽ xuất hiện. Như mong đợi, các mẫu cụ thể đã cho là dành cho một khoa để gán Internal\_grades và cho một khoa để gán bên ngoài\_grades. Hai mẫu tương ứng tương ứng với dòng 15 và 16. Quyền truy cập bị từ chối cho cả hai yêu cầu, cho thấy vi phạm yêu cầu an toàn S2.



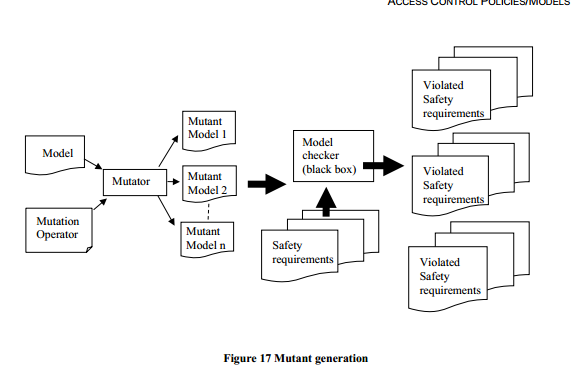
Mô hình đột biến thứ hai trong Hình 15 không bị giết bởi bất kỳ một trong ba yêu cầu an toàn nào, phản ánh rằng mô hình ban đầu không toàn diện và không hoàn toàn bao gồm các yêu cầu an toàn.

Xác minh đột biến phục vụ yêu cầu an toàn với hai mục đích: (1) để định lượng mức độ triệt để của mô hình bao gồm các yêu cầu an toàn (2) để tạo điều kiện thay đổi mô hình sao cho mô hình đáp ứng tất cả các yêu cầu an toàn hoặc ngược lại.

Trong một số trường hợp, thay vì sửa một mô hình, xác minh đột biến phục vụ mục đích vá các yêu cầu an toàn, do đó, sửa lỗi chính sách. Ví dụ: việc thêm yêu cầu an toàn S4 như dưới đây không thể áp dụng cho chính sách đột biến thứ hai trong xx4.

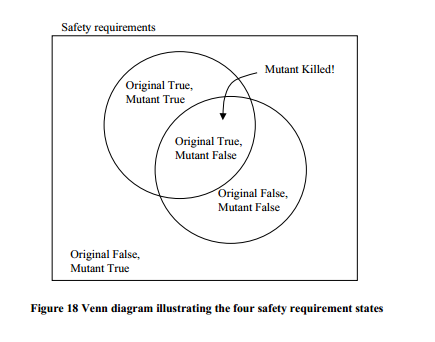
S4 Tất cả các thành viên của sinh viên có thể nhận được bên ngoài.

Tóm lại, Hình 17 minh họa các đầu vào cần thiết và đầu ra kết quả của xác minh đột biến. Các đầu vào là mô hình được thử nghiệm và, trong trường hợp này, một toán tử đột biến đơn. Bộ biến đổi sau đó tạo ra một tập hợp các mô hình đột biến, mỗi mô hình có một lỗi duy nhất. Toán tử đột biến tạo ra một đột biến cho mỗi quy tắc bằng cách phủ định quyết định của quy tắc đó. Mặc dù xác minh hộp đen thực hiện tương đối nhanh chóng cho các đột biến, các mô hình lớn có thể được sử dụng để dễ dàng tạo ra hàng ngàn mô hình đột biến. Một đột biến tương đương là một đột biến khác về mặt cú pháp so với mô hình ban đầu trong khi tương đương về mặt ngữ nghĩa. Nói cách khác, một đột biến tương đương sẽ tạo ra kết quả giống như mô hình ban đầu cho tất cả các yếu tố đầu vào và do đó không mang lại lợi ích và dẫn đến việc hạ thấp tỷ lệ giết chết đột biến, đưa ra phép đo chất lượng được đánh giá và không chính xác. Chúng tôi cũng có thể xác định các yêu cầu an toàn nào được giữ và không áp dụng cho cả mô hình ban đầu và từng mô hình đột biến. Lưu ý rằng để thực hiện xác minh theo chương trình, một chương trình Scheme và chương trình thực thi cho mô hình ban đầu và mỗi mô hình đột biến có thể được yêu cầu [MX07].



Một bước quan trọng của thử nghiệm đột biến là tính toán tỷ lệ giết đột biến. Tỷ lệ giết người đột biến là tỷ lệ số lượng người đột biến bị giết trên tổng số người đột biến. Tỷ lệ này đóng vai trò là một số liệu để định lượng mức độ bao phủ của các yêu cầu an toàn theo mô hình. Tỷ lệ tiêu diệt đột biến cao cho thấy mô hình ban đầu đáp ứng nhiều yêu cầu an toàn. Ví dụ, phạm vi bảo hiểm đột biến (nghĩa là tỷ lệ giết người đột biến) cho các yêu cầu an toàn S1, S2 và S3 trong ví dụ trên là 50% do chỉ một trong hai đột biến bị giết. Nếu S4 được thêm vào, thì tỷ lệ giết người đột biến sẽ tăng lên 100%.

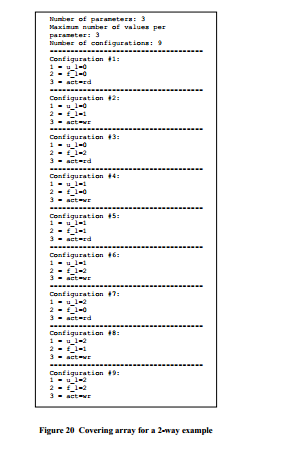
Các tệp theo dõi được tạo bởi xác minh yêu cầu an toàn được mô tả trước đó được phân tích cú pháp để phân chia yêu cầu an toàn được đặt thành bốn tập hợp con cho mỗi đột biến. Một sơ đồ Venn được minh họa trong Hình 18 mô tả mối quan hệ của bốn bộ này cho một mô hình đột biến duy nhất. Khu vực bên trong hộp đại diện cho tập hợp tất cả các yêu cầu an toàn. Khu vực bên trong vòng tròn ngoài cùng bên trái đại diện cho tập hợp các yêu cầu an toàn đúng với mô hình ban đầu. Do đó, khu vực bên ngoài vòng tròn bên trái nhất và bên trong hộp là tập hợp các yêu cầu an toàn không đúng với mô hình ban đầu (nghĩa là, các yêu cầu an toàn này không được giữ bởi mô hình ban đầu). Khu vực bên trong vòng tròn bên phải nhất đại diện cho mô hình đột biến giữ sai cho yêu cầu an toàn. Do đó, khu vực bên ngoài vòng tròn bên phải nhất và bên trong hộp thể hiện tập hợp các yêu cầu an toàn được giữ đúng cho mô hình đột biến. Khu vực quan tâm là giao điểm của hai vòng tròn. Nếu ít nhất một yêu cầu an toàn được giữ đúng cho mô hình ban đầu nhưng không đúng với mô hình đột biến, thì người đột biến bị giết. Nếu hai vòng tròn không giao nhau, thì người đột biến không bị giết. Yêu cầu an toàn được giữ đúng cho cả mô hình ban đầu và mô hình đột biến không có giá trị trong việc phơi bày lỗi trong mô hình đột biến vì yêu cầu an toàn không áp dụng cho phần của mô hình có lỗi. Yêu cầu an toàn được giữ sai cho mô hình ban đầu không có giá trị vì không rõ ràng nếu sự an toàn sai này yêu cầu được gây ra bởi một lỗi trong mô hình hoặc chính yêu cầu an toàn. Cụ thể hơn, trước khi tiến hành xác minh đột biến, các yêu cầu an toàn này phải được kiểm tra thủ công để xác định xem chúng có bị lỗi do lỗi trong mô hình không, lỗi trong yêu cầu an toàn hoặc lỗi trong các ràng buộc môi trường.



## Kiểm tra kết hợp tự động

Như được mô tả trong 4.3, một nhà tiên tri thử nghiệm được yêu cầu để kiểm tra việc thực hiện đúng mô hình AC. Tuy nhiên, một hệ thống AC trong thế giới thực có thể có quá nhiều biến số (chủ thể, hành động, đối tượng, điều kiện, v.v.) để tạo ra các trường hợp thử nghiệm toàn diện dựa trên các yêu cầu an toàn. Một trong những giải pháp khả thi để giảm kích thước do đó có hiệu quả trong tính toán để tạo ra phép thử là áp dụng công nghệ Kiểm tra kết hợp tự động cho phần mềm (ACTS) [ACTS], một phương pháp kiểm tra tất cả các kết hợp các giá trị tham số đầu vào. Hình thức phổ biến nhất là thử nghiệm theo cặp, trong đó tất cả các cặp giá trị đầu vào được bao phủ trong ít nhất một thử nghiệm. Các phiên bản cường độ cao hơn của phương pháp này bao gồm các tương tác 3 chiều, 4 chiều hoặc nhiều hơn ít nhất một lần. Ưu điểm của kiểm tra tổ hợp để xác minh thực hiện AC là AC thường dựa vào một số lượng nhỏ các giá trị riêng biệt cho hầu hết các tham số. Ví dụ: chính sách bảo mật đa cấp (nghĩa là chính sách phân loại quân sự tiêu chuẩn) có thể có các cấp độ chưa được phân loại, bí mật, bí mật, tối mật, cộng với một số lượng nhỏ các danh mục, tất cả được áp dụng cho một bộ sưu tập tài nguyên như tệp và chương trình. Để trình diễn, một mô hình đa cấp được đơn giản hóa, trong đó mỗi đối tượng (người dùng) có mức giải phóng mặt bằng u\_l và mỗi tệp có một mức phân loại f\_l. Các cấp được đưa ra là 0, 1 hoặc 2, có thể đại diện cho các cấp như Bí mật, Bí mật và Tối mật. Một người dùng u có thể đọc một tệp f nếu u\_l ≥ f\_l (quy tắc không đọc lên quy tắc) hoặc ghi vào một tệp nếu f\_l ≥ u\_l (quy tắc không ghi xuống dòng) như sau.

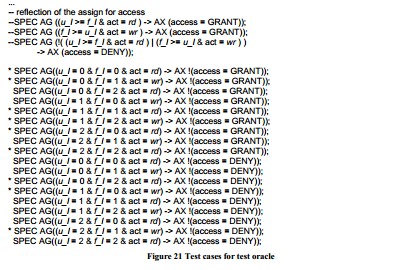
Một mảng bao phủ 2 chiều cho ví dụ được liệt kê trong Hình 20. Nếu chúng ta có số lượng tham số lớn hơn, chúng ta có thể tạo ra các cấu hình thử nghiệm bao gồm tất cả các kết hợp 3 chiều, 4 chiều, v.v. (Chỉ với ba tham số, tương tác 3 chiều tương đương với thử nghiệm toàn diện.)



Bước tiếp theo là gán các giá trị từ mảng che phủ cho các tham số được sử dụng trong mô hình. Đối với mỗi thử nghiệm, chúng tôi tuyên bố rằng kết quả dự kiến sẽ không xảy ra. Trình kiểm tra mô hình xác định các kết hợp sẽ từ chối các khiếu nại này, xuất ra các kết hợp này dưới dạng phản hồi mẫu. Mỗi mẫu thử sau đó có thể được chuyển đổi thành một thử nghiệm với kết quả dự kiến đã biết. Như có thể thấy bên dưới, với mỗi trong số 9 cấu hình trong mảng bao phủ của Hình 20, chúng tôi tạo ra một yêu cầu ĐẶC BIỆT của biểu mẫu:

ĐẶC BIỆT AG ((<bao gồm các giá trị mảng>) -> AX! (Access = <result>))

(SPEC là cú pháp của trình kiểm tra mô hình cho đặc tả yêu cầu an toàn); Quá trình này được lặp lại cho từng kết quả có thể xảy ra, trong trường hợp này là hoặc GRANT, hoặc DEN DENY, vì vậy chúng tôi có 9 yêu cầu cho mỗi hai kết quả như trong Hình 21.



Trình kiểm tra mô hình tạo ra các mẫu phản ứng trong đó các giá trị đầu vào sẽ từ chối các khiếu nại được chỉ định trong Hình 21. Mỗi mẫu này là một tập hợp dữ liệu thử nghiệm có kết quả mong đợi là GRANT hoặc DENY (như được liệt kê bởi \* trong Hình 21). Bây giờ chúng tôi loại bỏ các tên và giá trị tham số, từ Hình 21 đưa ra các thử nghiệm có thể được áp dụng cho hệ thống được thử nghiệm. Các thử nghiệm được tạo ra được hiển thị dưới đây:

*u\_l* = 0 & *f\_l* = 0 & act = *rd* -> access = GRANT  
 *u\_l* = 0 & *f\_l* = 1 & act = *wr* -> access = GRANT  
 *u\_l* = 1 & *f\_l* = 1 & act = *rd* -> access = GRANT  
 *u\_l* = 1 & *f\_l* = 2 & act = *wr* -> access = GRANT  
 *u\_l* = 2 & *f\_l* = 0 & act = *rd* -> access = GRANT

*u\_l* = 2 & *f\_l* = 2 & act = *rd* -> access = GRANT  
 *u\_l* = 0 & *f\_l* = 2 & act = *rd* -> access = DENY  
 *u\_l* = 1 & *f\_l* = 0 & act = *wr* -> access = DENY  
 *u\_l* = 2 & *f\_l* = 1 & act = *wr* -> access = DENY

Các định nghĩa kiểm tra này hiện có thể được xử lý hậu kỳ bằng cách sử dụng các tập lệnh đơn giản để tạo ra các trường hợp kiểm thử cho lời tiên tri kiểm tra.

## Thử nghiệm giả

Phần này mô tả phương pháp kiểm tra các hệ thống kiểm soát truy cập (hoặc các hệ thống khác có quy tắc được chỉ định chính thức) giả mạo, mà chúng tôi xác định là kiểm tra toàn diện tất cả các kết hợp các giá trị thuộc tính mà quyết định kiểm soát truy cập phụ thuộc vào [KHFKL16]. Một lợi thế của phương pháp này là nó có thể được sử dụng để tạo ra một bộ thử nghiệm hoàn chỉnh, theo nghĩa là tất cả các trường hợp tiêu cực cũng như tất cả các trường hợp dương tính đều được xác minh. Cách tiếp cận này tương tự như các phương pháp giả toàn diện để kiểm tra các mạch tổ hợp, trong đó vấn đề xác minh được giảm bớt bằng cách chỉ kiểm tra toàn bộ tập hợp con của đầu vào mà đầu ra phụ thuộc hoặc phân vùng mạch và kiểm tra toàn bộ từng phân đoạn. Chúng ta có thể sử dụng nguyên tắc cơ bản chỉ kiểm tra các tập hợp con của các thuộc tính mà quyết định phụ thuộc, mặc dù việc phân vùng được thực hiện theo cách khác so với các mạch tổ hợp. Cấu trúc của vấn đề kiểm soát truy cập cho phép áp dụng cùng một nguyên tắc bằng cách đưa ra các điều kiện cho mỗi cấp dưới dạng bình thường rời rạc, sau đó xem xét từng thuật ngữ riêng biệt.

Ví dụ: quy tắc có các thuộc tính Employ\_status và time\_of\_day có thể là, Nếu Nếu đối tượng là nhân viên và giờ là từ 9 giờ sáng đến 5 giờ chiều, thì hãy cho phép nhập cảnh. Vấn đề với cách tiếp cận này là các thuộc tính hoặc biến boolean dẫn đến khả năng Quy tắc 2n. Nhiều quy tắc như vậy có thể được bao gồm trong các tài liệu chính sách bằng văn bản và các quy tắc có thể bao gồm nhiều thuộc tính khác nhau. Đối với bất kỳ sự kết hợp nào của các giá trị thuộc tính, hệ thống phải thực hiện các quy tắc phản ánh chính xác chính sách bằng văn bản. Cấu trúc của các quy tắc như vậy thường như sau, trong đó Ri là các điều kiện boolean đánh giá các giá trị của một hoặc nhiều thuộc tính:

*R*1 → *grant  
R*2 → *grant  
…Rm*→ *grant  
else* → *deny*

*tương đương với:*

*R*1 → *grant  
R*2 → *grant  
…Rm*→ *grant*(*~R*1) (*~R*2)… (*~Rm*) → *deny*

*Ví dụ: Giả sử chúng ta có một quy tắc truy cập như dưới đây:*

if (a && (c && !d ||e)) grant();  
else if (!a && b && !c) grant();  
else deny();

Mã này có thể được ánh xạ tới biểu thức sau:

(*a*(*cd ̅* +*e*) → *grant*)  
(*a ̅ bc ̅* → *grant*)  
((∼(*a*(*cd ̅* +*e*)))(∼(*a ̅ bc ̅* )) → *deny*)

Các chữ boolean có thể đại diện cho các điều kiện, chẳng hạn như tuổi> 18 hoặc các thuộc tính boolean như nhân viên, nhưng cấu trúc sẽ như trong ví dụ. Đó là, một loạt các biểu thức chỉ định các tập hợp con của các điều kiện thuộc tính phải đúng với quyền truy cập được cấp, theo sau là quy tắc từ chối truy cập mặc định khi không có biểu thức thuộc tính nào được khởi tạo thành true.

Kiểm tra một triển khai yêu cầu cho thấy chính sách được chỉ định, P, được thực hiện chính xác. Chính sách được triển khai P ức phải được hiển thị để tạo ra phản hồi giống như P cho mọi kết hợp thuộc tính được sử dụng làm đầu vào. Nghĩa là, đối với các thuộc tính đầu vào x1, Vượt, xn, Pạn (x1, Mạnh, xn) = P (x1, Lỗi, xn).

Xác nhận rằng quyền truy cập sẽ được cấp cho người dùng với các thuộc tính phù hợp rất dễ dàng: chúng tôi chỉ cần đọc các điều kiện thuộc tính cho từng biểu thức cấp và xác minh rằng hệ thống kiểm soát truy cập trả lại ủy quyền trong từng trường hợp. Số lượng các bài kiểm tra như vậy là tuyến tính trong số các điều kiện cấp. Tuy nhiên, khó khăn hơn nhiều để đảm bảo rằng không có sự kết hợp các thuộc tính không hợp lệ sẽ dẫn đến ủy quyền. Với n thuộc tính hoặc biến boolean, có 2n tổ hợp thuộc tính có thể. Ví dụ, sẽ không có gì bất thường khi có 50 điều kiện hoặc thuộc tính Boolean, dẫn đến kết hợp 250 ≈ 1015, nhưng phải chứng minh rằng không có sự kết hợp nào sẽ cho phép truy cập không đúng cách.

Để làm cho việc kiểm tra có thể thực hiện được, chúng tôi sẽ sử dụng các mảng thuộc tính trong các chính sách đã được chuyển đổi thành dạng k-DNF. k-DNF đề cập đến hình thức bình thường rời rạc trong đó không có thuật ngữ nào chứa nhiều hơn k chữ. Hãy nhớ lại rằng một thuật ngữ là sự kết hợp của một hoặc nhiều nghĩa đen trong phạm vi. Ví dụ: abc + de chứa hai thuật ngữ, một có ba chữ và một có hai, vì vậy biểu thức ở dạng 3-DNF. Lớp phủ 3 chiều sẽ chứa tất cả các kết hợp 3 chiều của các giá trị biến (và do đó cũng là tất cả các kết hợp 1 chiều và 2 chiều). Khi một biểu thức nằm trong k-DNF, bất kỳ thuật ngữ nào chứa k chữ được phân giải thành true sẽ rõ ràng dẫn đến biểu thức đầy đủ được đánh giá là đúng. Ví dụ: quy tắc ở dạng 2-DNF có thể là: Kiếm nếu nhân viên

&& US\_citizen | | kiểm toán viên sau đó cấp cho. Quy tắc này chứa một thuật ngữ của hai thuộc tính và một thuật ngữ của một thuộc tính, vì vậy nó là 2-DNF. Bởi vì một mảng bao gồm sức mạnh k chứa mọi cài đặt có thể có của tất cả các k-tuples và i-tuples cho i <k, nên nó chứa mọi kết hợp các giá trị của bất kỳ chữ k nào.

Bao gồm các công cụ tạo mảng, chẳng hạn như ACTS, có thể bao gồm các ràng buộc ngăn chặn các kết hợp biến đáp ứng các tiêu chí được chỉ định trong cú pháp kiểu logic thứ tự đầu tiên. Ví dụ: nếu chúng tôi đang thử nghiệm các ứng dụng chạy trên nhiều tổ hợp hệ điều hành và trình duyệt khác nhau, chúng tôi có thể bao gồm một ràng buộc, chẳng hạn như “OS =, Linux Linux => browser! Các ràng buộc thường được sử dụng trong các tình huống như thế này, trong đó các kết hợp nhất định không xảy ra trong thực tế và do đó không nên đưa vào các thử nghiệm.

hương pháp: Đặt R = tiền đề quy tắc (phía bên trái của quy tắc hàm ý, chẳng hạn như p trong p → q) của một hoặc nhiều quy tắc chính sách đang được thử nghiệm trong k-DNF và Ti là các thuật ngữ (kết hợp của một hoặc nhiều thuộc tính) trong R Đối với ví dụ có trong phần giới thiệu, các thuật ngữ Ti của R sẽ là acd̅, ae và a̅bc̅. R không nhất thiết là chính sách hoàn chỉnh; ví dụ, đó có thể là tập hợp các quy tắc được liên kết với một tài nguyên cụ thể mà chúng tôi muốn kiểm tra.

Thử nghiệm tích cực: Tạo GTEST cho bộ thử nghiệm mà mọi thử nghiệm sẽ tạo ra phản hồi cấp. Nó phải được chỉ ra rằng đối với tất cả các đầu vào có thể, trong đó một số kết hợp của các giá trị đầu vào k khớp với một điều kiện cấp, một quyết định cấp được trả lại. Xây dựng bộ kiểm tra GTEST với một thử nghiệm cho mỗi thuật ngữ của R như sau



Việc xây dựng đảm bảo rằng mỗi thuật ngữ trong P được xác minh để độc lập tạo ra phản hồi cấp. Việc phủ định từng thuật ngữ Tj, i ≠ j, ngăn chặn việc che giấu lỗi khi có các kết hợp khác sẽ trả lại kết quả tương tự. Ví dụ: nếu một điều kiện quy tắc là ab + cd → cấp, thì các đầu vào của 1100, 1101, 1110 có thể được sử dụng để kiểm tra ab → cấp. Tuy nhiên, đầu vào 1111 sẽ không phát hiện ra lỗi nếu hệ thống bỏ qua biến a hoặc b, bởi vì điều kiện cd sẽ gây ra quyết định cấp và không có biến vị ngữ cấp nào khác được đánh giá. Một thử nghiệm như vậy là bắt buộc đối với mỗi thuật ngữ trong quy tắc cấp, do đó, đối với quy tắc m có trung bình p điều khoản mỗi loại, số lượng thử nghiệm được yêu cầu tỷ lệ thuận với mp.

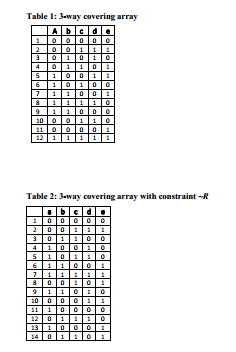
Thử nghiệm tiêu cực: Tạo một bộ thử nghiệm DTEST mà mọi thử nghiệm sẽ tạo ra phản hồi từ chối. Nó phải được chỉ ra rằng đối với tất cả các đầu vào có thể, trong đó không có sự kết hợp của các giá trị đầu vào k phù hợp với điều kiện cấp, một quyết định từ chối được trả về.

DTEST = bao gồm mảng cường độ k, cho tập hợp các thuộc tính có trong R, với các ràng buộc được chỉ định bởi ~ R.

Lưu ý rằng cấu trúc của đánh giá quy tắc kiểm soát truy cập cho phép sử dụng mảng che phủ cho DTEST, nén một số lượng lớn các điều kiện kiểm tra vào một vài thử nghiệm. Bởi vì việc từ chối chỉ được ban hành sau khi tất cả các điều kiện cấp đã được đánh giá, việc che giấu một kết hợp này bằng một kết hợp khác chỉ có thể xảy ra đối với DTEST khi thử nghiệm tạo ra phản hồi của cấp. Trong trường hợp như vậy, một lỗi đã được phát hiện, có thể được sửa chữa trước khi chạy lại bộ kiểm tra. Vì DTEST là một mảng bao phủ, nên số lượng thử nghiệm sẽ tỷ lệ thuận với vk log n, cho các giá trị v trên mỗi thuộc tính (thông thường v = 2 vì hầu hết sẽ là các điều kiện boolean) và n thuộc tính. Đối với quy tắc m, số lượng kiểm tra được nhân với hằng số m.

Ví dụ: Bảng 1 đưa ra một tập hợp các thuộc tính boolean từ a đến e, trong đó mỗi hàng xác định các giá trị cho các thuộc tính xác định quyết định, cấp hoặc từ chối. Do đó, một mảng bao phủ cho tiền đề R của quy tắc trong 3-DNF như (acd ̅ + a bc ̅ → Grant) được đưa ra trong Bảng 1. Tổng số kết hợp 3 chiều được bao phủ là số lượng cài đặt của ba biến nhị phân nhân với số cách chọn ba biến từ năm, tức là 23 5 3 = 80.

Bảng 2 cho thấy một mảng bao phủ cho tập hợp các biến được tạo bằng cách sử dụng ~ R làm ràng buộc. Đó là, hai thuật ngữ của quy tắc, acd và ̅ bc, đã được loại trừ khỏi mảng, nhưng tất cả các kết hợp 1-, 2- và 3 chiều khác có thể được tìm thấy trong mảng. Vì acd và ̅ bc ̅ là các điều kiện duy nhất theo đó quyền truy cập nên được cấp, nên mảng trong Bảng 2 sẽ dẫn đến phản hồi từ chối từ hệ thống kiểm soát truy cập cho mọi thử nghiệm. Nói chung, các bài kiểm tra bao gồm tất cả 78 cài đặt thuộc tính 3 chiều sẽ không thể khởi tạo quy tắc kiểm soát truy cập thành đúng.



Thuộc tính phát hiện lỗi: Chung, các thử nghiệm từ GTEST và DTEST sẽ phát hiện tất cả các lỗi được thêm, xóa hoặc thay đổi với tối đa k chữ. Nếu nhiều hơn k chữ được bao gồm trong thuật ngữ đã thay đổi, một số lỗi vẫn được phát hiện. Bằng chứng về các tính chất này có thể được tìm thấy trong [IWCT 16], trong đó cũng nêu chi tiết các đặc điểm của lỗi hơn k chữ sẽ được phát hiện.

Khái quát hóa phương pháp: Mặc dù chỉ có hai câu trả lời được xem xét trong phần mô tả phương pháp ở trên, nhưng thật đơn giản để khái quát thành nhiều hơn hai câu trả lời. Vì có thể có nhiều phản hồi, thay vì Gtest và Dtest để cấp và từ chối kết quả, chúng tôi sẽ đề cập đến mảng Ptesti và Ntesti, cho các kết quả có thể có của tôi và các hàng chỉ số Ptestij và Ntestij j của mỗi mảng.

• Thử nghiệm tích cực: Tạo một bộ thử nghiệm Ptesti mà mọi thử nghiệm sẽ tạo ra đầu ra phản hồi của Oi. Nó phải được chỉ ra rằng đối với tất cả các đầu vào có thể, trong đó một số kết hợp của k

các giá trị đầu vào khởi tạo điều kiện cho Oi thành true, kết quả của Oi được trả về. Cấu trúc cho Gtest được sử dụng cho mỗi đầu ra phản hồi có thể Oi để tạo ra một mảng dương cho mỗi Oi phản hồi.

• Kiểm tra tiêu cực: Tương tự, chúng tôi xây dựng một mảng âm Ntesti cho mỗi đầu ra kết quả có thể là một mảng cường độ k, cho các thuật ngữ trong quy tắc Ri dẫn đến Oi với các ràng buộc được chỉ định bởi ~ Ri, cho mỗi đầu ra i.

Cũng lưu ý rằng phương thức này không bị hạn chế đối với các ứng dụng điều khiển truy cập, nhưng có thể được sử dụng cho bất kỳ hệ thống được chỉ định chính thức nào, nơi các quy tắc có thể được chuyển đổi thành dạng k-DNF.

## Phát hiện thực thi quy tắc AC thời gian thực

Hầu hết các nghiên cứu về mô hình AC hoặc kỹ thuật xác minh chính sách đều tập trung vào một mô hình cụ thể và hầu hết tất cả các nghiên cứu đều nằm trong các phương pháp được áp dụng, yêu cầu các chính sách AC đã hoàn thành làm đầu vào cho quá trình xác minh hoặc thử nghiệm để tạo báo cáo lỗi (như đã đề cập trước đó) . Mặc dù đã đạt được xác minh chính xác và các mẫu phản biện có thể được tạo ra cùng với các lỗi được tìm thấy, các phương pháp đó không cung cấp thông tin về nguồn lỗi quy tắc có thể cho phép xung đột trong chuyển nhượng đặc quyền, rò rỉ đặc quyền hoặc xung đột quyền lợi. Khó khăn trong việc tìm ra nguồn lỗi được tăng lên đặc biệt là khi các quy tắc AC phức tạp bao trùm các biến trùng lặp đến một mức độ phức tạp. Sự phức tạp là do thực tế là một lỗi có thể không được gây ra bởi một quy tắc cụ thể; ví dụ: quy tắc x cấp quyền truy cập của chủ thể / thuộc tính cho đối tượng / thuộc tính o và quy tắc y từ chối chủ đề / thuộc tính nhóm g, s là thành viên của, truy cập vào đối tượng o. Xung đột như vậy chỉ có thể được giải quyết bằng cách xóa quy tắc x hoặc y hoặc thành viên g của s khỏi chính sách. Nhưng việc xóa x hoặc y ảnh hưởng đến các quy tắc khác phụ thuộc vào chúng (ví dụ: thành viên của nhóm chủ đề g k được cấp quyền truy cập vào đối tượng o) và xóa tư cách thành viên của s trong g sẽ vô hiệu hóa quyền truy cập hợp pháp vào các đối tượng / thuộc tính khác thông qua tư cách thành viên. Do đó, nó yêu cầu phân tích thủ công từng quy tắc trong chính sách để tìm giải pháp chính xác cho lỗi.

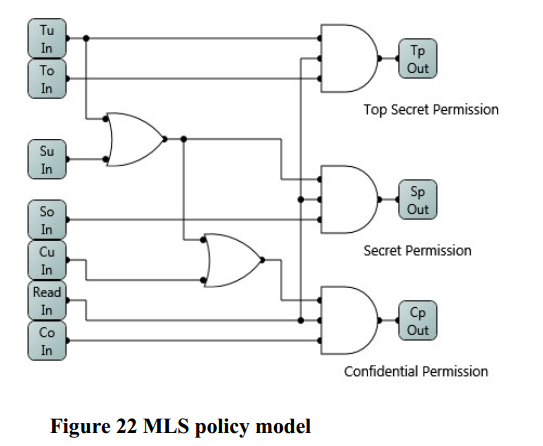
Để giải quyết vấn đề, kỹ thuật Mô phỏng mạch logic quy tắc AC (ACRLCS) [HS13], cho phép các tác giả AC phát hiện ra lỗi khi quy tắc AC gây ra lỗi được thêm vào mô hình, do đó việc khắc phục có thể được thực hiện trong thời gian thực. (tại chỗ) trước khi thêm các quy tắc khác làm phức tạp thêm nỗ lực phát hiện. Nói cách khác, thay vì kiểm tra bằng cách truy nguyên các mối tương quan giữa các quy tắc sau khi chính sách được hoàn thành, tác giả chính sách chỉ cần kiểm tra quy tắc mới được thêm vào so với các quy tắc chính xác trước đó. Trong ACRLCS, quy tắc AC được biểu diễn trong mạch logic mô phỏng (SLC) (phát âm là CELL-see). Bằng cách mô phỏng, điều đó có nghĩa là ACRLCS không nhất thiết phải được thực hiện bằng một mạch điện tử vật lý; tuy nhiên, khái niệm này có thể được thực hiện và tính toán thông qua phần mềm mô phỏng.

ACRLCS bao gồm các SLC đại diện cho các quy tắc AC được chỉ định trong các biểu thức Boolean. Một SLC sẽ có thể duy trì việc gán các biến AC và phân cấp đặc quyền (thông qua kế thừa) và đánh giá quyền truy cập (ví dụ: cấp hoặc từ chối) từ quy tắc được thực hiện. Với nguyên tắc này, kỹ thuật bao gồm hai quy trình chính:

Xây dựng SLC dựa trên các biến AC được chỉ định bởi các biểu thức Boolean hoặc phân cấp quan hệ được chỉ định bởi các quan hệ trong quy tắc AC. Trong SLC, mỗi biến AC được biểu thị bằng một công tắc đầu vào và toán tử logic quy tắc và quan hệ phân cấp được mô phỏng bằng các cổng logic.

Phát triển một thuật toán hiệu quả để phát hiện các lỗi quy tắc trong chính sách bằng cách kích hoạt các công tắc đầu vào đại diện cho các biến quy tắc AC được xác minh trong SLC hiện được xây dựng. Lỗi quy tắc được tạo dưới dạng đầu ra tín hiệu tích cực từ SLC, biểu thị xung đột trong các nhiệm vụ đặc quyền gây ra bởi quy tắc mới được thêm vào.

Hình 22 triển khai thuộc tính đọc Bell-Lapadula [BL73] đơn giản của mô hình chính sáchMLS, theo đó người dùng xếp hạng Top\_Secret có thể đọc các đối tượng trong các cấp bậc Top\_Secret, Bí mật và Bí mật; Người dùng xếp hạng bí mật có thể đọc các đối tượng trong hàng ngũ Bí mật và Bí mật; và người dùng xếp hạng Bí mật chỉ có thể đọc các đối tượng trong xếp hạng Bí mật.



## Đề án thử nghiệm

Khung kiểm tra hệ thống AC chung (như trong Hình 23) chứa bốn chức năng chính sử dụng các phương thức như đã nêu trong Phần 4. Trình phát hiện lỗi thời gian thực quy tắc AC được sử dụng tùy ý để thiết kế các mô hình AC ban đầu [HS13]. Black Box Tester kiểm tra xem một mô hình (nguyên bản hay đột biến) giữ cho các Yêu cầu an toàn được chỉ định. Black Box Tester (kết quả mẫu thử) cung cấp thông tin cho sửa lỗi mô hình ban đầu (một hành động của con người như đường chấm chấm trong hình) và kiểm tra tiêu diệt đột biến cho White Box Tester. Nó cũng lấy đầu ra từ Test Generator và trả về kết quả để tạo trường hợp thử nghiệm. White Box Tester tạo và giết chết các mô hình đột biến dựa trên mô hình ban đầu và các yêu cầu an toàn; các mô hình đột biến của nó được gửi đến Black Box Tester để kiểm tra tiêu diệt đột biến hoặc cho tác giả mô hình AC để tìm mô hình ban đầu hoặc sửa chữa yêu cầu an toàn (một hành động của con người như thể hiện trong các đường chấm trong hình). Trình tạo Oracle thử nghiệm tạo ra các trường hợp thử nghiệm dựa trên Yêu cầu an toàn và kết quả mẫu thử nghiệm của Hộp đen. Các bước quy trình được liệt kê dưới đây:

1. (tùy chọn) Các mô hình AC được thiết kế dựa trên chính sách AC bằng cách sử dụng Trình kiểm tra lỗi thời gian thực của Quy tắc AC (như được mô tả trong Phần 5.4).

2. Yêu cầu an toàn được chỉ định.

3. Mô hình AC gốc đã hoàn thành được kiểm tra theo các yêu cầu an toàn bằng Black Box Tester (như được mô tả trong Phần 3.1, 4.1 và 5.1). Nếu tìm thấy lỗi, mô hình ban đầu cần được sửa chữa (do đó lặp lại các bước từ 1 đến 3), nếu không thì chuyển sang bước 4 tiếp theo.

4. Đã sửa lỗi mô hình ban đầu được gửi đến White Box Tester để kiểm tra phạm vi và kiểm tra giam cầm (như được mô tả trong Phần 3.2, 4.2 và 5.2). White Box Tester sử dụng Black Box Tester để quyết định xem các mô hình đột biến được tạo ra có bị giết hay không. Nếu không, mô hình ban đầu hoặc các yêu cầu an toàn cần phải được sửa chữa (do đó lặp lại bước 1 đến 4), nếu không, hãy chuyển sang bước 5 tiếp theo.

5. Kiểm tra Trình tạo Oracle tạo các trường hợp kiểm tra dựa trên Yêu cầu an toàn, được gửi đến Black Box Tester để tạo kết quả cấp phép được sử dụng cho lời tiên tri kiểm tra (như được mô tả trong Phần 4.3 và 5.3).

Lưu ý rằng các thành phần trong Hình 23 và các bước được liệt kê ở trên không nhất thiết phải được yêu cầu cho xác minh mô hình AC; các lựa chọn của các thành phần và các bước có thể phụ thuộc vào độ phức tạp của mô hình và chi phí để thực hiện khung thử nghiệm. Do đó, khung kiểm tra mô hình AC có thể chứa các thành phần / chức năng tùy chọn trong Hình 23 ngoại trừ việc Hộp đen được kiểm tra là điều cần thiết.

