UC San Diego

UC San Diego Previously Published Works

Title

Path Dependencies in Bilateral Relationship-Based Access Control

Permalink

https://escholarship.org/uc/item/3jw4w27k

ISBN

9789811967900

Authors

Gupta, Amarnath

Bagchi, Aditya

Publication Date

2023

DOI

10.1007/978-981-19-6791-7_2

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, availalbe at https://creativecommons.org/licenses/by/4.0/

Peer reviewed

eScholarship.org Powered by the California Digital Library University of California

Path Dependencies in Bilateral Relationship-Based Access Control

Amarnath Gupta[1] and Aditya Bagchi[2]

[1] University of California San Diego, La Jolla CA, 92093, USA
a1gupta@ucsd.edu
[2] Indian Statistical Institute, Kolkata 700108, India
bagchi.aditya@gmail.com

Abstract. The Relationship based Access Control Model (ReBAC) generalizes Role-based Access Control (RBAC) by considering both hierarchical and non-hierarchical relationships between users to specify access control of a set of target resources (objects). This paper extends the ReBAC model by considering relationships between objects as well as between subjects and objects. This generalized model is expressed through the language of dependencies borrowed from data management. We develop a language for bilateral path dependencies which states that a chain of binary relationships over subjects and objects logically imply another chain of binary relationships. We show that this formalism is adequate to capture access control rules with no conflicts. In future work, this formalism will be extended to include conflict detection and resolution.

Keywords: Access control, · ReBAC, · Bilateral Relationship

1 Introduction

Access control policies primarily specify how a user (usually referred as subject) can access a resource (usually referred as object) with a set of access rights (read, write, execute etc.). An atomic access control policy is typically expressed as a 4-tuple structure (s, o, a, v) using a set of $Subject(S)$, $Object(O)$, $AccessRights(A)$ and $Sign(V)$ where $s \in S$, $o \in O$, $a \in A$ and $v \in V$. The sign in an access control policy can either be +ve or -ve, where a +ve sign indicates permission for the concerned subject to access the concerned object using the access right specified and a -ve sign indicates denial of such access. Logically, a policy set for a user/subject is expressed by many such atomic policies for different resources/objects and also by logical combinations of such atomic policies using Boolean operators[9]. Depending upon the context and policy, *Subject* can be users, group of users or roles, and *Object* can be any resource within an enterprise that can be accessed by a Subject. Moreover, at times, an Object can also be a Subject for an access policy. For example, when a user u tries to execute a program p, u is the Subject and p is the Object, but when the same program p accesses a file f during its execution, p is the Subject and f is the Object.

In practice, not all authorization rules are explicitly specified. Instead, inference rules are used to derive authorizations using the rules of the access control model used. Specifically, user-group hierarchy ensures that members of a sub-group inherit authorizations from its super-group. Similarly, role hierarchy permits a higher order role to inherit authorizations from the roles below. More recently, authorization models like RPPM [12] and ReBAC [11] have extended authorization inference beyond hierarchies and inheritance, and created graph based authorization schemes.

1.1 Motivation for the present work

Recently, multi-graph models are increasingly being used in studying both enterprise security as well as vulnerability analysis [6] because multi-graph models represent multiple types of non-hierarchical relationships that security rules can exploit. However, we find that many models, including ReBAC and RPPM, focus primarily on subject-subject relationships, less on subject-object relationships and rarely on object-object relationships. However, in real life, all three types of relationships should factor into a security specification. For example, if a user u has read access to a DBMS D, and D resides on server S, u must have access to S to exercise the read-access to D. However, *there is no access-control formalism today that expresses such non-hierarchical implications*. In this paper, we study such a formalism based on the language of dependencies borrowed from database theory [1].

We try to establish that an extension of ReBAC model is necessary where object/resource side and user/subject side relationships and hierarchies need to be incorporated. We call this model the Bilateral Relationship Based Access Control model (BiReBAC). We have introduced a new formalism based on dependency constraints that generalize the standard ReBAC model to give rise to our Bilateral Relationship Based Access Control model (BiReBAC). However, this being the first proposal for the composite model, we have adopted an assumption to ensure completeness and soundness of our authorization specification. Our proposed model is based on *Closed Policy* and any access is permitted only against explicit positive authorizations or any other authorizations inferred from them. So against any query with any subject-object combination only a positive authorization can be inferred. So for any access request if no positive authorization can be inferred, the access request will be denied. So in a *Closed Policy* no explicit negative authorization is specified. This assumption avoids possible conflicts between positive and negative authorizations and the well-known decidability problem as specified in the HRU model [8]. We will address this issue again in the last section to indicate about our future work.

We now present a running example, to explain our proposed model.

A Running Example Let's consider a modern day *executable electronic textbook* on Data Science that is available over the web. The book has a number of chapters, broken down into a hierarchy of sections and subsections. Some of the textual content are sample problems whose worked-out solutions are provided as executable Jupyter Notebooks that the readers can run. We call these Problem-text-to-Solution links as *forward links* (flinks). Each section also has a set of exercise problems and a reading list. The exercise problems are written in text; however, they are also connected to data sets and solutions (Jupyter Notebooks) provided by the authors. A Notebook may link back to the paragraphs of the book relevant to the exercise problem being solved. We call these Notebook-cell-to-text links as *reverse links* (rlinks). Let us also consider that the authors of the book have created a set of distinct tracks (e.g., "Beginners", "Advanced") which are pathways through the book for different audiences. A "track" is a tree-like structure through the chapters, sections and subsections of the book.

Clearly, our executable textbook is a *distributed object*, i.e., different part of the book (e.g., different sections) are on different physical web servers. The book has a set of "principal authors", but sample problems may be written by the graduate students of a principal author. Solutions to the exercise problems may be created by undergraduate students who work in a principal author's lab or have taken the Data Science class offered by one of the principal authors. The universe of these books can be represented as a graph shown in Fig. 1. We can write several access control policies based on this example. We can say that "every author has read, write and execute access on the Jupyter Notebooks he writes". We can also make relationship based access policy statements like "a graduate student S who works under the supervision of a professor P and researches on subject X, has read access to any textbook written by P on X. This paper introduces a formalism different from [5, 12] to cover access control policies where the objects and their fragments (e.g., a subsection of a book) which are objects themselves connected through a relationship network.

2 The Language of Dependencies

We formalize graph-based access rules in terms of dependency statements. The intuition behind dependency statements is to identify the critical factors on which the access of a user to a resource depends. Suppose we want to make the statement that user u has access to some portion p of a book b, if and only if he has access to the web server (WS) w that hosts p. We can write this as:

$$\forall u : user, p, b : book, m \ hasAccess(u, p, m) \Rightarrow \exists w : WS \mid \\ partOf^{+}(p, b), hosts(w, p), hasAccess(u, w, m) \quad (1)$$

where the "," symbol represents conjunction (AND), $: book, : WS$ are unary typing predicates, m, the free variable designates the access right (e.g., read, write, execute), and $partOf^{+}$ is the transitive closure of the part-of relationship. Thus, the access of u to p depends on the access of u to w. Dependency statements like Eq. 1 has practical consequences. If there is an access control policy that an undergraduate student has no access to a specific departmental web server, then no Jupyter Notebook for exercise problems can be placed on that server.

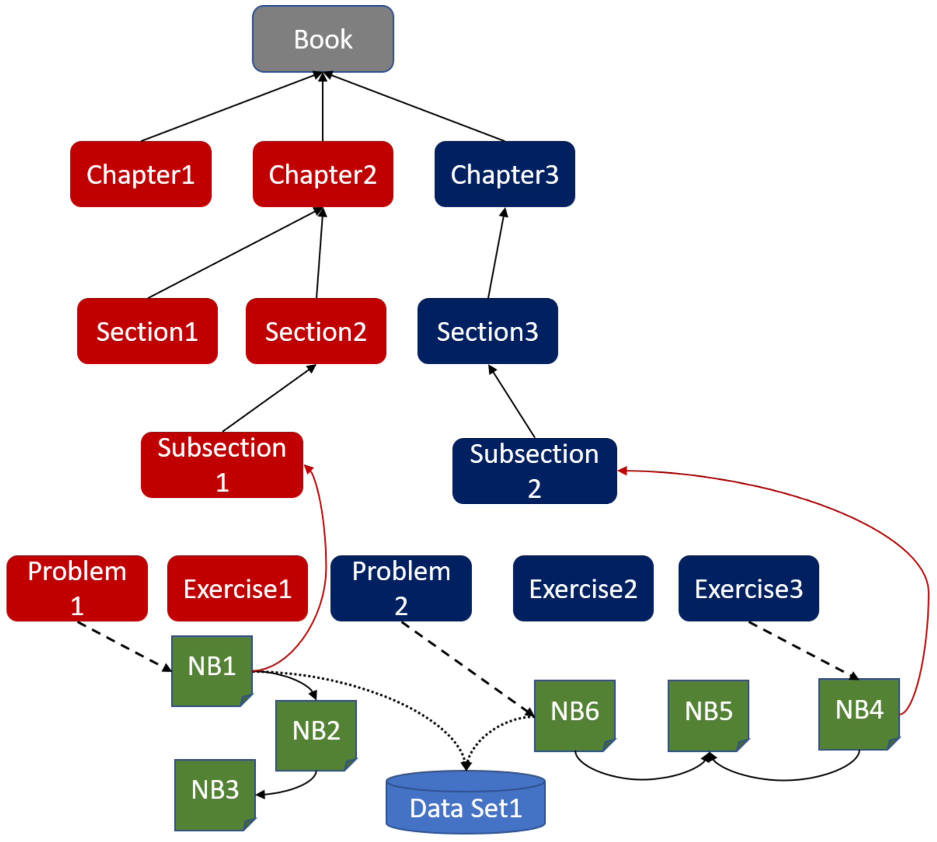

Fig. 1. A Graph-centric View of our object domain. Solid straight black arrows = $partOf$, solid curved black arrows = $connectedTo$, dashed arrows = $flink$, brown arrows = $rlink$. Red nodes = Track 1, dark blue nodes = Track 2.

Eq. 1 represents an access control rule with single-variable dependency (i.e., the access depends on the existence of just a web server with proper access right) which we call *node dependency*.

We can extend the notion of node dependency if we want to state that in addition to Eq. 1, a user u can execute a Jupyter Notebook (NB) on the web server w, if Python is installed on w and u has execute privileges for that Python installation. Eq. 2 states this condition.

$$\forall u : user, w : WS, j : NB\ hasAccess(u, j, \texttt{execute}) \Rightarrow \exists w, p \mid \\ installed(p, \texttt{Python}, w), runsOn(j, p), hasAccess(u, p, \texttt{execute}) \quad (2)$$

where predicate $installed(p, \texttt{Python}, w)$ states that p is an instance of Python that is installed on w. Eq. 2 differs from Eq. 1 because the two existentially quantified variables w and p on which the u's execute access to j depends, are constrained to satisfy a chain of relationships. We tend to call this form of multi-variable dependency as *chain dependency*. In general, a chain dependency relationship has the form

$$\forall vars1[: \mathit{typeSpec}], \mathit{hasAccess}(\mathit{user}, \mathit{resourceVar}1, \mathit{mode}) \Rightarrow \exists vars2 \mid \\ \mathit{chainExpression}, \mathit{hasAccess}(\mathit{user}, \mathit{resourceVar}2, \mathit{mode})\ldots \quad (3)$$

where $vars1$ and $vars2$ are sets of variables, $resourceVar1$ refers to the original resource whose access is being determined, and $resourceVar2$ refers to the resource on which the original access depends. The variable $resourceVar1$ belongs to the universally quantified variable set $vars1$ and $resourceVar2$ belongs to the existentially quantified variable set $vars2$. The chain expression is a conjunctive predicate over a set of participating relationships and their transitive closures, and must include all existentially quantified variables. The semantic types of the variables are optionally specified as *typeSpec* of unary atomic type specifiers for variables in $vars1 \cup vars2$. Although Eq. 3 has one *hasAccess* predicate, a general chain dependency expression may have multiple intermediary access requirements (indicated by the dots) for the LHS of the implication to be satisfied.

2.1 Inferences From Node and Chain Dependencies.

The implication in the dependency-based formulation leads to two kinds of inferences. The first category derives from *tuple-generating dependencies* (TGD) used in data management literature [3, 2], and the second relates to *implicit access-mode assignment* (IAM), a generalization from prior work by Dasgupta et al for ontological data access for digital libraries [7].

TGD: Node dependency is a form of TGD. In our example, if we have a ground fact like $\mathit{hasAccess}('\mathit{Joe}', '\mathit{Section}: 3.2', '\mathit{read}')$ Eq. 1 also asserts the existence of a tuple $\mathit{hasAccess}('\mathit{Joe}', w1, '\mathit{read}')$ in the *hasAccess* table for some web server $w1$. If not, the system is inconsistent. For chain dependency, a ground fact like $\mathit{hasAccess}('\mathit{Joe}', '\mathit{Exercise}: 3.2.14', \texttt{execute})$ implies a set of tuples

$\mathit{installed}(p1, \texttt{Python}, w1)$,
$\mathit{runsOn}('\mathit{Exercise}: 3.2.14', p1)$, and
$\mathit{hasAccess}('\mathit{Joe}', p1, \texttt{execute})$

where the predicate names map to table names in an access control database. However, by virtue of their generation rule, these three tuples are not independent of each other and must be considered to be a group. We call this generalized form of TGD a *tuple-group generating dependency* (TGGD). Note that the tuples inferred from a TGGD belong to *multiple relations*.

IAM: The IAM problem can be illustrated by slightly modifying Eq. 1 as follows.

$$\forall u: \mathit{user}, p, b: \mathit{book}, m, m'\ \mathit{user}(u), \mathit{hasAccess}(u, p, m) \Rightarrow \exists w: WS, f: \mathit{file}, m' \mid \\ \mathit{partOf}^{+}(p, b), \mathit{hosts}(w, p), \mathit{hasAccess}(u, f, m), \\ \mathit{contains}(w, f), \mathit{locatedIn}(p, f), \mathit{hasAccess}(u, w, m') \quad (4)$$

In this case, we have added an existentially quantified variable f which represents a file such that p, the part of the book, is located in file f, which is

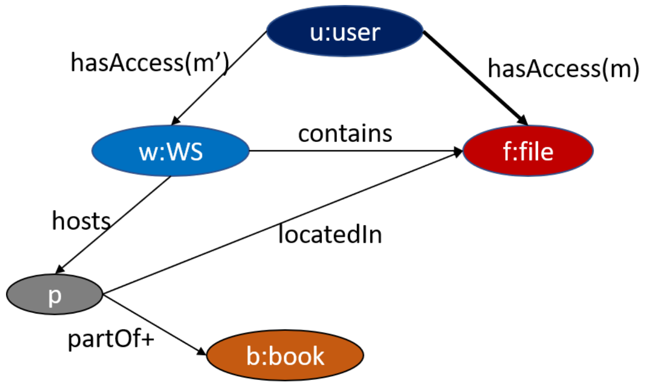

Fig. 2. A graph representation of dependency equation Eq. 4. The *hasAccess* edge from the LHS is made thicker.

contained in web server w. Now, if u has access to the file f in modality m, u must also logically have access to the web server w in some mode m'. However, the nature of m' is not specified, although it is clear that m' depends on the value of m. There are two issues to be resolved:

1. Why is m' different from m, the original access mode assigned to user u ?
2. How is m' implicitly assigned for accessing web server w ?

A similar situation was encountered in [7] where the bibliographic metadata of a digital library was represented by an ontology. A *read* access by a user u to a document d containing in the concept c would implicitly provide an access to the concept c itself. However, it does not infer that all documents in c can be read by u. The authors have solved this problem by defining an access mode called *browse* where the user can access concept c to reach the document d but cannot execute any operation related to other access modes like (*read, write, execute*). It makes authorization inference mechanism at the resource/object side different from the same at user/subject side.

To remove the ambiguity for m', we create a set of template IAM rules that apply to the right side of the dependency equation. To formulate the template IAM rule, we interpret the predicates of the dependency equation as a graph (Fig. 2), where a binary predicate like *contains* is interpreted as a directed edge from the first argument to the second, and a ternary predicate like *hasAccess* is interpreted as a directed edge from the first argument to the second, where the third argument is a property of the edge. A unary predicate like *book* is interpreted as the node type of its argument variable. We say that the LHS $hasAccess(m')$ edge is dependent on the RHS $hasAccess(m)$ edge, contingent upon the *constraining subgraph* that connects the resources w and f. From Figure 2, the $partOf+$ edge does not participate in the constraint because removing it does not change the nature of the dependency. Now we can formulate our template rule as follows. Given the constraining subgraph of the form $hosts(A, B), locatedIn(B, C), contains(A, C)$, if m is the mode of access for the LHS $hasAccess(u, A, m)$ edge, then the m', the mode of access for all RHS $hasAccess(u, C, m')$ edges will be given by `intention-to-`m which is m'. Thus if m is `read`, m' is `intention-to-read`, an idea borrowed from multiple granularity locking protocol in DBMS systems [10].

According to the locking protocol in a DBMS, if a table in a relational system is updated with a *write* lock, the entire database will have an `intention-to-write` lock indicating that down below in a more granular object actual write operation is in progress. The same concept can be used to infer access authorizations for objects which contain the actual object for which explicit authorization has been specified. Once again, access right of type `intention-to-`m assigned to work station w in Eq. 4 allows the concerned user to reach and access p and f by accessing work station w but no other operation is allowed on w.

To generalize this formulation, we recognize that like $partOf$, $hosts$, $contains$ and $locatedIn$ are transitive relationships. Now, we can write a more general form of the IAM dependency pattern, which applies to multiple dependency rules that have the same constraining subgraph.

IAM Template 1: Given an independent access relation $hasAccess(u, C, m')$, a dependent access relation $hasAccess(u, A, m)$, where m is a valid access mode, A, C are system resources, and a *constraining subgraph* pattern $hosts^+(A, B)$, $locatedIn^+(B, C)$, $contains^+(A, C)$ the implicit access mode m' is assigned as `intention-to-`m.

Using Chain Dependencies. There can be different use cases for chain-dependency in our example application.

Example 1. A reader of the book has read permission to the pages of a section/subsection s if he has read all the prerequisite portions for s, and completed their exercise problems (i.e., run the Jupyter Notebooks associated with the exercise problems).

$$\begin{aligned}&\forall u : user, b : book, s, (typeOf(s)\ \mathbf{in}\ ('section', 'subsection')), hasAccess(u, s, \texttt{read})\\ &\Rightarrow \exists\ s', e : exercise \mid (typeOf(s')\ \mathbf{in}\ ('section', 'subsection')), partOf^+(s, b),\\ &partOf(e, s'), prereq^+(s', s), \{hasAccessed(u, s', \texttt{read})\ \mathbf{then}\ hasAccessed(u, e, \texttt{execute})\}\end{aligned} \tag{5}$$

In this example, *hasAccessed* is a *state predicate* which serves as a precondition for the access rule on the LHS. Further, the signature

$$\{< statePredicate > \mathbf{then} < statePredicate >\}$$

indicates a sequence of states that must be satisfied one after another. To evaluate this signature, the exercise e which is executed has to be part of the section/subsection s', the antecedent of the **then** structure. In other words, the (s', e) pair in the precondition is constrained so that $partOf(e, s')$ holds.

Example 2. A user enrolled in a track T *does not* have read access to any subsection S of a book or write/execute access to the exercises if S is not in T.

$$\begin{aligned}
&\forall u: \mathit{user}, b: \mathit{book}, s: \mathit{subsection}, \mathit{partOf}^{+}(s,b), \mathit{hasAccess}(u,s,\texttt{read}) \Rightarrow \\
&\qquad\qquad \neg\exists\, t: \mathit{track} \mid \mathit{enrolledIn}(u,t), \mathit{inTrack}(s,t) \qquad (6) \\
&\forall u: \mathit{user}, e: \mathit{exercise}, b: \mathit{book}, \mathit{hasAccess}(u,e,\{\mathit{write}, \mathit{execute}\}) \Rightarrow \\
&\qquad \neg\exists\, t: \mathit{track}, s: \mathit{subsection} \mid \mathit{partOf}^{+}(s,b), \mathit{contains}(s,e), \\
&\qquad\qquad \mathit{enrolledIn}(u,t), \mathit{inTrack}(s,t) \qquad (7)
\end{aligned}$$

where the predicate *contains* is the inverse of predicate partOf. In other words, $\mathit{partOf}(a,b)$ means a is partOf b, whereas $\mathit{contains}(a,b)$ means b contains in a. In this pair of conditions, the chain dependency is purely *structural* because the access does not depend on any implicit or past access conditions.

Example 3. A student can create a new Jupyter Notebook page on a web-server only if he falls within the reporting hierarchy of any of the authors, and has the explicit permission from her supervisor to upload data to the web-server.

$$\begin{aligned}
&\forall s: \mathit{student}, j: NB, w: WS, b: \mathit{book}, \mathit{canCreate}(s,j,w) \Rightarrow \\
&\qquad \exists\, p: \mathit{professor}, d: \mathit{dataFile}, l: \mathit{permissionToken}, x \mid \\
&\qquad\quad \mathit{author}(p,b), \mathit{supervises}^{+}(p,s), \mathit{supervises}(x,s), \\
&\qquad\qquad \mathit{hasPermission}(s,l,\mathit{canUpload}(s,d,w)) \qquad (8)
\end{aligned}$$

We use $\mathit{canCreate}(s,j,w)$ as a specialization of the more standard form of $\mathit{hasAccess}(\mathit{user}, \mathit{resource}, \mathit{accessMode})$ used so far. We can rewrite the $\mathit{canCreate}(s,j,w)$ predicate as $\mathit{hasAccess}(s,w,\texttt{create}(j))$ where the access mode `create` is parameterized by the object to be created. The permission token l is similar to an API key used for accessing web and mobile services. To apply the token, we use $\mathit{hasPermission}(\mathit{user}, \mathit{token}, \mathit{accessPredicate})$ as a *second-order predicate* that enables an access pattern via an explicit permission condition. The permission condition can be viewed as a type of eventive precondition that must be satisfied for the LHS *hasAccess* to take effect.

Incidentally, this chain dependency also takes care of provisions (pre-conditions) and obligations (post-conditions) involved in specifying access control rules [4].

2.2 From Chain Dependencies to Bilateral Path Dependencies

In the previous subsection, we used the term chain dependency to refer to a series of connected predicates on the RHS of the dependency rule. We can think of the aforementioned chains as "RH paths" that exist only on the right hand side of the implication symbol. A more general form of dependency can be defined by having a path expression on both sides of the implication. We call this form a *bilateral path dependency* rule.

We initiate our approach with a simple implication rule that does not have a dependency formulation but has a LH path expression. In our example situation,

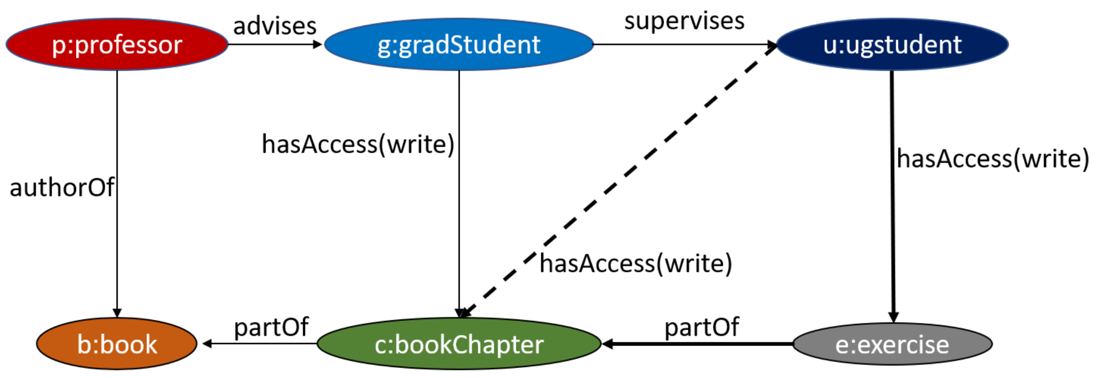

Fig. 3. A graph representation of dependency equation Eq. 10. The edge from the RHS are made thicker. The dashed edge reflects the secondary implication.

such a rudimentary (but unrealistic) relationship-based access control statement can be "if g, a graduate student, has write access to a book chapter authored by p who is g's faculty advisor, then so does g's undergraduate advisee u".

$$\begin{aligned}&\forall u : ugStudent, g : gradStudent, p : professor, b : book, c : bookChapter,\\&\quad authorOf(p, b), partOf(c, b), advisor(p, g), supervises(g, u),\\&\qquad hasAccess(g, c, \texttt{write}) \Rightarrow hasAccess(u, c, \texttt{write})\end{aligned} \tag{9}$$

We can make this statement more realistic by adding the additional condition that u has this write access only if u writes an exercise for some portion of the chapter. Now Eq. 9 becomes

$$\begin{aligned}&\forall u : ugStudent, g : gradStudent, p : professor, b : book, c : bookChapter,\\&\quad authorOf(p, b), partOf(c, b), advisor(p, g), supervises(g, u),\\&\quad hasAccess(g, c, \texttt{write}) \Rightarrow \exists e : exercise \mid hasAccess(u, e, \texttt{write}),\\&\qquad partOf(e, c), hasAccess(u, c, \texttt{write})\end{aligned} \tag{10}$$

With this extension, we have path expressions on both sides of the implication in a dependency rule, making this constraint an example of a bilateral path dependency. The rule is shown as a graph in Fig. 3, where existence of the subgraph with heavy edges depends on the existence of the subgraph.with light edges. We make the following observations about Eq. 10:

(a) One prerequisite for the ugStudent to get write access to the exercise is that he or she is an advisee of the author of the book. This encodes an ReBAC-style condition within the fold of the bilateral path dependencies. Thus, the language of bilateral path dependencies have the expressivity to capture both the subject-side ReBAC criteria and its object-side extensions.
(b) The ugStudent primarily has write access to the exercise, and consequently also has write access to the book chapter. However, this is not explicitly cap-

tured in the equation. One way to capture this "derived" mode of write access is to change the access mode to the book chapter to `intention-to-write` as we did in the IAM case. The second, more direct way to represent this is to add a secondary implication in the RHS. In this case, the RHS is:

$$\exists e : exercise \mid hasAccess(u, e, \texttt{write}), partOf(e, c), \longrightarrow hasAccess(u, c, \texttt{write})$$

We prefer to use this secondary implication notation to make the dependency expression more precise.

(c) In this case, the undergraduate student gets write access to the exercise of the chapter that the graduate student has write access to primarily because the graduate student supervises the undergraduate student. We can write a companion, non-dependency-generating rule that the graduate student who has access to a resource r has the capability to grant access of r or a part thereof, to the undergraduate student that he(she) supervises, is:

$$\begin{aligned}&\forall u : ugStudent, g : gradStudent\ \exists_0 r : resource\\&\quad \mid hasAccess(g, r, m), grantsAccess(g, u, r', m'), partOf^*(r', r)\end{aligned} \quad (11)$$

We use the symbol $\exists_0$ to denote that while g has the capability of granting access, in reality there may be no resources for which the capability is exercised. Further, in reality, the modality term *grantsAccess*(grantor, grantee, resource, modality) predicate can be more nuanced. For example, if g has read-access, he cannot grant a write or execute access to u. This consideration directly shows that the well known Discretionary Based access control (available even in SQL) can also be mapped to our model.

Coupled and Independent Bilaterality. In Eq. 10, the variables used on the LHS of the implication are $V_L = (u, g, p, b, c)$ and those on the RHS are $V_R = (u, c, e)$. Thus, $V_L \cap V_R = (u, c)$, i.e., $V_L \cap V_R \neq \emptyset$. In this case, we call the dependency rule as *coupled bilateral path dependency* (or *coupled bilaterality*); on the other hand, a bilateral path dependency rule where $V_L \cap V_R = \emptyset$ is called *independent bilateral path dependency.* We make the following assertion.

Assertion 1 *An access control rule with an independent bilateral path dependency is unsatisfiable.*

Justification: Instead of a formal proof we prefer to present a logical justification in support of the assertion made.

1. As shown in Eq. 10 and in other equations earlier, the complete set of chains on both sides of an equation is formed by considering the transitive closure of all the relationships connecting both the subject side and the object side. Thus $advises/supervises^+$ connects ugstudent to gradstudent to professor and $partOf^+$ connects exercise to bookchapter to book as shown in Fig. 3. Thus chain of inferred authorizations connect different users to the initial user-group, the authors of the book and offer access to different parts of the initial object, the book.

2. If $V_L \cap V_R = \emptyset$ then it implies that even after considering the transitive closure of all relationships on both subject side and object side, some subjects/users cannot reach to initial set of subjects and/or objects where explicit authorizations were specified. Thus, such subjects/users will not have proper authorizations inferred to access the required objects. From graphical point of view, no path will be available for any such access.

So, the assertion stands and only *coupled bilateral path dependency* are allowed.

In light of Assertion 1, we only focus on rules with coupled bilateral path dependency. Recall from Eq. 10 that nodes u, c are in the intersection of the LHS and RHS – we call them the *anchor nodes* of the dependency graph (Fig. 3). Justification given for Assertion 1, also asserts that coupled bilateral path dependency graphs (CBPD graphs) are always connected. Further, the anchor nodes of CBPD graphs can have outgoing edges only to other anchor nodes or to variables on the RH side of the implication.

2.3 CBPD Graphs and Hierarchies

Access control rules typically make use of subject-side hierarchies (e.g. over users and roles) as well as object-side hierarchies (e.g., a classification of books in a digital library [7]). A hierarchy-based access control specification is represented as two rules – the first specifies the hierarchy-generating relationship r_h (e.g., *supervises(user, user)*), and the second specifies the access implication of a member m_1 of a hierarchy with respect to member m_2 if they are in the same tree-path (sometimes DAG path) induced by r_h.

To express the interaction between a CBPD graph and hierarchies, we extend our example in Eq. 10. We discuss a scenario where professors have senior postdoctoral students, who supervise junior post doctoral students, postdoctoral students (junior and senior) supervise senior graduate students, who in turn supervise the work of junior graduate students. Here, we treat the *supervises* relationship as transitive and notice that it creates a DAG because a senior graduate student may be supervised by a junior or a senior postdoc. Now, we would like to still use the rule in Eq.10 with the following differences:

(a) A subgraph S depicting the *supervises* hierarchy must be created.
(b) Following the same notation as Eqs. 7 and 8, the *advises* edge is replaced with a *supervises+* edge in Figure 3.
(c) The variables (nodes) p and g should now be connected to the corresponding nodes of S by an explicit $instanceOf$ edge. This presents a slight expressivity problem if the node marked g can be any direct or indirect supervisee of p (i.e., a senior/junior graduate student or postdoc). We accomplish this by
   - creating a subgraph S' of S where S' does not include professors
   - creating a *type-predicate* where instead of making the type assertion $g : gradStudent$, we write $g : memberOf(nodes(S'))$

   The modified part of the graph is shown in Figure 4

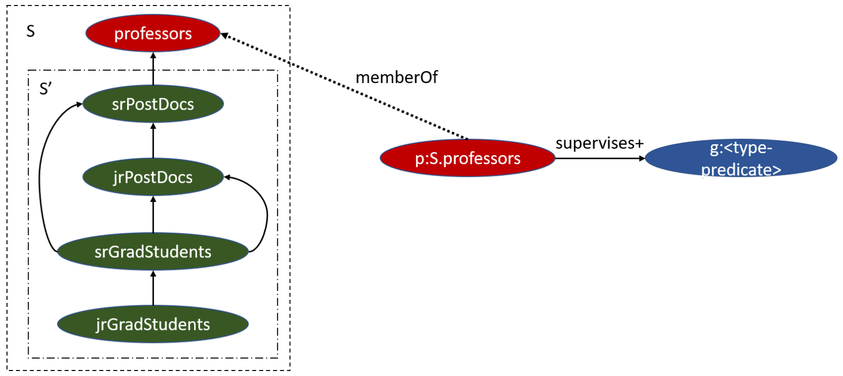

Fig. 4. The hierarchy and the affected nodes from Figure 3 are shown. Note that the $memberOf$ edge is equivalent to the type assertion $p : S.professors$ made in the node.

Thus, the inclusion of hierarchies changes the nature of the CBPD graphs because it introduces the need to create *named subgraphs* which can be referenced by type resolution logic at a node. To test whether a specific undergraduate student has write access to the exercises, the eligibility of her supervisor must resolve to a node type within subgraph S'.

2.4 Towards a BiReBAC Graph Model

From the example of book writing project and gradual development of a new relationship based access control model through Eq. 1 to Eq. 10 we possibly could establish that an extension of ReBAC model is necessary where object/resource side and user/subject side relationships and hierarchies need to be incorporated. We call this model the Bilateral Relationship Based Access Control model (BiReBAC). While developing a full translation of the dependency based formalism is beyond the scope of this paper, we outline below a rough sketch of how a property graph corresponding to Figs. 3 and 4 may be constructed from our rules, based on the Eq. 10.

The general scheme of a BiReBac graph construction follows:

1. Both objects and subjects are represented as typed nodes such that node types are entity classes defined by the problem space. If these classes are enumerated over a hierarchy as in Fig. 4, nodes in the hierarchy are represented as node types in a BiReBAC graph. As defined in the explanation of the figure, a node may have multiple types specified through a *type-predicate*.
2. There are three types of edges - $hasAccess$ edges that specify access rights, relationship edges between subject pairs, object pairs and subject-object pairs, and $subclass/membership$ edges. Traditional authorization relationships such as inheritance through user-group/subgroup or super-role/sub-role are expressed directly as edges or as predicates over these edges.
3. Node properties of BiRBAC graphs need to specify if a node is on the antecedent side, the consequent side or is an anchor node. Similarly, a node

property also captures if a node is existentially qualified on the consequent side (otherwise, all nodes are universally quantified).

4. When multiple dependency rules apply, our strategy is to create multiple graphs, and using the principles of the web ontology language (OWL), declare comparable nodes as equivalent by creating an `equivalent` edge between them. For example, the node type `Book` is used in graphs of Fig. 2 and 3 – these nodes will be connected through the equivalence edge.

   In future work, we will present a formal construction for BiReBAC graphs and prove that the construction will produce unambiguous graphs.

3 Discussion and Future Work

We have introduced a new formalism based on dependency constraints that generalize the standard ReBAC model to give rise to our Bilateral Relationship Based Access Control model (BiReBAC). However, this being the first proposal for the composite model, swe have adopted *Closed Policy* to ensure completeness and soundness of our authorization specification.

In the well-known HRU model [8] it has been shown that in a network structure where both positive and negative authorizations are present, authorization of a particular node may be undecidable. However, since our present proposal is based on only positive authorizations such decidability problem will not be present. Presence of both positive and negative authorizations, related undecidability problems and possible mitigation is part of future work.

In Section 2.4, we qualitatively argued that the dependency-based formalism lends itself well to a property graph model. We will develop a provably sound and correct bidirectional translation from the dependency-rules to an extended version of the property graph model. We also expect to develop a modified query engine that will accept this extended property graph. We will prove that the implication rules presented in the paper will be appropriately translated into a combination of query operations and inferencing over these property graphs. Finally, we expect to develop a policy server that will use this extended property graph based model for its operations.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases, vol. 8. Addison-Wesley Reading (1995)
2. Baudinet, M., Chomicki, J., Wolper, P.: Constraint-generating dependencies. Journal of Computer and System Sciences **59**(1), 94–115 (1999)
3. Beeri, C., Vardi, M.Y.: Formal systems for tuple and equality generating dependencies. SIAM Journal on Computing **13**(1), 76–98 (1984)
4. Bettini, C., Jajodia, S., Wang, X.S., Wijesekera, D.: Provisions and obligations in policy rule management. Journal of Network and Systems Management **11**(3), 351–372 (2003)

5. Crampton, J., Sellwood, J.: Path conditions and principal matching: a new approach to access control. In: Proc. of the 19th Symp.on Access control models and technologies. pp. 187–198. ACM (2014)
6. Das, S.K., Bagchi, A.: Representation and validation of enterprise security requirements, a multigraph model. In: Advanced Computing and Systems for Security, vol. 6, pp. 153–167. Springer (2018)
7. Dasgupta, S., Pal, P., Mazumdar, C., Bagchi, A.: Resolving authorization conflicts by ontology views for controlled access to a digital library. J. Knowledge Management **19**(1), 45–59 (2015)
8. Harrison, M.A., Ruzzo, W.L., Ullman, J.D.: Protection in operating systems. Communications of ACM **19**(8), 461–471 (1976)
9. Jajodia, S., Samarati, P., Subrahmanian, V.: A logical language for expressing authorizations. In: Proc. IEEE Symposium on Security and Privacy (Cat. No. 97CB36097). pp. 31–42. IEEE (1997)
10. Molina, H.G., Ullman, J.D., Widom, J.: Database systems the complete book (2002)
11. Rizvi, S.Z.R., Fong, P.W.: Interoperability of relationship-and role-based access control. In: Proc. of the 6th Int. Conf. on Data and Application Security and Privacy (CODASPY). pp. 231–242. ACM (2016)
12. Sellwood, J.: RPPM: A Relationship-Based Access Control Model Utilising Relationships, Paths and Principal Matching. Ph.D. thesis, Royal Hallway, University of London (May 2017)