Healthcare and business solutions are transforming towards decentralized architectures. In this regard, many blockchain-based solutions have been proposed. Some are order-execute architecture while some are based on execute-order architecture of blockchain. Execute-order architectures are famous for general purpose business applications. Hyperledger-composer is a smart-contract framework specifically designed to model business apps comfortably. Hyperledger-composer supports role-based access control, however, it is new to adopt by Healthcare applications. The most important aspect of Healthcare applications is to provide access to sensitive documents. In this paper, we are formally defining constraint role-based access control (CRBAC) in Z formalization language to prove the properties of CRBAC in a Healthcare document sharing for blockchain, specifically for Hyperledger-composer. The paper aims to take Healthcare document’s access via constraint-based access control while the interface to the roles and documents are decentralized.

**Keyword:** Blockchain, Healthcare, Decentralized Healthcare, Access Control, Z formalization
among the patient and Healthcare roles, such as, doctor, technician, laboratories representatives. There are many use-cases where the release of health record needs trust among various stakeholders. Blockchain based solutions help to tackle these problems. One of the IBMs solutions provide Hyperledger composer. Composer is based on role-based access control. For further reading on composer access control, the author refers the reader to the document [9].

In this regard, we are providing a constraint role-based control formalization in Z notation. The formalization is specifically made for Healthcare document sharing on Hyperledger composer among various roles. It will help formally prove all the transactions in a Healthcare record setting for the newly designed Healthcare apps over blockchain. Z formalization is very well defined in [8] for further reading.

Access control is a core functionality required when working with Healthcare. It is required in provision of basic resources needed to certain groups or participants. In [4], the concept of distributed ledger utilized to store transactions. They have ensured that various stakeholders in the e-health have knowledge about rest of parties that are dealing with e-Health resources in order to maintain integrity, scalability, authenticity and audibility. However, the work done in this research deviates from our work in many aspects.

In [7], a cloud based Healthcare system is presented that possess important features necessary for ensuring privacy of Healthcare applications and confidential data. The concept of utilizing clouds for applications has become famous. However, it faces challenges in terms of privacy. In [2] Horizon et. al, employed general data protection regulations (GDPR) to embed multiple security tests and data protection tools in common deployable infrastructure. Study reported in [11] by Zhang et. al, applied various software patterns to address interoperability in Blockchain-based Healthcare Applications. They identified features and implementation challenges in Healthcare interoperability. Given an end-to-end case study of a blockchain-based healthcare application.

Yue et al. [10] developed an application namely Healthcare Data Gateway (HGD) based on blockchain. The solution helps control and share clients data easily without compromising privacy. It provides an excellent way to increase intelligence of Healthcare systems and at the same time keeps patient data private. Their proposed access control model make sure to provide control over Healthcare records in Blockchain network. Dagher et al. [3] presented a framework, a blockchain-based, for providing access to various stackholders in a secure manner. The stackholders includes providers, third parties and patients. The framework is interoperable and work efficiently. The framework preserves the privacy of patients sensitive information. Our framework, named Ancile, utilizes smart contracts in an Ethereum-based blockchain for heightened access control and obfuscation of data, and employs advanced cryptographic techniques for further security.

**Contribution:** The work done so far provides various aspects of Healthcare records sharing. However, based on our knowledge none of the previous research shown access control mechanisms for Healthcare apps on a private blockchain, such as, Hyperledger Composer. In this paper, we are formally defining in Z Notation all the properties of RBAC and CRBAC of Healthcare document sharing based on SECTET.

**Paper Structure:** The rest of the paper is organized as follows. Section 2 defines complete RBAC properties in Z formalization. Section 3 shows extension to the NIST formalization of RBAC and that more fits in private blockchains. Section 4 brings constraints to RBAC and explain the document sharing with a role in a Healthcare environment. In section 5 the author concludes the paper.
Specifying RBAC in Z

RBAC is the core access control model of Hyperledger Composer for simplicity of the reader we are initially defining the complete RBAC formalization in Z. Later we will proceed with constraint-based access control for Healthcare document sharing based on SECTET.

2.1 Building Blocks of RBAC

This is merely an outline of the RBAC model and is described purely in Z notation. For a discussion of the constraints and semantics, see [6].

USERS, ROLES and OPERATIONS are the only basic datatypes in RBAC.

\[ \text{[USERS, ROLES, OPERATIONS]} \]

We need to define a shorthand symbol (\( \rightsquigarrow \)) for inherits function since it is used by many other functions and constraints. \( r_1 \rightsquigarrow r_2 \) is a relation which returns true if role \( r_1 \) inherits from role \( r_2 \). We also define the transitive and the transitive, reflexive closure of this relation.

\[
\begin{align*}
\text{relation (}_\rightsquigarrow_\text{)} & \quad \text{relation (}_\rightsquigarrow^+_\text{)} \\
\text{relation (}_\rightsquigarrow^*_{_\text{)}} & \quad : \text{ROLES} \times \text{ROLES} \\
\forall x, y : \text{ROLES} & \quad \cdot \ x \rightsquigarrow y \iff (x, y) \in \text{inherits} \\
\forall x, y : \text{ROLES} & \quad \cdot \ x \rightsquigarrow^+ y \iff y \in \bigcup \{ n : \text{N} \mid n \geq 1 \cdot \text{inherits}^n(x) \} \\
\forall x, y : \text{ROLES} & \quad \cdot \ x \rightsquigarrow^* y \iff x \rightsquigarrow^+ y \lor x = y
\end{align*}
\]

Some helper functions:

\[
\begin{align*}
\text{authorizedRoles : USERS} & \rightarrow \mathbb{P} \text{ ROLES} \\
\forall u : \text{USERS} & \cdot \text{authorizedRoles}(u) \iff \\
& \{ r : \text{ROLES} \mid \exists p : \text{ROLES} \mid p \in \text{assignedRoles}(u) \land p \rightsquigarrow^* r \} \\
\text{authorizedUsers : ROLES} & \rightarrow \mathbb{P} \text{ USRERS} \\
\forall r : \text{ROLES} & \cdot \text{authorizedUsers}(r) \iff \\
& \{ u : \text{USERS} \mid r \in \text{authorizedRoles}(u) \}
\end{align*}
\]

We define some properties that must hold for a State to be consistent.
Now, let’s define what a State is in RBAC.

2.2 State in RBAC

2.2 State in RBAC

For a complete description of these constraints for consistency of a State, please refer to [6], Section 3.4.

2.3 Operations on State

2.3 Operations on State
### addRole

**ΔState**

\[ \text{role? : ROLES} \]

- \( \text{role?} \notin \text{roles} \)
- \( \text{roles}' = \text{roles} \cup \{\text{role}\} \)
- \( \text{cardinality}' = \text{cardinality} \cup \{\text{role}\} \rightarrow \infty \)

### rmRole

**ΔState**

\[ \text{role? : ROLES} \]

\[ \forall \ u : \text{USERS} \bullet \text{role?} \notin \text{assignedRoles}(u) \]

\[ \neg(\exists p : \text{ROLES} \bullet p \leftarrow \text{role?} \lor \text{role?} \leftarrow p) \]

\[ \neg(\exists p : \text{ROLES} \bullet (p, \text{role?}) \in \text{ssd}) \]

\[ \neg(\exists p : \text{ROLES} \bullet (p, \text{role?}) \in \text{dss}) \]

\[ \text{roles}' = \text{roles} \setminus \{\text{role}\} \]

\[ \text{cardinality}' = \{\text{role}\} \leftarrow \text{cardinality} \]

Before removing a role from assignedRoles, you need to make sure that there is another role which is assigned so that it can take care of the currently activeRoles of the user. Otherwise, the user will be in activeRoles even though he's not authorized by any role.

### addAssignment

**ΔState**

\[ \text{user? : USERS} \]

\[ \text{role? : ROLES} \]

\[ \forall \ u : \text{USERS} \bullet \text{role?} \notin \text{assignedRoles}(u) \]

\[ \forall p: \text{ROLES} \bullet \text{role?} \leftarrow p \Rightarrow p \notin \text{assignedRoles}(\text{user}) \]

\[ \forall p: \text{ROLES} \bullet p \in \text{assignedRoles}(\text{user}) \Rightarrow (p, \text{role?}) \notin \text{ssd} \]

\[ \forall p: \text{ROLES} \bullet \text{role?} \leftarrow p \Rightarrow \#(\text{authorizedUsers}(p)) < \text{cardinality}(p) \]

### rmAssignment

**ΔState**

\[ \text{user? : USERS} \]

\[ \text{role? : ROLES} \]

\[ \forall \ u : \text{users} \bullet \text{role?} \in \text{roles} \]

\[ \text{role?} \in \text{assignedRoles}(\text{user?}) \]

\[ \forall p : \text{ROLES} \bullet \text{role?} \leftarrow p \Rightarrow p \notin \text{assignedRoles}(\text{user}) \]

\[ \forall p : \text{ROLES} \bullet p \in \text{assignedRoles}(\text{user}) \Rightarrow (p, \text{role?}) \notin \text{ssd} \]

\[ \forall p : \text{ROLES} \bullet \text{role?} \leftarrow p \Rightarrow \#(\text{authorizedUsers}(\text{user})) \cup \text{authorizedUsers}(\text{r1}) \]

\[ \text{inherits}' = \text{inherits} \cup (\text{r1}, \text{r2}) \]
\[ \begin{align*} \Delta \text{State} \\
r1? : \text{ROLES} \\
r2? : \text{ROLES} \\
r1? \in \text{roles} \land r2? \in \text{roles} \\
r1? \not\subset r2? \\
\forall u : \text{USERS}; r : \text{ROLES} \Rightarrow \\
\quad \exists p : \text{ROLES}; \\
\quad \text{newinherits} : \text{ROLES} \leftrightarrow \text{ROLES} \Rightarrow \\
\quad p \in \text{assignedRoles}(u) \land \\
\quad \text{newinherits} = \text{inherits} \setminus (r1?, r2?) \land \\
\quad (r \in \bigcup \{n : \mathbb{N} \mid n \geq 1 \cdot \text{newinherits}^n(p)\}) \\
\text{inherits}' = \text{inherits} \setminus (r1?, r2?) \\
\end{align*} \]
MODIFICATIONS TO NIST

The basic NIST RBAC paper [6] does not define what ROLES are. It treats them as basic type and leaves it at that. For our purposes, they are used to define what operations a role is allowed to perform. This feature is added to State by means of the partial function allowedOps. We call the modified state, MState.

Now we define how we can add and remove operations and allowed operations to and from an MState.

### 3.1 Deciding Validity
5.1 Document Model

3.1 Deciding Validity

A user is allowed to perform an operation if there is a role which is allowed to perform that operation and the user belongs to active users of that role. We define a syntax for using this function. \( o \models u \) (read as \( o \) is valid for \( u \)) iff user \( u \) is allowed to perform the operation \( o \).

\[
\text{relation } o \models u \quad : \quad \text{OPERATIONS} \times \text{USERS}
\]

\[
\forall o : \text{OPERATIONS}, u : \text{USERS} \quad o \models u \iff \exists r : \text{ROLES} \quad r \in \text{roles} \quad o \in \text{allowedOps}(r) \quad \land r \in \text{activeRoles}(u)
\]

The initialization of \( State \) is given by the axiomatic definition. We only need to define \( \text{assignedRoles}, \text{users} \) and \( \text{roles} \) as an empty set. All the other sets and relations depend on the existence of one of these due to properties \( P1 \ldots P17 \) and \( State \) constraints. They are, therefore, automatically initialized to empty sets.

\[
\text{InitState} \; == \; \{ \text{assignedRoles} = \emptyset, \text{users} = \emptyset, \text{roles} = \emptyset \}
\]

In the next section, we add the definitions which enable us to use CRBAC in addition to RBAC.

Since the architecture of CRBAC depends on the Document Model of \textit{SECTET} for the definition of constraints, we need to define the document model before we can define CRBAC.

5.1 Document Model

The \textit{SECTET} architecture is based on UML. The Document Model describes different entities and how they are associated with each other. It includes different constructs such as association and generalization. Please refer to Figure 4. in [?] for an example of a Document Model.

This specification of the Document Model is based on the formalization of UML by Evans [5]. First, we need to define a class in terms of its attributes, and methods. Values will be useful when we define instantiation of classes to objects. Names are used to define roles of classes in associations and the names of associations themselves.

\[
[\text{ATTRIBUTES}, \text{NAME}, \text{ARGS}]
\]

\[
\text{BASICTYPE} \; ::= \; \text{nil} \mid \text{DATE} \mid \text{TIME} \mid \text{true} \mid \text{false} \mid \text{N} \mid \text{Z}
\]

\[
\text{METHODS} \; ::= \; (\text{NAME} \times \mathbb{P} \text{ARGS}) \rightarrow \text{BASICTYPEARGS} \; ::= \; \emptyset
\]

Association between classes is defined using associations and association ends. An association end describes the role of the class in the association and an association is composed of a name and two association ends.
Now, we can define the document model precisely.

\[
\begin{array}{ll}
\text{ASSOCIATION} & \text{name : NAME} \\
 & \text{c1, c2 : ASSOCIATIONEND} \\
 & \text{c1.roleName} \neq \text{c2.roleName}
\end{array}
\]

\[
\begin{array}{ll}
\text{ASSOCIATION} & \text{rolename : NAME} \\
 & \text{class : CLASS} \\
 & \text{mul : PN}
\end{array}
\]

We remove the possibility of circular inheritance using the second constraint.

\[
\forall c_1, c_2 : \text{CLASS} \bullet c_1 \sqsubset c_2 \Rightarrow \\
\forall c : \text{CLASS} \bullet \neg (c \sqsubseteq c)
\]

This takes care of the instances of the classes as well as instances just assign values to attributes.

To define objects, we need to define two constructs. One is a function which returns the objectID – a unique identifier which identifies an object of a class – and the other is links which gives links between instances of class.

\[
\begin{array}{ll}
\text{DOCU\-MENTMODEL} & \text{classes : } \mathbb{P} \text{ CLASS} \\
 & \text{assocs : } \mathbb{P} \text{ ASSOCIATION} \\
 & \text{superclass : CLASS }\rightarrow\text{ CLASS} \\
\forall c_1, c_2 : \text{CLASS} \bullet \text{superclass}(c_2) = c_1 \Rightarrow c_1 \sqsubseteq c_2 \\
\forall c : \text{CLASS} \bullet \neg (c \sqsubseteq c)
\end{array}
\]

\[
\begin{array}{ll}
\text{Instantiation} & \text{obj : OBJECTS} \\
 & \text{links : ASSOCIATION }\rightarrow (\text{ObjID } \times \text{ ObjID}) \\
 & \text{values : (ObjID } \times \text{ ATTRIBUTES) }\rightarrow \text{ BASTIC\-TYPE} \\
 & \text{objClass : ObjID }\rightarrow\text{ CLASS} \\
\forall o : \text{ObjID}; c : \text{CLASS} \bullet o \in \text{obj}(c) \Rightarrow \\
& \text{objClass}(o) = c \\
\forall as : \text{ASSOCIATION}; o_1 : \text{ObjID}, o_2 : \text{ObjID} \\
& \bullet (o_1, o_2) \in \\
& \text{links}(as) \Rightarrow \\
& \text{as.e1.class} = \text{objClass}(o_1) \wedge \text{as.e2.class} = \text{objClass}(o_2)
\end{array}
\]

The first constraint is self-explanatory and defines what objClass is. The second one means that a link can only be created between two objects if there is an association between the classes of the objects.
5.2 Defining CRBAC expressions

5.1.1 Minor Variables and Functions

Before using CRBAC state, we need to define a global variable which can return the value of the current user logged making the request. We call this the requestingUser. This value will be initialized by the system when a request is made. We also define a method for returning the objID of the object of a class created for the user making the request. Both of these operations are implementation dependant and cannot be described here.

requestingUser : USERS
userObj : (USERS × CLASS) → ObjID

The function pickOne, when given a set of BASICTYPE, returns a single element belonging to the set. If the set contains more than one elements, the function returns false. (It is not however called for sets which have more than one elements.)

∀ x : P BASICTYPE; y : BASICTYPE | #x = 1
∧ y ∈ x • pickOne(x) = y
∀ x : P BASICTYPE; y : BASICTYPE | #x ≠ 1 • pickOne(x) = false

5.1.2 Redefining Operations

In SECTET, constraints are dependant on the values passed to operations. So, we cannot leave operations as basic types as in RBAC. We redefine operations as such:

OPERATION
name : NAME
params : P NAME

An operation has a name and a list of parameters. An operation call assigns all values of the arguments to some basic type. These are the arguments passed to the operation during the call.

5.2 Defining CRBAC expressions

CRBAC adds access constraints to the basic RBAC model.

CRBAC describes expressions using a predicate language called SECTET-PL. A SECTET-PL expression (Ψ) is of two types: navigation expression (Θ) and predicate expression (Φ).

Ψ ::= Θ | Φ
Θ ::= BASICTYPE | ObjID | val⟨⟨ObjID×ATTRIBUTES⟩⟩ | opval⟨⟨NAME⟩⟩ | selectOne⟨⟨ObjID×NAME×Φ⟩⟩ | select⟨⟨ObjID×NAME×Φ⟩⟩ | funCall⟨⟨ObjID×NAME×P ARGS⟩⟩ | subjectMap⟨⟨CLASS⟩⟩
Φ ::= true | false | or⟨⟨Φ×Φ⟩⟩ | and⟨⟨Φ×Φ⟩⟩ | neg⟨⟨Φ⟩⟩ | eq⟨⟨Ψ×Ψ⟩⟩ | ls⟨⟨Θ×Θ⟩⟩ | gr⟨⟨Θ×Θ⟩⟩ | leq⟨⟨Θ×Θ⟩⟩ | geq⟨⟨Θ×Θ⟩⟩ | let⟨⟨VAR×Θ⟩⟩

The NAME in both select and selectOne refers to the name of the association end that the selection is referring to. NAME in funcall refers to the name of the function being called.

Where ∨, ∧, ¬, <, >, ≤, ≥ have the same meanings as their numerical and logical counterparts.

5.2.1 Evaluating SECTET Expressions

For the purpose of assigning values to a SECTET expression, we define an operator called called eval. For evaluating permission predicates, the relation defined in Section 5.5.2 is used. The evaluation of navigation expressions is done directly by the eval function.

relation eval ~
5.3 Defining PACs

A PAC is defined as a collection of operation, role and a predicate expression.

\[
\begin{array}{c}
\text{PAC} \\
\text{op} : \text{OPERATION} \\
\text{role} : \text{ROLE} \\
\text{pred} : \Phi
\end{array}
\]

The three types of PAC are defined using this same definition. Their semantics are made clear from their use.

The evaluations are self explanatory but require some clarifications at a few points.

\(\text{eval(val(o, a), s, c)}\) returns the value assigned to attribute \(a\) of object \(o\) in CRBACState \(s\) using the function \(\text{values}\) in \(s\).

\(\text{eval(opval(n), s, c)}\) returns the value passed by the user during the opcall \(c\) for the variable \(n\). This is given by the \(\text{vals}\) function of opcall \(c\).

In \text{select}, the predicate specifies that the evaluation of \text{select} navigation expression yields all objects which are linked to the object passed with an instantiation of the association end name passed. Specifically, all objects such that there exists an association which links them with the original object with the association end name passed to \text{select}. It also ensures that the predicate expression yields true.

\(\text{eval(funcall(o, n, as), s, c)}\) returns the value computed by calling method \(n\) with arguments \(as\) of object \(o\) in CRBACState \(s\). It first evaluates all navigation expressions passed as arguments and combines the results in a set and passes it to the method \(n\). The function \text{methods} in every class maps a method name and arguments to its value. This function is used here for computation of final result of the function call.

\(\text{eval(subjectMap(e), s, c)}\) returns the object of a class corresponding to the calling user. This is implemented through the minor functions described in Section 5.1.1
5.4 CRBAC State

Now, the MState can be refined using DOCUMENT-MODEL, Instantiation and PACs for CRBAC. We call this new state, the CRBACState.

\[
\begin{align*}
\text{CRBACState} & \quad \Delta \text{CRBACState} \\
\text{MState} & \quad p? : \text{PAC} \\
\text{DOCUMENT-MODEL} & \quad p? \notin \text{ddPACs} \\
\text{Instantiation} & \quad \text{ddPACs}' = \text{ddPACs} \cup \{p?\} \\
g\text{PACs} & : \mathbb{P} \text{ PAC} \\
\text{ddPACs} & : \mathbb{P} \text{ PAC} \\
n\text{PACs} & : \mathbb{P} \text{ PAC} \\
\forall p : \text{PAC} \quad p \notin ((\text{ddPACs} \cap \text{nPACs}) \cup (\text{ddPACs} \cap \text{gPACs}) \cup (\text{gPACs} \cap \text{nPACs})) \\
\end{align*}
\]

The only constraint for PACs is that a single PAC cannot be of any two types.

We define some operations for addition and removal of PACs from a state.

\[
\begin{align*}
\text{addGPAC} & \quad \Delta \text{CRBACState} \\
\text{CRBACState} & \quad p? : \text{PAC} \\
p? \notin \text{gPACs} & \quad \text{gPACs}' = \text{gPACs} \cup \{p?\} \\
\end{align*}
\]

We do not need to check if p? is already contained in another PAC collection because we already have this constraint in the basic CRBACState definition.

\[
\begin{align*}
\text{rmGPAC} & \quad \Delta \text{CRBACState} \\
\text{CRBACState} & \quad p? : \text{PAC} \\
p? \in \text{gPACs} & \quad \text{gPACs}' = \text{gPACs} \setminus \{p?\} \\
\end{align*}
\]

Similar operations are defined for ddPAC and nPAC:

\[
\begin{align*}
\text{addDDPAC} & \quad \Delta \text{CRBACState} \\
\text{CRBACState} & \quad p? : \text{PAC} \\
p? \notin \text{ddPACs} & \quad \text{ddPACs}' = \text{ddPACs} \cup \{p?\} \\
\end{align*}
\]

\[
\begin{align*}
\text{rmDDPAC} & \quad \Delta \text{CRBACState} \\
\text{CRBACState} & \quad p? : \text{PAC} \\
p? \in \text{ddPACs} & \quad \text{ddPACs}' = \text{ddPACs} \setminus \{p?\} \\
\end{align*}
\]

\[
\begin{align*}
\text{addNPAC} & \quad \Delta \text{CRBACState} \\
\text{CRBACState} & \quad p? : \text{PAC} \\
p? \notin \text{nPACs} & \quad \text{nPACs}' = \text{nPACs} \cup \{p?\} \\
\end{align*}
\]

\[
\begin{align*}
\text{rmNPAC} & \quad \Delta \text{CRBACState} \\
\text{CRBACState} & \quad p? : \text{PAC} \\
p? \in \text{nPACs} & \quad \text{nPACs}' = \text{nPACs} \setminus \{p?\} \\
\end{align*}
\]

5.5 Defining Satisfiability and Validity

In order to determine which operations are allowed to which users, we need to define two operators.

5.5.1 Constrained Validity

We defined the basic validity operator | in Section 3.1. Here we define constrained validity using the operator $\models_c$. We say that an operation o is valid with constrains in state s for a user u iff

1. the user belongs to a role which is in activeRoles
of that operation,

2. there is at least one general or data dependant constraint allowing that operation to that role and

3. there is no negative constraint prohibiting that role from performing that operation

This is written as \( (o, s) \models_c u \). Specifically, the relation \( \models_c \) is

\[
\models_c : (\text{OPCALL} \times \text{CRBACState}) \times \text{USERS}
\]

\[
\forall o : \text{OPCALL}, s : \text{CRBACState}; u : \text{USERS} \bullet (o, s) \models_c u \iff
(\exists r : \text{ROLES} \mid r \in s.\text{roles} \bullet
\begin{align*}
& o.\text{op} \in \text{allowedOps}(r) \land r \in \text{activeRoles}(u) \\
& (\exists p : \text{PAC} \mid p.\text{op} = o.\text{op} \land p.\text{role} = r) \\
& (p \in \text{gPACs} \lor p \in \text{ddPACs}) \land (o, s) \models_s p.\text{pred} \\
& \land (\neg \exists p : \text{PAC} \mid p.\text{op} = o \land p.\text{role} = r \bullet p \in \text{nPACs} \land (o, s) \models_s p.\text{pred})
\end{align*}
\]

5.5.2 Satisfiability

A CRBACState \( s \) is said to satisfy a predicate \( p \) if the predicate returns the value true when applied to that state. For this purpose, we define the satisfiability operator \( \models_s \). If \( s \models_s p \), we say that state \( s \) satisfies the predicate expression \( p \).

relation \( \models_s \)
relation \( \not\models_s \)

\[
\models_s : (\text{OPCALL} \times \text{CRBACState}) \times \Phi
\]

\[
\not\models_s : (\text{OPCALL} \times \text{CRBACState}) \times \Phi
\]

\[
\forall c : \text{OPCALL}; s : \text{CRBACState}; p : \Phi \bullet (s, c) \models_s p \iff p = \text{true}
\]

\[
\forall c : \text{OPCALL}; s : \text{CRBACState}; p : \Phi \bullet (s, c) \not\models_s p
\]

\[
\exists p_1, p_2 : \Phi \bullet (p = \text{or}(p_1, p_2)) \land ((s, c) \models_s p_1 \lor (s, c) \models_s p_2)
\]

\[
\exists p_1, p_2 : \Phi \bullet (p = \text{and}(p_1, p_2)) \land ((s, c) \models_s p_1 \land (s, c) \models_s p_2)
\]

\[
\exists p_0 : \Phi \bullet (p = \text{neg}(p_0)) \land ((s, c) \not\models_s p_0)
\]

\[
\exists p_1, p_2 : \Psi \bullet (p = \text{eq}(p_1, p_2)) \land
\begin{align*}
(p.\text{op} \in \text{allowedOps}(r) \land r \in \text{activeRoles}(u)) \\
\land (\exists p : \text{PAC} \mid p.\text{op} = o.\text{op} \land p.\text{role} = r \land (p \in \text{gPACs} \lor p \in \text{ddPACs}) \land (o, s) \models_s p.\text{pred})
\end{align*}
\]

\[
\exists c_0 : \text{VAR}; c_0 : \Theta; p_0 : \Phi \bullet
\begin{align*}
(p = \text{let}(c_0, c_0, p_0)) \land
\begin{align*}
& (o, s) \models_s p_0 \mid \text{eval}(c_0, s, c)/c_0) \\
& \land (s, c) \not\models_s p
\end{align*}
\end{align*}
\]

The semantics of these definitions are all evident except for \( \text{let} \). A \( \text{let} \) statement creates a lexical scope for the variable in a permission predicate expression. We use the default \textit{substitution} syntax of \( \Phi \) to accommodate this. Any appearance of the variable \( z_0 \) in permission predicate \( p_0 \) is replaced with the value given by evaluation of the navigation expression \( c_0 \) passed to the \( \text{let} \) expression. The resulting permission predicate is evaluated as any normal permission predicate.

The use of \textit{pickOne} is necessary in places where the comparison is being made because \textit{eval} returns a set of \textit{BASICTYPES}. They would contain only a single element which is returned by \textit{pickOne}. Only in \textit{let} is there a possibility (and indeed use) of multiple return values (due to \textit{select} type navigation expression).
usage in let and select is the reason eval was defined to return a set in the first place.

CONCLUSION

The paper formally proves the properties of RBAC and CRBAC in Z Notation and further shows how to share Healthcare documents using SECTET. The reason to show this formalization is due to the new era of decentralized network apps that brought a radical change in future Healthcare and Business apps. RBAC and CRBAC are the very important components of existing blockchain solutions. However, various architectures are available for blockchain, such as, public and private blockchain. CRBAC is required in private blockchain because a user in private blockchain shows his/her identity before getting eligibility to become part of the network.


