Correctness of the Sage algorithm
Enforcing Usage Constraints on Credentials for Web Applications

Jinwei Hu, Heiko Mantel, and Sebastian Ruhleder
1. Preliminaries

1.1. Principals

A principal is a uniquely identifiable entity in a context, i.e. someone who interacts and communicates with other entities in the same context. This includes but is not limited to people, companies, processes etc. Since principals must be distinguishable from each other, we use concepts of public-key cryptography to model the identity of a principal.

**Definition 1.1.** A principal is identified by a public key. Whoever can prove possession of the identity’s corresponding private key, is said to own the identity.

We do not concern with the cryptographic foundation of this notion, as long as it is kept consistent throughout the use of this paper. We assume a principal has sufficient capabilities to protect the secrecy of its private key.

**Definition 1.2.** Let Pub\((K_A)\) and Priv\((K_A)\) denote the public and private key of the principal \(K_A\), respectively.

One of the benefits of modeling an identity as a public key, is its ability to sign messages. Accordingly, when a group of principals wants to communicate with each other, they can use their secret private keys to sign messages.

**Definition 1.3.** A statement of the form \(K_A\) signs \(X\) is a signed message. It denotes the fact that the principal \(K_A\) has signed the message \(X\) with its private key.

At the moment, we leave the intuition of messages unspecified. However, when we use signed messages from now on, we assume them to be verifiable with the corresponding public key.

1.2. Policy Statements

In role-based access control systems, access to an object is restricted to subjects assigned to a specific role. While role-based access control concerns with role assignment, role authorisation, and permission authorisation, we assume a less complex scenario for simplicity: An object is protected by a role and every principal who can prove it is assigned to this role can gain access.

**Definition 1.4.** The statement \(R\) protects \(o\) denotes that an object \(o\) is protected by a role \(R\). Only principals who can prove to be assigned to the role \(R\) can access \(o\).
We use the $RT_0$ policy language [LWM03] to define roles and the relationships between them. As in the $RT$ framework, we assume every principal has a name space to define roles.

**Definition 1.5.** A role $K_A.R$ consists of a defining principal $K_A$ and a role term $R$. A role $K_A.R$ is said to belong to the name space of $K_A$.

From now on, we refer only to roles who are defined in the name space of a principal. In order to populate roles with principals, we use the following $RT_0$ policy statements.

**Definition 1.6.** An $RT_0$ policy statement has one of the following forms:

- A *simple membership* policy statement has the form
  \[ K_A.R \leftarrow K_D \]
  and says that the principal $K_D$ is assigned to the role $K_A.R$.

- A *simple containment* policy statement has the form
  \[ K_A.R \leftarrow K_B.R_1 \]
  and says that all principals assigned to the role $K_B.R_1$ are also assigned to the role $K_A.R$.

- A *linking containment* policy statement has the form
  \[ K_A.R \leftarrow K_A.R_1.R_2 \]
  and says that all principals assigned to the role $K_B.R_2$ with $K_B$ being assigned to the role $K_A.R_1$ are also assigned to the role $K_A.R$.

- An *intersection containment* policy statement has the form
  \[ K_A.R \leftarrow K_{B_1}.R_1 \cap \ldots \cap K_{B_n}.R_n \]
  and says that all principals that are assigned to the roles
  \[ K_{B_1}.R_1, \ldots, K_{B_n}.R_n \]
  are also assigned to the role $K_A.R$.

Given an arbitrary $RT_0$ policy statement $X \leftarrow Y$, we say $X$ is the head and $Y$ is the body of the policy statement.
1.3. Credentials

Since $RT_0$ policy statements declare how principals should be assigned to roles they cannot be declared arbitrarily. We use the intuition of credentials as signed to policy statements to enable principals to argue about roles.

Intuitively, a principal should be the only one able to assign principals to roles defined in its own name space. For example, only a company `compA` should be able to assign principals to its role `compA.Employee`. This leads to the following definition of credentials.

**Definition 1.7.** A statement $K_A$ signs $X$ is a credential if and only if

- $K_A$ is a principal,
- $X$ is an $RT_0$ policy statement of the form $K_A.R ← Y$.

When a principal creates a new credential, it can share this credential with other principals. For example, a principal $K_A$ issue the credentials

$$K_A.R ← K_B$$

and shares it with the principal $K_B$. Now, when $K_B$ wants to prove that it is assigned to the role $K_A.R$, it can use this credential to do so. Later on, we will introduce a notion of proofs to formalise this informal process of proving role assignments. Still, $K_B$ needs to be in possession of this credential (and possibly others) to construct proofs.

**Definition 1.8.** A set of credentials $C$ is called a credential context.

As a consequence, a principal can only construct proofs using credentials in his credential context.

1.4. Proofs over $RT_0$ Credentials

Consider the following two credentials $c_1$ and $c_2$:

$$c_1 = K_A \text{ signs } K_A.R ← K_B.R_1$$
$$c_2 = K_B \text{ signs } K_B.R_1 ← Alice$$

When a principal Alice is in possession of $c_1$ and $c_2$, she can argue to be assigned to the role $K_B.R_1$ (due to $c_2$) and therefore to be assigned to the role $K_A.R$ as well (due to $c_1$).

However, a reference monitor is not able to verify such an informal proof. Therefore, a formalisation of proofs over $RT_0$ credentials is necessary to allow a principal to use its credential context to construct proofs for arbitrary role assignments. To this end, we informally propose the following intuition of proofs.

A proof $P$ is a tuple $(p, r, c, s)$ where
• \( p \) is a principal,
• \( r \) is a role,
• \( c \) is an \( RT_0 \) credential,
• and \( s \) is a list of proofs called the list of subproofs of \( P \).

This intuition is motivated by the interdependency of \( RT_0 \) credentials. The base case is represented by simple member credentials: They directly state a role assignment. All other types of credentials, however, rely on other role assignments to work. A simple containment credential, for example, only grants a role assignment if someone has already proven to be assigned to the contained role. The recursive structure of proofs therefore captures this interdependency.

In order to provide a formal model of the intuitive deduction given above, we have constructed a set of inference rules to allow formal deductions on \( RT_0 \) credentials.

**Definition 1.9.** Let \( I = \{sm, sc, ic, lc\} \) denote the set of inference rules over \( RT_0 \) credentials where:

\[
\frac{K_A \text{ signs } K_A.R \leftarrow K_D}{K_D \text{ in } K_A.R} \quad \text{sm} \quad \frac{K_D \text{ in } K_B.R_1 \quad K_A \text{ signs } K_A.R \leftarrow K_B.R_1}{K_D \text{ in } K_A.R} \quad \text{sc}
\]

\[
\frac{K_D \text{ in } K_{B_1}.R_1 \quad \ldots \quad K_D \text{ in } K_{B_k}.R_k}{K_D \text{ in } K_A.R} \quad \text{ic}
\]

with \( c_{ic} = K_A \text{ signs } K_A.R \leftarrow K_{B_1}.R_1 \cap \ldots \cap K_{B_k}.R_k \)

\[
\frac{K_B \text{ in } K_A.R_1 \quad K_D \text{ in } K_B.R_2 \quad K_A \text{ signs } K_A.R \leftarrow K_A.R_1.R_2}{K_D \text{ in } K_A.R} \quad \text{lc}
\]

**1.5. Role Paths**

We have formalised proofs as recursive structures containing possibly non-empty sets of subproofs. With this structure in mind, we can generate a directed, acyclic graph of the role assignments in a proof.

**Definition 1.10.** A proof \((p, r, c, s)\) spans directed, acyclic role graph. This graph is generated as follows:

1. Set \( r \) as the current node.
2. If \( s = \emptyset \), mark the current node as a leaf. Otherwise, add every role \( r_i \) of every \( p_i \in s \) as a child node of the current node.
3. Repeat steps 1 and 2 for every \( r_i \) of every \( p_i \in s \).
We denote the role graph of a proof $P$ as $\text{graph}(P)$.

This graph provides a clear overview on all role assignments in a proof. Later on, we will formalise constraints on credentials. An intuitive approach for constraints might be to operate on the role assignments of a proof. The following definitions break down the role graph in several role paths.

**Definition 1.11.** A role path is a tuple $(r_0, \ldots, r_n)$ where $r_0, \ldots, r_n \in \text{Role}$

**Definition 1.12.** A proof $P$ induces a set of role paths $\text{paths}(P)$. Every role path $rp \in \text{paths}(P)$ is a path in $\text{graph}(P)$ from the root node to a leaf node.

Consider the proof $P$ we have seen before:

$$P = (\text{Alice}, K_A.R, c_1, \{(\text{Alice}, K_B.R_1, c_2, \emptyset)\})$$

Its role graph $\text{graph}(P)$ is:

$$K_A.R \quad | \quad K_B.R_1$$

Accordingly, the set of role paths $\text{paths}(P)$ is:

$$\text{paths}(P) = \{(K_A.R, K_B.R_1)\}$$

### 1.6. Constraints

A credential issuer cannot foresee the implications of issuing a credential, since she might not be aware of all credentials present in a given context. Therefore, *usage constraints* can be placed on credentials. A constraint limits the use of a credential to a specific purpose. We use non-deterministic finite automata to model constraints.

**Definition 1.13.** A constraint is modeled as a non-deterministic finite automaton, formally represented by a tuple $(Q, \Sigma, \delta, s_0, F)$, where

- $Q$ is a finite set of states,
- $\Sigma$ is a finite alphabet with $\Sigma \subseteq \text{Role}$,
- $\delta: S \times (\Sigma \cup \{\epsilon\}) \mapsto 2^S$ is a transition function,
- $s_0 \in S$ is the initial state,
- and $F \subseteq S$ is the set of accepting states.

Since the inputs of a constraint are roles, we define how to determine whether a constraint accepts a role path.
Definition 1.14. A constraint $(Q, \Sigma, \delta, s_0, F)$ accepts a role path 
\[ rp = (r_0, \ldots, r_n) \]
if a sequence of states $a_0, \ldots, a_k$ exists in $Q$ with the following conditions:

- $a_0 = s_0$,
- $a_{i+1} \in \delta(a_i, r_{i+1})$ for $i = 0, \ldots, n - 1$,
- and $a_k \in F$.

A proof induces a set of role paths and a constraint can accept a role path or not. The intuition of whether a proof satisfies a constraint or not is straightforward.

Definition 1.15. A proof $P$ satisfies a constraint $\text{con}$ if $\text{con}$ accepts every role path $rp \in \text{paths}(P)$.

Since constraints are meant to be placed on credentials, we have to extend the definition of credentials to support constraints.

Definition 1.16. A constrained credential is a statement of the form

\[ K_A \text{ signs } \langle X, \text{con} \rangle \]

where

- $K_A \text{ signs } X$ is a credential,
- and $\text{con}$ is a set of constraints.

From now on, when we refer to credentials we implicitly mean constrained credentials, since credentials without constraints simply set

\[ \text{con} = \emptyset \]

We do not redefine the set of inference rules to work with constrained credentials, since the constraints do not influence the deduction process. Whenever the inference rules are applied, a constrained credential appears in its non-constrained form. However, we have to adapt the validity of a proof to support constraints.

Definition 1.17. A proof $P = (p, r, c, s)$ induces a set of constraints $\text{Con}(P)$. We define the set $\text{Con}(P)$:

- Let $c$ be of the form
  \[ K_A \text{ signs } \langle X, \text{con} \rangle \]
  Then $\text{Con}(P) := \{ \text{con} \}$
- For every $P_i \in s$, let $\text{Con}(P) := \text{Con}(p) \cup \text{Con}(P_i)$. 

7
Definition 1.18. Given a principal \( p \), a role \( r \), and a set \( CS \) of credentials, we say a tuple \((p, r, c, \emptyset)\) is a base proof of \( p \) in \( r \) based on \( CS \) if \( c \) is a simple membership credential \( r \leftarrow p \). We say \((p, r, c, s)\) is a proof of \( p \) in \( r \) based on \( CS \) if either it is a base proof or \( s \) is a set of proofs based on \( CS \), called sub-proofs of \( P \), such that for any \((p_i, r_i, c_i, s_i) \in s\)

1. there exists \( c \in CS \) such that

\[
\begin{align*}
&\quad \text{\( p_1 \in r_1 \) \ldots \( p_k \in r_k \) \( c \in r \)} \quad \text{for some} \quad l \in \{sm, sc, ic, lc\} \quad \text{and} \\
&\quad \text{\( p \in l \)}
\end{align*}
\]

2. \((p_i, r_i, c_i, s_i)\) is also a proof based on \( CS \).

We say a proof \( P \) of \( p \) in \( r \) based on a set of credentials \( CS \) is semi-valid.

Definition 1.19. A proof \( P = (p, r, c, s) \) is valid if and only if

- \( P \) is semi-valid,
- every constraint \( con \in Con(P) \) accepts every role path \( rp \in paths(P) \).

We now have a formal model of credentials, constraints, and proofs. Whenever a principal constructs a proof to show a certain role assignment, a reference monitor can use the above mechanism to determine whether the proof is valid or not.

1.7. Conformity of Proofs

We assume a principal manages a finite set of \( RT_0 \) credentials. Accordingly, a principal can only refer to this finite set when constructing proofs. Although there might be other credentials allowing an easier construction process or even enabling one in the first place, it might be that the principal is not aware (and therefore not in possession of) this credential. In order to bind a proof to the finite set of credentials of its issuer, we introduce the notion of conformity between credential sets and proofs.

A proof contains (or uses) a finite set of credentials. We define a function to return this finite set of credentials and define the concept of conformity between proofs and credential sets.

Definition 1.20. Given a proof \( P = (p, r, c, s) \), let \( credentials(P) \) denote the set of all credentials used in \( P \). We define \( credentials(P) \) inductively as:

\[
\text{credentials}((p, r, c, s)) := \{ c \} \cup \left( \bigcup_{P_i \in s} \text{credentials}(P_i) \right)
\]

Definition 1.21. We say a proof \( P \) is conform to a set of \( RT_0 \) credentials \( C \) if and only if \( credentials(P) \subseteq C \).
2. The Sage Algorithm

A set of $RT_0$ credentials might allow the construction of several proofs for a role assignment. Although all such proofs share a common goal (i.e. proving a role assignment for a specific role and principal), they might differ in structure and complexity. For example, given two proofs $P_1$ and $P_2$, a principal might want to choose the proof that discloses the least credentials. Another principal might choose the proof that is smallest in size.

As a consequence, it is desirable to find an algorithm that outputs all possible proofs over a set of $RT_0$ credentials given a role and a principal. In this paper, we provide such an algorithm, called the SAGE algorithm.

The SAGE algorithm can be found in Appendix A. The following theorem reflects the intuition of the algorithm.

**Theorem 2.1.** Based on a set of credentials $C$, the function call

$$\text{SAGE-Prove}(C, r, p)$$

returns the set of all proofs over $C$ proving that the principal $p$ is assigned to the role $r$. 
3. Correctness Theorem

In this chapter, we will show that the inference rules reflect our intuition of $RT_0$ credentials faithfully. Then, we will formalise correctness as a property of the SAGE algorithm.

3.1. Faithfulness of Inference Rules

The idea behind $RT_0$ credentials is to show role assignments and the relationships between different roles. We can classify the four types of $RT_0$ credentials into those that directly assign a principal to a role (simple member credentials) and those that rely on other roles to assign principals to a role (simple containment, linking containment, and intersection containment credentials). Credentials of the first category do not require previous role assignments while those of the second category do.

In the following, we will compare the four types of $RT_0$ credentials with their associated inference rules and argue why the relationships between them capture this idea of requirements faithfully.

- A credential
  \[ c_1 = K_A \text{ signs } K_A.R \leftarrow K_D \]
  is a simple member credential and assigns the principal $K_D$ to the role $K_A.R$. The $sm$ inference rule
  \[
  \frac{c_1}{KD \text{ in } K_A.R} \quad \text{sm}
  \]
  captures the idea of a direct role assignment faithfully, since it directly allows us to deduce the role assignment from simple member credentials.

- A credential
  \[ c_2 = K_A \text{ signs } K_A.R \leftarrow K_B.R_1 \]
  is a simple containment credential and defines the role $K_A.R$ to contain the role $K_B.R_1$. The $sc$ inference rule
  \[
  \frac{K_D \text{ in } K_B.R_1 \quad c_2}{KD \text{ in } K_A.R} \quad \text{sc}
  \]
  captures the idea of a contained role assignment faithfully, since it requires a principal to be assigned to the role $K_B.R_1$ in order to deduce its assignment to the role $K_A.R$. 

• A credential

\[ c_3 = K_A \text{ signs } K_A.R \leftarrow K_A.R_1 . R_2 \]

is a linking containment credential and defines the role \( K_A.R \) to contain every role \( K_B . R_2 \) defined by every principal \( K_B \) that is assigned to the role \( K_A.R_1 \). The \( lc \) inference rule

\[
\frac{K_B \text{ in } K_A.R_1 \quad K_D \text{ in } K_B.R_2 \quad c_3 \quad K_D \text{ in } K_A.R}{lc}
\]

captures the idea of a linked role assignment faithfully, since it requires a principal to be assigned to a role \( K_B.R_2 \) and the principal \( K_B \) to be assigned to the role \( K_A.R_1 \) in order to deduce its assignment to the role \( K_A.R \).

• A credential

\[ c_4 = K_A \text{ signs } K_A.R \leftarrow K_B_1 . R_1 \cap \ldots \cap K_B_k . R_k \]

is an intersection containment credential and defines the role \( K_A.R \) to contain every principal that is assigned to the roles \( K_B_1 . R_1 , \ldots , K_B_k . R_k \). The \( ic \) inference rule

\[
\frac{K_D \text{ in } K_B_1 . R_1 \quad \ldots \quad K_D \text{ in } K_B_k . R_k \quad c_4 \quad K_D \text{ in } K_A.R}{ic}
\]

captures the idea of an intersected role assignment faithfully, since it requires a principal to be assigned to every role \( K_B_1 . R_1 , \ldots , K_B_k . R_k \) in order to deduce its assignment to the role \( K_A.R \).

The premises of the inference rules are a credential of the associated type and the possibly required role assignments needed for that type. Accordingly, the inference role \( sm \) for simple member credentials only lists a simple member credential in its premises, while all other inference rules list the required role assignments alongside a credential of the type.

### 3.2. Correctness of the Algorithm

An informal way to characterise the correctness of the SAGE algorithm is: Based on a set of \( RT_0 \) credentials, a role, and a principal, the SAGE algorithm outputs all possible and valid proofs that

• show that the principal is assigned to the role,

• and only use credentials from the supplied set of \( RT_0 \) credentials.

First, we define the set of all possible proofs conform to a set of \( RT_0 \) credentials:

**Definition 3.1.** For any proof \( P_i \) that can be constructed from a given set of \( RT_0 \) credentials \( C \), \( P_i \in Proofs(C, r, p) \) if and only if

• \( P_i \) is valid and conform to \( C \),
• and $P_i$ has the form $P_i = (p, r, c_i, s_i)$.

In other words, if there is a valid proof $P$ conform to $C$ showing that a principal $p$ is assigned to the role $r$, then $P \in \text{Proofs}(C, r, p)$. With this definition in place, we can now formalise the correctness of the SAGE algorithm:

**Theorem 3.2.** Given a set of $RT_0$ credentials $C$, a role $r$, and a principal $p$, the assumption

$$\text{SAGE-Prove}(C, r, p) = \text{Proofs}(C, r, p)$$

holds.
4. Understanding the Sage Algorithm

This chapter provides an informal description of the mechanics of the Sage algorithm and a line by line explanation of the pseudocode, which can be found in the appendix.

4.1. Informal Description

One requirement when determining whether a proof \( P = (p, r, c, s) \) is valid or not is checking the applicability of one of the four inference rules. Since the conclusion of the inference rules must be the role assignment \( p \text{ in } r \), the premises must consist of the credential \( c \) and all subproofs \( s_i \in s \) and must match one of the four inference rules \( sm, sc, ic, \) or \( lc \). Consequently, the type of the credential \( c \) directly influences what proofs must be present in \( s \) for this to be possible.

The Sage algorithm utilizes this dependency between \( c \) and \( s \) to construct proofs recursively: When a role assignment \( p \text{ in } r \) should be proven, select all credentials with \( r \) as the head role, extract for every such credential the role assignments that must be proven in order to satisfy the appropriate inference rule and then call the Sage algorithm again with the same principal for every such role assignment. If all necessary role assignments can be proven, construct (and later on return) a proof with the recursively constructed proofs and the associated, previously selected credential.

4.1.1. The Nodes Environment

The algorithm operates on a set of credentials \( C \). Since for every role assignment \( p \text{ in } r \), every credential in \( C \) with \( r \) as the head role must be checked, it is necessary to group credentials with the same head role together.

The algorithm therefore introduces nodes. A node is a container for credentials with the same head role. If a node contains only credentials with the head role \( r \), the node is said to be associated to this role. Before the algorithm starts to construct proofs, the set of credentials \( C \) is converted to a set of nodes so that every credential \( c \in C \) is added to exactly one node, i.e. the node associated to its head role.

The concept of nodes simplifies the way the different credentials in a set \( C \) can be accessed.

4.1.2. Enforcing Constraints

Since finding a matching inference rule only ensures semi-validity, the algorithm also has to enforce the constraints of the credentials used to construct a certain proof. To this end, two distinct checks are performed inside the algorithm:
1. The initial constraint check inspects the current role path towards whether all accumulated constraints can still reach an accepting state from it. Every time the algorithm is called recursively, both the role path and set of constraints are updated based on the selected credential.

2. The final constraint check filters the created set of proofs for those proofs who are valid, i.e. whose constraints accept every role path of them.

In combination, both constraint checks accomplish different goals: The initial constraint check in order to minimize the creation of semi-valid but not valid proofs, while the final constraint check then enforces all constraints which could have not been checked due to the recursive nature of the algorithm.

4.2. Explaining the Pseudocode

We now explain the algorithm in Appendix A. The main entry point of the algorithm is the SAGE-PROVE function: It expects a set of $RT_0$ credentials, a role, and a principal and returns a set of proofs showing the provided principal is assigned to the provided role. In line 1, the set of credentials is used to create a node environment, which is then stored in the variable nodes. In line 2, the variable constraints is initialized as an empty set, since the accumulated constraints at this point do not contain anything. Finally, in line 3, the SAGE-PROVE function passes on the appropriate parameters to the CONSTRUCT-PROOFS function, which then effectively computes the set of proofs (therefore the result of this computation is directly returned).

4.2.1. CREATE-NODES

This function expects one parameter, a set of $RT_0$ credentials, and creates a node environment. It traverses the set of credentials and performs the following steps for every credential $c$:

1. If there already exists a node associated to the role $\text{Head}(c)$, $c$ is added to that node.

2. Otherwise, a new node is created and then $c$ is added to that node.

Finally, the set of created nodes is returned.

4.2.2. CONSTRUCT-PROOFS

First, the provided role path (which is initially empty) is extended by the provided role parameter in line 1. This extended role path serves as a trace at which point the algorithm is in relation to the overall structure of the constructed proofs. In lines 2 to 4, this extended role path is used to check whether all constraints in the provided set constraints can still reach an accepting state from it. If not, at least one of the
constraints cannot reach an accepting state, which makes all further proofs constructed in this evaluation invalid. Therefore, computation is stopped in this case.

In line 5, the computation is stopped if there is no node associated to the provided role, since if there does not exist such a node, there are no credentials available to construct further proofs. Otherwise, the associated node is then assigned to the variable node in line 7 and the set proofs is initialized as an empty set in line 8. The set proofs serves as a container for proofs and is finally returned in line 22 after being filtered to contain only valid proofs (final constraint check).

In lines 9 to 10, all simple member credentials of node are traversed and a proof is added to the set proofs for every credential which shows the provided principal is assigned to the provided role.

Once all proofs using simple member credentials have been added, the set of credentials of node is traversed in line 11 for every credential c. Here, first the variable sets is initialized as an empty set in line 12. This variable serves as a container for sets of subproofs. After the distinction of the type of c, the variable sets is supposed to contain all sets of subproofs that, in combination with c, allow the construction of (semi-)valid proofs. This is reflected in line 21, in which for every set in sets a new proof is added to proofs using c.

Since we know the set of subproofs directly depends on the type of the credential, the type of c is distinguished in lines 14 to 19. In this part, the three remaining types of credentials (simple, intersection, or linking containment) are handled by three distinct helping functions. Each of these functions returns a set of sets of subproofs, which can be used to construct proofs in combination with c.

4.2.3. Handle-Simple-Containment

If c is a simple containment credential, this function is evaluated. It simply calls Construct-Proofs with a new role parameter: The contained role of the credential c. Every such proof is then later on used in Construct-Proofs in combination with c to add (semi-)valid proofs to the set proofs.

4.2.4. Handle-Intersection-Containment

If c is an intersection containment credential, this function is evaluated. In lines 3 to 4, for every role in the intersection of the body of c proofs are created by evaluating Construct-Proofs and then assigned to the variable ip. Finally, in lines 5 to 7, the cartesian product of all proofs in ip is calculated and returned in line 8.

4.2.5. Handle-Linking-Containment

If c is a linking containment credential, this function is evaluated. First, in line 4, all principals who define a role using the linked role term are traversed. For each such principal, Construct-Proofs is evaluated in line 5 to calculate proofs showing the principal is assigned to the defining role of c, which are assigned to the variable dp. If there exist proofs of this form, i.e. if dp is non-empty, Construct-Proofs is evaluated
again for the actual principal and the linked role, and the result is assigned to the variable $lp$. Then, in line 8, the cartesian product of $dp$ and $lp$ is calculated and later on returned.
5. Proving Correctness

By definition, a proof $P = (p, r, c, s)$ is a recursive structure: The set $s$ might contain other proofs, which in turn might contain other proofs, and so on. As a result, a proof can be displayed as a rooted tree. For example, the proof

$$P_{i,1} = (p_{i,1}, r_{i,1}, c_{i,1}, \{P_{i+1,1}, \ldots, P_{i+1,k}\})$$
with $P_{i+1,1} = (p_{i+1,1}, r_{i+1,1}, c_{i+1,1}, \{P_{i+2,1}, \ldots, P_{i+2,l}\})$

generates the following tree:

$$\begin{array}{c}
P_{i,1} \\
P_{i+1,1} & \cdots & P_{i+1,k} \\
P_{i+2,1} & \cdots & P_{i+2,l} \end{array}$$

We can formalise this intuition by defining the rooted tree induced by a proof:

**Definition 5.1.** A proof $P = (p, r, c, s)$ induces a rooted tree, $\text{tree}(P)$, as follows:

- Set $P$ as the root node of $\text{tree}(P)$.
- For every $P_i \in s$, add $\text{tree}(P_i)$ as a child of the root node.

An important concept of trees is their height, i.e. the length of the longest path from the root to a leaf node.

**Definition 5.2.** Let $\text{tree}(P)$ denote the rooted tree induced by a proof $P = (p, r, c, s)$. We define the height of $\text{tree}(P)$ as

$$\text{height}(\text{tree}(P)) = \begin{cases} 0 & \text{if } s = \emptyset \\ 1 + \max(\text{height}(\text{tree}(P_1)), \ldots, \text{height}(\text{tree}(P_k))) & \text{if } s = \{P_1, \ldots, P_k\} \end{cases}$$

where $\max$ returns the maximum of the supplied integer parameters.

From now on, we use $\text{height}(P)$ as a shorthand for $\text{height}(\text{tree}(P))$. For example, $\text{height}(P_{i,1}) = 2$.

**Case 1.** Let $P \in \text{Proofs}(C, r, p)$ denote a proof of the form $P = (p, r, c, s)$. We will use structural induction and show that CONSTRUCT-PROOFS returns $P$ for
• \( \text{height}(P) \leq K \) with \( K = 1 \),

• and \( \text{height}(P) = K + 1 \).

First, we assume \( \text{height}(P) \leq K \) with \( K = 1 \). Since \( c \) is an \( RT_0 \) credential, it must be either a simple member, simple containment, intersection containment, or linking containment credential. We distinguish these four cases.

**Simple Member**

If \( c \) is a simple member credential, it must be of the form

\[
c = K_A \text{ signs } \langle r \leftarrow p, \text{con} \rangle
\]

with \( r \) being a role in the namespace of \( K_A \). Furthermore, we make the following observations:

• Since \( P \) is a valid proof,
  • the constraint \( \text{con} \) accepts every role path of \( P \),
  • and \( P \) can be used to construct the inference

\[
\frac{c}{p \text{ in } r \sm}
\]

• Since the inference does not require any other premise than the credential \( c_i \), the set of subproofs \( s \) must be empty, i.e. \( s = \emptyset \).

• Since \( s = \emptyset \),
  • the height of \( P \) is \( \text{height}(P) = 0 \) and therefore \( \text{height}(P) \leq K \) with \( K = 1 \),
  • and the set of role paths is \( \text{paths}(P) = \{(r)\} \).

When evaluating \( \text{CONSTRUCT-PROOFS}(r, p, \text{nodes}_C, (), \emptyset) \), in line 1 the current role path \( (()) \) is extended with the role \( r \). This results in a role path \( r' = (r) \). Since \( \text{constraints} = \emptyset \), the check in lines 2 to 4 succeeds.

Let \( n = (r, m, cr) \) denote the node in \( \text{nodes}_C \) bound to the role \( r \). Since \( c \in C \), the node environment \( \text{nodes}_C \) is non-empty and contains the node \( n \). Line 7 assigns this node to the \text{node} variable. Due to its type, \( c \) is contained in the set of members \( m \) in the node \( n \) and therefore also in the set of members of \text{node}. In line 9, \( c \) is found in \( m \) and bound to the variable \( c \), since \( (c, p) \in m \) due to the definition of the node environment.

For \( c = c \), a new proof is added to the variable \text{proofs} in line 10. This proof has the structure

\[
\text{NEW-PROOF}(p, r, c, \emptyset)
\]

which is equal to \( P \) (according to the definition of \text{NEW-PROOF} and \( c = c \)). We now know that \( P \in \text{proofs} \).
We can skip lines 11 to 21, since \( c \) is not an element of the set of credentials in \( n \) (due to its type). Finally, the variable \( \textit{proofs} \) is filtered with \textsc{Filter-Valid-Proofs} in line 22.

Since we know that the constraint \( \text{con} \) of \( c \) accepts \( P \) and \( P \in \textit{proofs} \) and the function \textsc{Filter-Valid-Proofs} returns all proofs that satisfy all constraints, we can follow that

\[
P \in \textsc{Construct-Proofs}(r, p, \textit{nodes}_{\text{C}}, (\), \emptyset)
\]

**Simple Containment**

If \( c \) is a simple containment credential, it must be of the form

\[
c = K_A \text{signs } (r \leftarrow r_1, \text{con})
\]

with \( r \) being a role in the namespace of \( K_A \) and \( r_1 \) being an arbitrary role. Furthermore, we make the following observations:

- Since \( P \) is a valid proof,
  - the constraint \( \text{con} \) accepts every role path of \( P \),
  - and \( P \) can be used to construct the inference
    \[
    \begin{array}{c}
    p \text{ in } r_1 \text{ c} \\
    \hline
    p \text{ in } r \text{ sc}
    \end{array}
    \]

- Since the inference requires \( p \) to be assigned to the role \( r_1 \), the set of subproofs \( s \) must contain a proof for this assignment, i.e. \( s = \{P_y\} \) with \( P_y = (p, r_1, c_1, s_1) \).

- Since \( \text{height}(P) \leq K \) with \( K = 1 \), \( s_1 = \emptyset \) (otherwise \( \text{height}(P) > K \)). Therefore, \( c_1 \) must be a simple member credential (all other types of credentials require more premises in their inferences and therefore non-empty sets of subproofs).

- The set of role paths is \( \text{paths}(P) = \{(r), (r,r_1)\} \).

When evaluating \textsc{Construct-Proofs}(r, p, \textit{nodes}_{\text{C}}, (\), \emptyset), in line 1 the current role path \((())\) is extended with the role \( r \). This results in a role path \( r' = (r) \). Since \( \text{constraints} = \emptyset \), the check in lines 2 to 4 succeeds.

Let \( n = (r, m, c_r) \) denote the node in \( \textit{nodes}_{\text{C}} \) bound to the role \( r \). Since \( c \in C \), the node environment \( \textit{nodes}_{\text{C}} \) is non-empty and contains the node \( n \). Line 7 assigns this node to the \textit{node} variable. Due to its type, \( c \) is contained in the set of credentials \( c_r \) in the node \( n \) and therefore also in the set of credentials of \( \text{node} \). Lines 9 to 10 are skipped, since \( c \) is not a simple member credential.

In line 11, \( c \) is selected, since it is contained in the set of credentials of \( \text{node} \). After \( c \) is selected, the so far empty set of \textit{constraints} is extended with the constraint \( \text{con} \) of \( c \) and assigned to the variable \textit{constraints}' in line 13. Since \( c \) is a simple
containment credential, the function \texttt{HANDLE-SIMPLE-CONTAINMENT} is evaluated in line 15.

By definition, \texttt{HANDLE-SIMPLE-CONTAINMENT} simply evaluates

\[
\texttt{CONSTRUCT-PROOFS}(r_1, p, \texttt{nodes}_C, (r), \{\textit{con}\})
\]

since \(r' = (r)\) and \(\text{constraints}' = \{\textit{con}\}\). We have show that this evaluation returns \(P_y\). We make the following observations.

- Since \(\text{height}(P) = 1\), the height of \(P_y\) must be \(\text{height}(P_y) = 0\) and therefore \(c_1\) must be a simple member credential.
- The previous case (simple member) shows that \texttt{CONSTRUCT-PROOFS} returns proofs of the height 0 correctly for rolepath = () and constraints = \(\emptyset\). Therefore we only have to show that the initial constraint check succeeds for the new rolepath and constraints parameters.

In this evaluation of \texttt{CONSTRUCT-PROOFS} the supplied role path parameter \((r)\) is extended with the role \(r_1\), thereby forming the role path \((r, r_1)\). Now, constraints contains only one constraint, namely \textit{con} of \(c\). Since \(P\) is valid, we know that \textit{con} accepts \((r, r_1)\) and therefore the initial constraint check in lines 2 to 4 succeeds. Therefore \(P_y\) is returned by this evaluation.

In the original evaluation, \texttt{HANDLE-SIMPLE-CONTAINMENT} returns \(\{P_y\}\) in a set of proofs and this set of proofs is assigned to the variable \texttt{sets} in line 15. Accordingly, \(P_y \in \texttt{sets}\).

Finally, in line 21 the set \(\{P_y\}\) is used in the construction of a new proof of the form

\[
\texttt{NEW-PROOF}(p, r, c, \{P_y\})
\]

which is equal to \(P\) (according to the definition of \texttt{NEW-PROOF} and \(c = c\)). We now know that \(P \in \texttt{proofs}\).

Since we know that the constraints \textit{con} and \texttt{Con}(c_1) of \(c\) and \(c_1\) accept \(P\) and \(P \in \texttt{proofs}\) and the function \texttt{FILTER-VALID-PROOFS} returns all proofs that satisfy all constraints, we can follow that

\[
P \in \texttt{CONSTRUCT-PROOFS}(r, p, \texttt{nodes}_C, (\), \emptyset)\]

**Intersection Containment**

If \(c\) is an intersection containment credential, it must be of the form

\[
c = K_A \texttt{signs} (r \leftarrow r_1 \cap \ldots \cap r_k, \textit{con})
\]

with \(r\) being a role in the namespace of \(K_A\) and \(r_1, \ldots, r_k\) being arbitrary roles. Furthermore, we make the following observations:

- Since \(P\) is a valid proof,
  - the constraint \textit{con} accepts every role path of \(P\),
– and $P$ can be used to construct the inference

$$\begin{array}{c}
  p \text{ in } r_1 \ldots p \text{ in } r_k c \text{ in } r \\
\end{array}$$

- Since the inference requires $p$ to be assigned to the roles $r_1$ to $r_k$, the set of subproofs $s$ must contain proofs for these assignments, i.e. $s = \{P_1, \ldots, P_k\}$ with $P_1 = (p, r_1, c_1, s_1)$ to $P_k = (p, r_k, c_k, s_k)$.
- Since $\text{height}(P) \leq K$ with $K = 1$, $s_1 = \emptyset, \ldots, s_k = \emptyset$ (otherwise $\text{height}(P) > K$). Therefore, $c_1, \ldots, c_k$ must be a simple member credentials (all other types of credentials require more premises in their inferences and therefore non-empty sets of subproofs).
- The set of role paths is $\text{paths}(P) = \{(r), (r, r_1), \ldots, (r, r_k)\}$.

When evaluating $\text{Construct-Proofs}(r, p, \text{nodes}_C(), \emptyset)$, in line 1 the current role path $()$ is extended with the role $r$. This results in a role path $r' = (r)$. Since $\text{constraints} = \emptyset$, the check in lines 2 to 4 succeeds.

Let $n = (r, m, \text{cr})$ denote the node in $\text{nodes}_C$ bound to the role $r$. Since $c \in C$, the node environment $\text{nodes}_C$ is non-empty and contains the node $n$. Line 7 assigns this node to the $\text{node}$ variable. Due to its type, $c$ is contained in the set of credentials $\text{cr}$ in the node $n$ and therefore also in the set of credentials of $\text{node}$. Lines 9 to 10 are skipped, since $c$ is not a simple member credential.

In line 11, $c$ is selected, since it is contained in the set of credentials of $\text{node}$. After $c$ is selected, the so far empty set of $\text{constraints}$ is extended with the constraint $\text{con}$ of $c$ and assigned to the variable $\text{constraints}'$ in line 13. Since $c$ is an intersection containment credential, the function $\text{HANDLE-INTERSECTION-CONTAINMENT}$ is evaluated in line 17.

When evaluating $\text{HANDLE-INTERSECTION-CONTAINMENT}$, for every role in the body of $c$ the function $\text{Construct-Proofs}$ is evaluated and the returned set of proofs added to the set $\text{ip}$ in lines 3 and 4. Now, since

- $P_1, \ldots, P_k$ all have a height of 0, i.e. $\text{height}(P_1) = \ldots = \text{height}(P_k) = 0$,
- and the role paths in this evaluation of $\text{Construct-Proofs}$

are $(r, r_1), \ldots, (r, r_k)$, respectively;

the returned sets contain $P_1, \ldots, P_k$. This is due to the fact that the function $\text{Construct-Proofs}$ behaves correctly for a height of $K = 0$ and the respective role paths are, by definition of the validity of $P$, accepted by the constraint $\text{con}$ (which allows the initial constraint check to succeed for every $(r, r_i)$).

We know that for every proof $P_i$ with $i \in \{1, \ldots, k\}$ there exists a set $X \in \text{ip}$ with $P_i \in X$. Lines 5 to 7 simply construct all possible combinations of sets of the sets in $\text{ip}$. Therefore, the set $\{P_1, \ldots, P_k\}$ is added to the variable $\text{sets}$ in line 7. The
variable sets is then returned by \texttt{Handle-Intersection-Containment} in line 8.

This returned set of sets of subproofs is then assigned to the sets variable in line 17 of the original evaluation of \texttt{Construct-Proofs}.

Finally, in line 21 the set \( \{P_1, \ldots, P_k\} \) is used in the construction of a new proof of the form

\[
\text{New-Proof}(p, r, \{P_1, \ldots, P_k\})
\]

which is equal to \( P \) (according to the definition of \texttt{New-Proof} and \( c = c \)). We now know that \( P \in \text{proofs} \).

Since we know that the constraints \( con, Con(c_1), \ldots, Con(c_k) \) of \( c, c_1, \ldots, c_k \) accept \( P \) and \( P \in \text{proofs} \) and the function \texttt{Filter-Valid-Proofs} returns all proofs that satisfy all constraints, we can follow that

\[
P \in \text{Construct-Proofs}(r, p, \text{nodes}_C, ()), \emptyset)
\]

**Linking Containment**

If \( c \) is a linking containment credential, it must be of the form

\[
c = K_A \text{ signs } \langle r \leftarrow K_A.R_1.R_2, con \rangle
\]

with \( r \) being a role in the namespace of \( K_A \), \( R_1 \) being a role in the namespace of \( K_A \), and \( R_2 \) being an arbitrary role. Furthermore, we make the following observations:

- Since \( P \) is a valid proof,
  - the constraint \( con \) accepts every role path of \( P \),
  - and \( P \) can be used to construct the inference

\[
\begin{array}{c}
p \text{ in } K_B.R_2 \hfill K_B \text{ in } K_A.R_1 \hfill c \text{ in } l_c
\end{array}
\]

- Since the inference requires \( p \) to be assigned to a role \( K_B.R_2 \) for a principal \( K_B \) who is assigned to the role \( K_A.R_1 \), the set \( s \) contains two proofs: \( P_l = (p, K_B.R_2, c_l, s_l) \) and \( P_d = (K_B, K_A.R_1, c_d, s_d) \).

- Since \( \text{height}(P) = 1 \), both \( P_l \) and \( P_d \) must have a height of \( \text{height}(P_l) = \text{height}(P_d) = 0 \) and therefore both credentials \( c_l \) and \( c_d \) must be simple member credentials.

When evaluating \texttt{Construct-Proofs}(r, p, \text{nodes}_C, ()), \emptyset), in line 1 the current role path (()) is extended with the role \( r \). This results in a role path \( r' = (r) \). Since \( \text{constraints} = \emptyset \), the check in lines 2 to 4 succeeds.

Let \( n = (r, m, cr) \) denote the node in \( \text{nodes}_C \) bound to the role \( r \). Since \( c \in C \), the node environment \( \text{nodes}_C \) is non-empty and contains the node \( n \). Line 7 assigns this node to the node variable. Due to its type, \( c \) is contained in the set of credentials
cr in the node n and therefore also in the set of credentials of node. Lines 9 to 10 are skipped, since c is not a simple member credential.

In line 11, c is selected, since it is contained in the set of credentials of node. After c is selected, the so far empty set of constraints is extended with the constraint con of c and assigned to the variable constraints in line 13. Since c is a linking containment credential, the function HANDLE-LINKING-CONTAINMENT is evaluated in line 19.

In the evaluation of HANDLE-LINKING-CONTAINMENT, for every principal that defines a role with a role term matching the linking role of c (line 4), the function CONSTRUCT-PROOFS is evaluated in line 5 to prove that this principal is member of the defining role of c. Since P is valid and c is a linking containment credential, the set s must contain two proofs: Pd, proving that a principal KA,R1 is member of the defining role KA,R1, and Pl, proving that the principal p is a member of the linked role KB,R2. Since P has a height of height(P) ≤ K with K = 1, the height of Pd and Pl must be 0. We have shown that CONSTRUCT-PROOFS behaves correctly for proofs with a height of 0, therefore both the construction in line 5 and in line 7 (proving the assignment of p to the linked role) return sets containing Pd and Pl, respectively.

We now know that Pd ∈ dp and Pl ∈ lp. In line 8 of the evaluation of the HANDLE-LINKING-CONTAINMENT function, the cartesian product of both these set is constructed and for every combination added to the sets variable. Accordingly, a set {Pd, Pl} is contained within sets and therefore returned.

In the original evaluation of CONSTRUCT-PROOFS, the returned set of sets of subproofs is then assigned to the sets variable in line 19.

Finally, in line 21 the set {Pd, Pl} is used in the construction of a new proof of the form

\[
\text{NEW-PROOF}(p, r, c, \{Pd, Pl\})
\]

which is equal to P (according to the definition of NEW-PROOF and c = c). We now know that P ∈ proofs.

Since we know that the constraints con and Con(cl) of c and cl accept P and P ∈ proofs and the function FILTER-VALID-PROOFS returns all proofs that satisfy all constraints, we can follow that

\[
P \in \text{CONSTRUCT-PROOFS}(r, p, \text{nodes}_C, (), \emptyset)
\]

We have shown that P ∈ CONSTRUCT-PROOFS(r, p, nodes_C, (), \emptyset) for every P with height(P) ≤ K with K = 1. Accordingly, from now on we assume CONSTRUCT-PROOFS returns all P ∈ proofs(C, r, p) with height(P) ≤ 1.

For the next step of the structural induction, we now assume that P has a height of height(P) = K + 1. Since K + 1 > 0 for every K ∈ N, the credential c of P cannot be a simple member credential: Assume c = (p, r, c, s) is a simple member credential. Therefore, s = \emptyset. Since s is empty, the height of P would be 0. This violates the
assumption that the height of $P$ is $K + 1$. Accordingly, $c$ must be either a simple containment, intersection containment, or linking containment credential.

The only difference between this next step of the structural induction and the previous one is the height of $P$, which is now assumed to be $K + 1$. However, most of the reasoning of the previous step can be applied to this step:

- The initial constraint check and the final constraint check still succeed, since we have already proven that the extension of role paths correctly reflects the structure of the real role paths of $P$.
- The node bound to $r$ is still selected correctly, as is the credential $c$ inside this node.
- If $s \in \text{sets}$, $P$ will be constructed in line 21 and consequently returned (since the final constraint check succeeds for all valid $P$).

Accordingly, in the following we will assume the points mentioned above to be already proven. In order to show that $P$ with a height of $\text{height}(P) = K + 1$ is returned by $\text{CONSTRUCT-PROOFS}$, we only have to show that the set $s$ of $P$ is added to $\text{sets}$ correctly.

Therefore, we distinguish the three type cases of $c$:

**Simple Containment** If $c$ is a simple containment credential, it must be of the form

$$c = K_A \text{ signs } (r \leftarrow r_1, \text{con})$$

with $r$ being a role in the namespace of $K_A$ and $r_1$ being an arbitrary role. Furthermore, we make the following observations:

- Since $P$ is a valid proof,
  - the constraint $\text{con}$ accepts every role path of $P$,
  - and $P$ can be used to construct the inference
    $$\frac{p \text{ in } r_1}{p \text{ in } r} \frac{c}{\text{sc}}$$

- Since the inference requires $p$ to be assigned to the role $r_1$, the set of subproofs $s$ must contain a proof for this assignment, i.e. $s = \{P_y\}$ with $P_y = (p, r_1, c_1, s_1)$.
- Since $\text{height}(P) = K + 1$, the height of $P_y$ must be $K$.

When evaluating $\text{CONSTRUCT-PROOFS}(r, p, \text{nodes}_C, (\), \emptyset)$, the reasoning on why the initial constraint check succeeds and on why the correct node is selected is equivalent to the simple containment case for $K \leq 1$.

After $c$ has been selected in line 11, the function $\text{HANDLE-SIMPLE-CONTAINMENT}$ is evaluated in line 15. In line 1 of $\text{HANDLE-SIMPLE-CONTAINMENT}$, the function $\text{CONSTRUCT-PROOFS}$ is evaluated to return all proofs that show that $p$ is assigned to $\text{BODY}(c) = r_1$. We know that
• $P_y$ is such a proof since $P_y = (p, r_1, c_1, s_1)$,
• $\text{height}(P_y) = K$, and
• $\text{CONSTRUCT-PROOFS}$ returns all proofs $P_i \in \text{Proofs}(C, r, p)$ with a height of $\text{height}(P_i) = K$.

Therefore $\{P_y\}$ (amongst others) is returned by this evaluation. Let $\text{sets}'$ denote the set of proofs returned by $\text{HANDLE-SIMPLE-CONTAINMENT}$ in line 15. We know that $s = \{P_y\} \in \text{sets}'$. In line 15 of the original evaluation of $\text{CONSTRUCT-PROOFS}$, $\text{sets}'$ is assigned to the variable $\text{sets}$ and therefore $s = \{P_y\} \in \text{sets}$.

Therefore, we can follow that

$$P \in \text{CONSTRUCT-PROOFS}(r, p, \text{nodes}_C, (), \emptyset)$$

**Intersection Containment** If $c$ is an intersection containment credential, it must be of the form

$$c = K_A \text{ signs } \langle r \leftarrow r_1 \cap \ldots \cap r_k, \text{con} \rangle$$

with $r$ being a role in the namespace of $K_A$ and $r_1, \ldots, r_k$ being arbitrary roles. Furthermore, we make the following observations:

- Since $P$ is a valid proof,
  - the constraint $\text{con}$ accepts every role path of $P$,
  - and $P$ can be used to construct the inference

  $$
  \begin{array}{c}
  p \text{ in } r_1 \ldots p \text{ in } r_k \\
  \hline
  p \text{ in } r \\
  \hline
  c
  \end{array}
  \quad i_c
  $$

- Since the inference requires $p$ to be assigned to the roles $r_1$ to $r_k$, the set of subproofs $s$ must contain proofs for these assignments, i.e. $s = \{P_1, \ldots, P_k\}$ with $P_1 = (p, r_1, c_1, s_1)$ to $P_k = (p, r_k, c_k, s_k)$.
- Since $\text{height}(P) = K + 1$, the heights of $P_1, \ldots, P_k$ must be less than or equal to $K$, i.e. $\text{height}(P_1) \leq K, \ldots, \text{height}(P_k) \leq K$.

When evaluating $\text{CONSTRUCT-PROOFS}(r, p, \text{nodes}_C, (), \emptyset)$, the reasoning on why the initial constraint check succeeds and on why the correct node is selected is equivalent to the intersection containment case for $K \leq 1$.

After $c$ has been selected in line 11, $\text{HANDLE-INTERSECTION-CONTAINMENT}$ is evaluated in line 17. In lines 3 to 4 of $\text{HANDLE-INTERSECTION-CONTAINMENT}$ for every role $r_1, \ldots, r_k$ the function $\text{CONSTRUCT-PROOFS}$ is evaluated. We know that

- $P_1, \ldots, P_k$ each have a height of less than or equal to $K$, and
- $\text{CONSTRUCT-PROOFS}$ returns all proofs $P_i \in \text{Proofs}(C, r, p)$ with a height of $\text{height}(P_i) = K$. 

25
Accordingly, every evaluation of $\text{Construct-Proofs}$ correctly returns a set containing \{P_1\}, \ldots, \{P_k\}. At this point, the variable $ip$ contains a set $ps$ for every $P_i \in \{P_1, \ldots, P_k\}$ with $P_i \in ps$.

In lines 5 to 7, the cartesian product over all sets $ps \in ip$ is constructed and every element in this cartesian product is then added to $sets$. Consequently, $sets$ is eventually extended with a set $\{P_1, \ldots, P_k\} = s$. Therefore, the evaluation of $\text{Handle-Intersection-Containment}$ returns a set containing $s = \{P_1, \ldots, P_k\}$.

This returned set is then assigned to the variable $sets$ of $\text{Construct-Proofs}$ in its original evaluation and therefore $\{P_1, \ldots, P_k\} = s \in sets$.

Therefore, we can follow that

$$P \in \text{Construct-Proofs}(r, p, nodesC, (\cdot), \emptyset)$$

**Linking Containment** If $c$ is a linking containment credential, it must be of the form

$$c = K_A \text{ signs } (r \leftarrow K_A.R_1.R_2, \text{con})$$

with $r$ being a role in the namespace of $K_A$, $R_1$ being a role in the namespace of $K_A$, and $R_2$ being an arbitrary role. Furthermore, we make the following observations:

- Since $P$ is a valid proof,
  - the constraint $\text{con}$ accepts every role path of $P$,
  - and $P$ can be used to construct the inference

$$\frac{p \text{ in } K_B.R_2 \text{ in } K_A.R_1}{p \text{ in } r} \quad c \quad lc$$

- Since the inference requires $p$ to be assigned to a role $K_B.R_2$ for a principal $K_B$ who is assigned to the role $K_A.R_1$, the set $s$ contains two proofs: $P_l = (p, K_B.R_2, c_l, s_l)$ and $P_d = (K_B, K_A.R_1, c_d, s_d)$.

- Since $\text{height}(P) = K + 1$, the heights of $P_l$ and $P_d$ must be less than or equal to $K$, i.e. $\text{height}(P_l) \leq K$ and $\text{height}(P_d) \leq K$.

When evaluating $\text{Construct-Proofs}(r, p, nodesC, (\cdot), \emptyset)$, the reasoning on why the initial constraint check succeeds and on why the correct node is selected is equivalent to the linking containment case for $K \leq 1$.

After $c$ has been selected in line 11, $\text{Handle-Linking-Containment}$ is evaluated in line 19. In line 4, the defining principal $K_B$ is eventually selected. For this principal, $\text{Construct-Proofs}$ is evaluated in line 5 to show that $K_B$ is assigned to the role $K_A.R_1$. We know that

- $P_l$ shows this assignment,
- $P_d$ has a height of less than or equal to $K$, and
Construct-Proofs returns all proofs $P_i \in \text{Proofs}(C, r, p)$ with a height of $\text{height}(P_i) = K$.

Accordingly, the variable $dp$ afterwards contains $P_d$. Since $dp$ is not empty, in line 7 Construct-Proofs is evaluated to show that $p$ is assigned to the linked role $K_B.R_2$. We know that

- $P_i$ shows this assignment,
- $P_i$ has a height of less than or equal to $K$, and
- Construct-Proofs returns all proofs $P_i \in \text{Proofs}(C, r, p)$ with a height of $\text{height}(P_i) = K$.

Accordingly, the variable $lp$ afterwards contains $P_l$. At this point, we know $P_d \in dp$ and $P_l \in lp$. In line 8, the cartesian product of $dp$ and $lp$ is constructed and every element of this cartesian product is subsequently added to $sets$. Therefore, $\{P_d, P_l\} = s$ is eventually added to $sets$. Therefore, the evaluation of the function Handle-Intersection-Containment returns a sets containing $s = \{P_d, R_l\}$.

This returned set is then assigned to the variable $sets$ of Construct-Proofs in its original evaluation and therefore $\{P_d, P_l\} = s \in sets$.

Therefore, we can follow that

$$P \in \text{Construct-Proofs}(r, p, nodes_C, (\), \emptyset)$$

We have shown that for every $P \in \text{Proofs}(C, r, p)$ the function call

$$\text{Construct-Proofs}(r, p, nodes_C, (\), \emptyset)$$

returns $P$ for

- $\text{height}(P) \leq 1$, and
- $\text{height}(P) = K + 1$ under the assumption Construct-Proofs returns all $P_i \in \text{Proofs}(C, r, p)$ with $\text{height}(P_i) = K$ (with $K = 1$).

Therefore, we can follow that

$$\text{Proofs}(C, r, p) \subseteq \text{Construct-Proofs}(r, p, nodes_C, (\), \emptyset)$$

Case 2. Let $P \in \text{Construct-Proofs}(r, p, nodes_C, (\), \emptyset)$ be a proof of the form $P = (p, r, c, s)$. We will show $P \in \text{Proofs}(C, r, p)$, i.e. $P$ is a valid proof showing $p$ is assigned to the role $r$.

The following observations help us to use induction over the height of $P$: A proof needs to be derivable using the inference rules in order to be valid. Since the simple member inference rule is the only inference rule without any premises, every valid proof has to be based on proofs using simple member credentials. Furthermore, in every valid proof $P = (p, r, c, s)$ with $\text{height}(P) = 1$, $c$ must be a simple member credentials (otherwise the set $s$ would be non-empty and therefore $\text{height}(P) > 1$).

In combination, both observations allow us to use the following induction over the height of $P$:
• As a base case, we prove $P \in \text{Proofs}(C, r, p)$ for all $P$ with $\text{height}(P) \leq K$, $K = 1$.

• As the induction step, we assume $P \in \text{Proofs}(C, r, p)$ for all $P$ with $\text{height}(P) \leq K$ and $K = 1$ and prove the assumption for all $P$ with $\text{height}(P) = K + 1$.

**Base Case:** First, let $P \in \text{Construct-Proofs}(r, p, \text{nodes}_C, (\cdot), \varnothing)$ denote an arbitrary but fixed proof of the form $P = (p, r, c, s)$ with $\text{height}(P) = 1$ (there are no proofs with a height smaller than this).

When evaluating $\text{Construct-Proofs}$, the initial constraint check succeeds (since $\text{constraints} = \varnothing$) and the appropriate node is selected in line 7 is assigned to the variable $\text{node}$. If no appropriate node would have been found, the evaluation would have returned an empty set (see line 5), which violates the assumption that $P$ is returned by $\text{Construct-Proofs}$.

The set $\text{proofs}$, initialized in line 8 as an empty set, is the container for proofs throughout the evaluation. When $P$ is returned, $P$ has to be added to $\text{proofs}$ at some point of the evaluation. Furthermore, $\text{proofs}$ is only extended in two distinct places (lines 10 and 21) of which only the first place extends $\text{proofs}$ with proofs of height 1. Consequently, $P$ must have been added to $\text{proofs}$ in line 10.

In lines 9 to 10, the node’s set of members is traversed and for every element $(c, p')$ in $\text{node}\.\text{members}$ where $p' = p$ a new proof is added to $\text{proofs}$ with $c$ as the credential and $s = \varnothing$. Since the selected node contains only credentials with the head role $r$ and its set of members contains only simple member credentials, $c$ must be a simple member credential assigning $p$ to the role $r$. $P = (p, r, c, \varnothing)$ is therefore added to the set $\text{proofs}$ in line 10.

At this point we know $P \in \text{proofs}$. Furthermore, $P$ is at least semi-valid, since $c$ is a simple member credential showing $p$ is assigned to $r$ and $s = \varnothing$. Now, the set $\text{proofs}$ is filtered in line 22 before being returned. But since we know $P$ will be returned by $\text{Construct-Proofs}$, this check succeeds.

Consequently, we follow that $P$ must be valid, i.e. $P \in \text{Proofs}(C, r, p)$. The height of $K = 1$ serves as the base case for the induction.

From this point on, we assume all $P \in \text{Construct-Proofs}(r, p, \text{nodes}_C, (\cdot), \varnothing)$ with $\text{height}(P) = K$ are valid, i.e. $P \in \text{Proofs}(C, r, p)$.

**Induction Step:** Let $P \in \text{Construct-Proofs}(r, p, \text{nodes}_C, (\cdot), \varnothing)$ be an arbitrary but fixed proof of the form $P = (p, r, c, s)$ with $\text{height}(P) = K + 1$. We can make the following observations:

• Since the height of the proof $P$ is $K + 1$, all proofs $P_i \in s$ must have a height of $\text{height}(P_i) \leq K$.

• If $c$ is a simple member credential, the height of $P$ is $K$. Since the height of $P$ is $K + 1$, $c$ must be a simple containment, intersection containment, or linking containment credential.

• Since $c$ is not a simple member credential, $P$ must be added to $\text{proofs}$ in line 21 and $c$ must be taken from the node’s set of credentials, i.e. $\text{node}\.\text{credentials}$.
As in the previous case, we have to show the semi-validity of $P$. A proof $P$ is semi-valid if and only if the role assignment of $P$ is deducible with the inference rules by using the credential $c$ in combination with the set $s$. Accordingly, we will make a case distinction on the type of $c$ and show this relationship between $c$ and $s$:

**Simple Containment** Let $c$, selected in line 11, be a simple containment credential of the form

$$K_A \text{ signs } r \leftarrow r_1$$

with $K_A$ being the defining principal of $r$ and $r_1$ being an arbitrary role. In order for $P$ to be semi-valid, $c$ and $s$ have to satisfy the simple containment inference rule:

$$
\begin{array}{c}
p \text{ in } r_1 \\
p \text{ in } c \\
p \text{ in } r \\
\end{array}
\quad sc
$$

Therefore, the set $s$ obviously has to contain a proof $P_1$ showing that $p$ is assigned to the role $r_1$ in order for $P$ to be semi-valid. Since $c$ is a simple containment credential, the check in line 14 succeeds and HANDLE-SIMPLE-CONTAINMENT is evaluated in line 15.

This evaluation simply calls CONSTRUCT-PROOFS with the same parameters as in the initial evaluation but with a different role: the body of the credential $c$, i.e. the role $r_1$. Since $P$ has a height of $K+1$, we know every proof in $s$ has to have a height of $height(P_i) \leq K$. Accordingly, this evaluation of CONSTRUCT-PROOFS returns a non-empty set of proofs showing $p$ is assigned to $r_1$. The result of this evaluation is assigned to the variable $sets$ in line 15 of the initial evaluation of CONSTRUCT-PROOFS.

In line 21, for every set in $sets$ a new proof of the form $P_i = (p, r, c, s_i)$ is added to the set $proofs$. Since $sets$ now contains single-element sets with proofs showing $p$ is assigned to $r_1$, every combination of $s_i \in sets$ and $c$ satisfies semi-validity and therefore $P$ is also semi-valid.

**Intersection Containment** Let $c$, selected in line 11, be an intersection containment credential of the form

$$K_A \text{ signs } r \leftarrow r_1 \cap \ldots \cap r_k$$

with $K_A$ being the defining principal of $r$ and $r_1, \ldots, r_k$ being arbitrary roles. In order for $P$ to be semi-valid, $c$ and $s$ have to satisfy the intersection containment inference rule:

$$
\begin{array}{c}
p \text{ in } r_1 \\
\ldots \\
p \text{ in } r_k \\
p \text{ in } r \\
\end{array}
\quad ic
$$

Therefore, the set $s$ obviously has to contain proofs $P_1, \ldots, P_k$ showing that $p$ is assigned to $r_1, \ldots, r_k$, respectively, in order for $P$ to be semi-valid. Since
c is an intersection containment credential, the check in line 16 succeeds and HANDLE-INTERSECTION-CONTAINMENT is evaluated in line 17.

This evaluation calls CONSTRUCT-PROOFS for every $r_i \in \text{BODY}(c)$, i.e. $r_1, \ldots, r_k$, in lines 3 to 4. Since $P$ has a height of $K+1$, we know every proof in $s$ has to have a height of $\text{height}(P_i) \leq K$. Accordingly, these evaluations of CONSTRUCT-PROOFS return non-empty sets of proofs showing $p$ is assigned to $r_1, \ldots, r_k$, respectively. The resulting sets are assigned to the variable $ip$. In lines 5 to 7, the cartesian product over these sets is calculated and later on - in the initial evaluation of CONSTRUCT-PROOFS - assigned to the variable $sets$.

In line 21 (of the initial evaluation), for every set in $sets$ a new proof of the form $P_i = (p, r, c, s_i)$ is added to the set $proofs$. Since $sets$ now contains sets with proofs showing $p$ is assigned to $r_1, \ldots, r_k$, respectively, every combination of $s_i \in sets$ and $c$ satisfies semi-validity and therefore $P$ is also semi-valid.

**Linking Containment** Let $c$, selected in line 11, be a linking containment credential of the form

$$K_A \text{ signs } r \leftarrow K_B.r_1.r_2$$

with $K_A$ being the defining principal of $r$, $K_B$ being an arbitrary principal defining a role $r_1$, and $r_2$ being an arbitrary role term. In order for $P$ to be semi-valid, $c$ and $s$ have to satisfy the linking containment inference rule:

$$\frac{p_1 \in K_B.r_1 \quad p \in p_1.r_2 \quad c}{p \in r} lc$$

Therefore, the set $s$ obviously has to contain proofs $P_d$ and $P_l$ showing that a principal $p_1$ is assigned to the role $K_B.r_1$ and that $p$ is assigned to the role $p_1.r_2$, respectively, in order for $P$ to be semi-valid. Since $c$ is a linking containment credential, the check in line 18 succeeds and HANDLE-LINKING-CONTAINMENT is evaluated in line 19.

In this evaluation, for every principal $p_1$ who defines a role with the role term $r_2$ the function CONSTRUCT-PROOFS is evaluated with the defining role $K_B.r_1$ as the role parameter. The result is stored in the set $dp$. Then, for every such principal $p_1$ the function CONSTRUCT-PROOFS is evaluated with $p$ as the principal and the linked role $p_1.r_2$ as the role parameter. The result is then stored in the set $lp$.

The returning set of HANDLE-LINKING-CONTAINMENT is then extended with the cartesian product of the sets $dp$ and $lp$. Now, since $P$ has a height of $K+1$, we know every proof in $s$ has to have a height of $\text{height}(P_i) \leq K$. Accordingly, the sets $dp$ and $lp$ contain non-empty sets with proofs showing an arbitrary $p_1$ is assigned to $K_B.r_1$ and $p$ is assigned to $p_1.r_2$, respectively. The cartesian product is then - in the initial evaluation of CONSTRUCT-PROOFS - assigned to the variable $sets$ in line 19.

In line 21 (of the initial evaluation), for every set in $sets$ a new proof of the form $P_i = (p, r, c, s_i)$ is added to the set $proofs$. Since $sets$ now contains sets with proofs
showing an arbitrary \( p_1 \) is assigned to \( K_B r_1 \) and \( p \) is assigned to \( p_1 r_2 \), every combination of \( s_i \in \text{sets} \) and \( c \) satisfies semi-validity and therefore \( P \) is also semi-valid.

We now know, when \( P \) is added to the set \( \text{proofs} \) in line 21 it is (at least) semi-valid. Since \( \text{CONSTRUCT-PROOFS} \) returns only valid proofs, due to the filtering process in line 22, we can follow that \( P \) indeed must be valid, i.e. \( P \in \text{Proofs}(C, r, p) \).

In general, the induction over the height of \( P \) shows that an arbitrary but fixed

\[
P \in \text{CONSTRUCT-PROOFS}(r, p, \text{nodes}_C, (), \emptyset)
\]

is valid, i.e. \( P \in \text{Proofs}(C, r, p) \). Therefore we can follow that

\[
\text{CONSTRUCT-PROOFS}(r, p, \text{nodes}_C, (), \emptyset) \subseteq \text{Proofs}(C, r, p)
\]
Appendices
A. Algorithm

Definition A.1. The function call $\text{New-Proof}(r, p, c, s)$ creates a new proof data structure. Let $\text{proof}$ denote this proof data structure. The function sets the field role of $\text{proof}$ to $r$, the field principal to $p$, the field credential to $c$, and the field subproofs to $s$. The expression $\text{proof}.x$ accesses the field $x$ of the proof data structure $\text{proof}$.

CREATE-NODES($\text{credentials}$)
1 $\text{nodes} = \emptyset$
2 for each credential $c \in \text{credentials}$
3 if $\text{nodes}$ contains no node for $\text{HEAD}(c)$
4 $\text{nodes} = \text{nodes} \cup \{\text{new node for } \text{HEAD}(c)\}$
5 $\text{node} = \text{node for } \text{HEAD}(c)$
6 if $c$ is a simple member credential
7 $\text{node.members} = \text{node.members} \cup \{c, \text{BODY}(c)\}$
8 else
9 $\text{node.credentials} = \text{node.credentials} \cup \{c\}$
10 return $\text{nodes}$

HANDLE-SIMPLE-CONTAINMENT($\text{credential}, \text{principal}, \text{nodes}, \text{rolepath}, \text{constraints}$)
1 return $\text{Construct-Proofs(\text{BODY}(c), \text{principal}, \text{nodes}, \text{rolepath}, \text{constraints})}$

HANDLE-LINKING-CONTAINMENT($\text{credential}, \text{principal}, \text{nodes}, \text{rolepath}, \text{constraints}$)
1 $\text{sets} = \emptyset$
2 $\text{defining-role} = \text{defining role of } \text{BODY}(c)$
3 $\text{linked-role-term} = \text{linked role of } \text{BODY}(c)$
4 for each principal $p$ who defines a role $r$ with the role term $\text{linked-role-term}$
5 $\text{dp} = \text{Construct-Proofs(\text{defining-role}, p, \text{nodes}, ()}, \emptyset)$
6 if $\text{dp}$ is not empty
7 $\text{lp} = \text{Construct-Proofs}(r, \text{principal}, \text{nodes}, \text{rolepath}, \text{constraints})$
8 for each $e \in \text{dp} \times \text{lp}$
9 $\text{sets} = \text{sets} \cup \{e\}$
10 return $\text{sets}$
HANDLE-INTERSECTION-CONTAINMENT(credential, principal, nodes, rolepath, constraints)

1. sets = ∅
2. ip = ∅
3. for each role r_i ∈ Body(c)
   4. ip = ip ∪ CONSTRUCT-PROOFS(r_i, principal, nodes, rolepath, constraints)
5. let a_1, ..., a_n denote all sets in ip, with a_i = a_j ⇔ i = j
6. for each combination e = {e_1, ..., e_n} with e_i ∈ a_i
7. sets = sets ∪ e
8. return sets

EXTEND-ROLE-PATH(rolepath, role)

1. if rolepath = (r_0, ..., r_i)
   2. return (r_0, ..., r_i, role)
3. else
   4. return (role)

FILTER-VALID-PROOFS(proofs)

1. for every proof p ∈ proofs
2. for every constraint con ∈ Con(p)
3. if con does not accept p
   4. proofs = proofs \ {p}
5. return proofs
\textbf{Construct-Proofs}(role, principal, nodes, rolepath, constraints)

1. \( r' = \text{Extend-Role-Path}(rolepath, role) \)
2. \textbf{for} every constraint \( con \in \text{constraints} \)
   \begin{itemize}
   \item if \( con \) cannot reach an accepting state after the input \( r' \)
   \quad \textbf{return} \emptyset
   \item if \( nodes \) contains no node for \( role \)
   \quad \textbf{return} \emptyset
   \end{itemize}
3. \( node = \text{node for role} \)
4. \( proofs = \emptyset \)
5. \textbf{for} each \( \{c, p\} \in \text{node.members} \) with \( p == \text{principal} \)
   \begin{itemize}
   \item \( proofs = proofs \cup \text{NEW-PROOF}(role, principal, c, \emptyset) \)
   \end{itemize}
6. \textbf{for} each credential \( c \in \text{node.credentials} \)
7. \( sets = \emptyset \)
8. \( constraints' = constraints \cup \{\text{Con}(c)\} \)
9. if \( c \) is a \textbf{simple containment} credential
   \begin{itemize}
   \item \( sets = \text{HANDLE-SIMPLE-CONTAINMENT}(c, principal, nodes, r', constraints') \)
   \end{itemize}
10. elseif \( c \) is an \textbf{intersection containment} credential
    \begin{itemize}
    \item \( sets = \text{HANDLE-INTERSECTION-CONTAINMENT}(c, principal, nodes, r', constraints') \)
    \end{itemize}
11. elseif \( c \) is a \textbf{linking containment} credential
    \begin{itemize}
    \item \( sets = \text{HANDLE-LINKING-CONTAINMENT}(c, principal, nodes, r', constraints') \)
    \end{itemize}
12. \textbf{for} each set of subproofs \( s \in sets \)
    \begin{itemize}
    \item \( proofs = proofs \cup \text{NEW-PROOF}(role, principal, c, s) \)
    \end{itemize}
13. \textbf{return} \text{FILTER-VALID-PROOFS}(proofs)

\textbf{SAGE-Prove}(credentials, role, principal)

1. \( nodes = \text{CREATE-NODES}(credentials) \)
2. \( constraints = \emptyset \)
3. \textbf{return} \text{Construct-Proofs}(role, principal, nodes, (), constraints)
B. Cycle-Safe Algorithm

HANDLE-SIMPLE-CONTAINMENT(credential, principal, nodes, rolepath, constraints, history)
1 \textbf{return} Construct-Proofs(Body(c), principal, nodes, rolepath, constraints, history)

HANDLE-LINKING-CONTAINMENT(credential, principal, nodes, rolepath, constraints, history)
1 \texttt{sets} = \emptyset
2 \texttt{defining-role} = \text{defining role of Body}(c)
3 \texttt{linked-role-term} = \text{linked role of Body}(c)
4 \textbf{for} each principal \( p \) who defines a role \( r \) with the role term \texttt{linked-role-term}
5 \hspace{1em} dp = Construct-Proofs(\texttt{defining-role}, p, nodes, (\), \emptyset, history)
6 \hspace{1em} if \ dp \text{ is not empty}
7 \hspace{2em} lp = Construct-Proofs(r, principal, nodes, rolepath, constraints, history)
8 \hspace{2em} \textbf{for} each \( e \in dp \times lp \)
9 \hspace{3em} \texttt{sets} = \texttt{sets} \cup e
10 \textbf{return} \texttt{sets}

HANDLE-INTERSECTION-CONTAINMENT(credential, principal, nodes, rolepath, constraints, history)
1 \texttt{sets} = \emptyset
2 \texttt{ip} = \emptyset
3 \textbf{for} each role \( r_i \in \text{Body}(c) \)
4 \hspace{1em} \texttt{ip} = \texttt{ip} \cup Construct-Proofs(\( r_i \), principal, nodes, rolepath, constraints, history)
5 \hspace{1em} \textbf{let} \( a_1, \ldots, a_n \) denote all sets in \( \texttt{ip} \), with \( a_i = a_j \leftrightarrow i = j \)
6 \hspace{1em} \textbf{for} each combination \( e = \{e_1, \ldots, e_n\} \) with \( e_i \in a_i \)
7 \hspace{2em} \texttt{sets} = \texttt{sets} \cup e
8 \textbf{return} \texttt{sets}

EXTEND-ROLE-PATH(rolepath, role)
1 \textbf{if} rolepath = (\( r_0, \ldots, r_i \))
2 \hspace{1em} \textbf{return} (\( r_0, \ldots, r_i \), role)
3 \textbf{else}
4 \hspace{1em} \textbf{return} (\( \text{role} \))

FILTER-VLID-PROOFS(proofs)
1 \textbf{for} every proof \( p \in \text{proofs} \)
2 \hspace{1em} \textbf{for} every constraint \( \text{con} \in Con(p) \)
3 \hspace{2em} \textbf{if} \ con \text{ does not accept } p
4 \hspace{3em} \text{proofs} = \text{proofs} \setminus \{p\}
5 \textbf{return} \text{proofs}
**Construct-Proofs**\((role, principal, nodes, rolepath, constraints, history)\)

1. if \((role, principal)\) ∈ history
2. return ∅
3. \(h' = history \cup \{(role, principal)\}\)
4. \(r' = \text{Extend-Role-Path}(rolepath, role)\)
5. for every constraint \(con \in constraints\)
6. if \(con\) cannot reach an accepting state after the input \(r'\)
7. return ∅
8. if \(nodes\) contains no node for \(role\)
9. return ∅
10. node = node for \(role\)
11. proofs = ∅
12. for each \(\{c, p\} \in node.members\) with \(p == principal\)
13. proofs = proofs \∪ \text{New-Proof}(role, principal, c, ∅)
14. for each credential \(c \in node.credentials\)
15. constraints' = constraints \∪ \{\text{Con}(c)\}
16. if \(c\) is a **simple containment** credential
17. sets = \text{Handle-Simple-Containment}(c, principal, nodes, \(r'\), constraints', \(h'\))
18. elseif \(c\) is an **intersection containment** credential
19. sets = \text{Handle-Intersection-Containment}(c, principal, nodes, \(r'\), constraints', \(h'\))
20. elseif \(c\) is a **linking containment** credential
21. sets = \text{Handle-Linking-Containment}(c, principal, nodes, \(r'\), constraints', \(h'\))
22. for each set of subproofs \(s \in sets\)
23. proofs = proofs \∪ \text{New-Proof}(role, principal, c, s)
24. return \text{Filter-Valid-Proofs}(proofs)

**Sage-Prove**\((credentials, role, principal)\)
1. nodes = \text{Create-Nodes}(credentials)
2. constraints = ∅
3. return **Construct-Proofs**\((role, principal, nodes, (), constraints, ∅)\)
Bibliography