A Proof-Carrying File System

Deepak Garg and Frank Pfenning

June 6, 2009
CMU-CS-09-123

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper presents the design and implementation of PCFS, a file system that uses formal proofs and capabilities to efficiently enforce access policies expressed in a rich logic. Salient features include backwards compatibility with existing programs and automatic enforcement of access rules that depend on both time and system state. We rigorously prove that enforcement using capabilities is correct, and evaluate the file system's performance.

This work was supported partially by the iCAST project sponsored by the National Science Council, Taiwan, under grant no. NSC97-2745-P-001-001, and partially by the Air Force Research Laboratory under grant no. FA87500720028.

Keywords: Access control, proof carrying authorization, file system

1 Introduction

There is a significant mismatch in the complexity of file access policies prevalent in large organizations like intelligence and military establishments, and the sophistication of mechanisms currently available for their enforcement. Policies often rely on high level concepts like delegation of rights, time-based expiration of credentials, and attributes of individuals and files, whereas the only available mechanism for enforcing these policies in file systems today is access control lists. Translating the intent of complex policy rules to these low level lists, and keeping the latter up-to-date with respect to changing credentials requires substantial and continuous manual effort and is a source of many policy enforcement errors.

These considerations suggest the need for an architecture that represents the high-level intent of policy rules directly, and automatically enforces access control. Proof-carrying authorization (PCA) [7–9] is a promising, open-ended architecture for this purpose; it has previously been applied to web services and distributed access control for physical devices. In PCA, policy rules are represented as logical formulas at a high level of abstraction and enforced automatically with proofs. However, during each access to a resource, a logical proof that establishes relevant access rights for the calling process must be verified. This is a slow process that takes tens or hundreds of milliseconds, making the architecture infeasible for a realistic file system.

This paper presents the design and implementation of a file system that adapts PCA to provide *direct and efficient* enforcement of complex access policies. Like PCA, access in the file system depends on proofs, and hence we call it the Proof-Carrying File System (PCFS). To be precise, however, access requests in PCFS do not actually carry proofs in them as they do in PCA. Instead, proofs are verified by a trusted program in advance of access, and exchanged for capabilities that are used to authorize access. By combining proofs and capabilities in this manner, PCFS retains PCA's high-level policy enforcement, without the bottleneck due to verification of proofs at the point of access.

Briefly, PCFS works as follows. The access policy is represented as a set of logical formulas and distributed to users in the form of digital certificates signed by policy administrators. A user constructs formal proofs which show that the policy entails certain permissions for her. Each proof is checked by a trusted proof verifier which gives the user a signed capability in return. This capability, called a *procap* (for *pro*ven *cap*ability), can be used repeatedly to get authorized access to the file system. A capability can be checked in a few microseconds. As a result, file access in PCFS is very efficient. Another merit of exchanging proofs for capabilities in advance of access is that the implementation factors into two parts that interact via capabilities only: (a) the front end that deals with policies, proofs and generation of capabilities, and (b) the backend that uses capabilities to authorize access and perform I/O. Indeed the PCFS backend is independent of the logic used in the front end, and it can be used with any policy infrastructure that produces compatible capabilities.

Besides the fact that PCFS is the first implementation of a file system that uses logic for rigorous, automatic, and efficient policy enforcement, we believe that our work makes three technically challenging contributions. The first contribution is an expressive logic for writing policies, called BL, which allows a novel combination of user-defined predicates, predicates that capture the state of the system, and rules and credentials that are valid only in stipulated intervals of time. The latter two allow representation of policy rules that depend on file meta data (like extended attributes), as well as rules that expire automatically.

Second, we develop an end-to-end enforcement mechanism for such rich policies. This is non-trivial because constraints about time and system state that occur in logical formulas must also be reflected in proofs, and subsequently in capabilities that are used for enforcement. For this reason, capabilities used in PCFS are conditional on the time of access and the prevailing system state. In addition to the implementation, we prove a theorem which shows that enforcement using capabilities is sound with respect to a PCA-like enforcement where proofs are checked directly at each access.

Third, as opposed to all existing implementations that use PCA or related mechanisms for enforcement of policies, PCFS is compliant with the POSIX file system call interface, and is backward compatible with existing programs. This is made possible due to two design decisions. First, instead of requiring programs to pass capabilities during file system calls, capabilities are put in an indexed store on disk from where they are read by the file system interface (hence existing programs don't have to change). Second, when a new file is created, the user creating the file automatically gets access to the file for a fixed period of time. As a result, programs can freely create and use temporary files, without requiring administrators to create policies to govern them.

The intended deployment for PCFS is in file servers where multiple users log into the same machine and access shared files, which need to be protected through complex rules. Another interesting application of the PCFS architecture could be in situations where the storage is not powerful enough to verify complete proofs, but has enough computational power to check the much simpler capabilities (e.g., in embedded devices). We also expect that this combination of logic and capabilities can be used for access control in other operating system interfaces besides file systems.

Organization. The rest of this paper is organized as follows. In Section 2 we introduce the architecture of PCFS and its various components. Section 3 covers the logic used to represent policies, its features and meta-theoretic properties. Section 4 describes the front end of the file system including our implementation of certificates, automatic proof search, and proof verification that creates procaps. Section 5 discusses the backend of the file system that uses procaps to authorize permissions. Section 6 evaluates PCFS in terms of expressiveness and performance. Section 7 discusses related work, and Section 8 concludes the paper.

2 Overview of PCFS

PCFS is implemented as a local file system for the Linux operating system. It is based on the Fuse kernel module [2]. Technically, PCFS is a *virtual file system* since it uses an underlying file system (ext3 in all experiments reported in this paper) to perform I/O after relevant access checks are made. PCFS is mounted using the command:

```
$> sudo pcfs-main /path/to/src /path/to/mountpoint
```

Here `/path/to/src` is an existing directory in an ext3 system, and `/path/to/mountpoint` is an empty directory. After the execution of this command, any file system call on

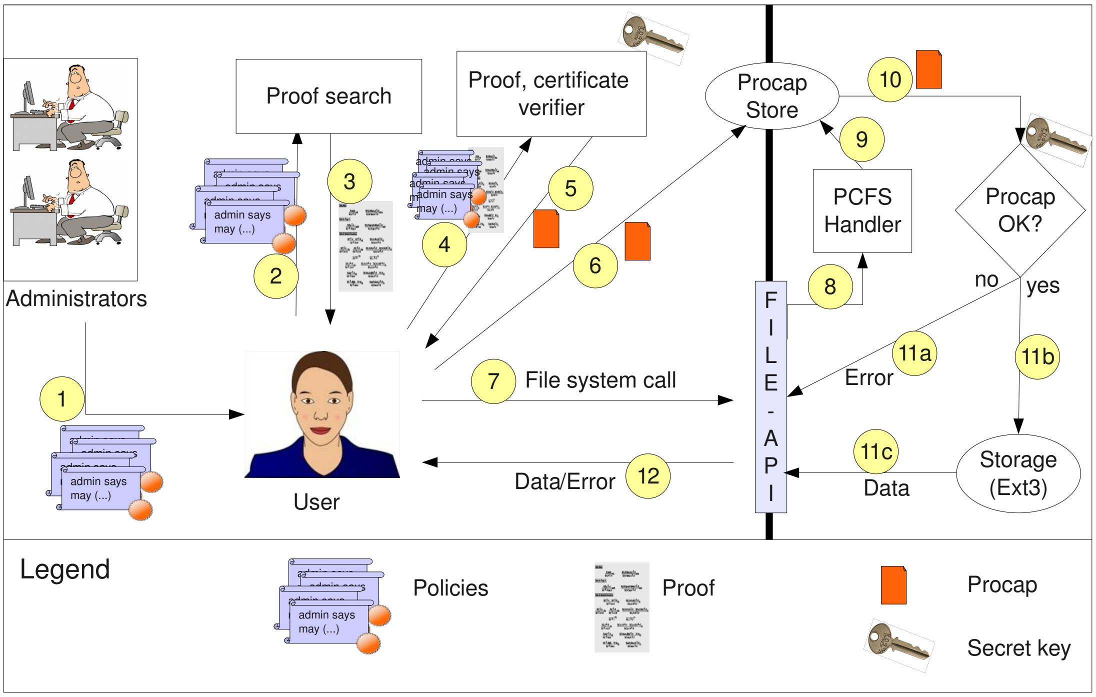

Figure 1: PCFS Workflow

a path like `/path/to/mountpoint/foo/bar` results in a corresponding operation on `/path/to/src/foo/bar`, but is subject to rigorous access checks.

Access checks in PCFS rely on a combination of proof-carrying authorization (PCA) [7–9] and cryptographic capabilities. PCA provides the backbone for enforcement of the access policy through formal logic and digital certificates, while capabilities are used to improve efficiency. We consider this combination a novel contribution of this work, although capabilities have been used in other settings in the past to offset the cost of access checks [6, 20, 28, 31, 32]. Capabilities in PCFS are called *pro*ven *cap*abilities, or *procaps*, since they are obtained by verifying formal logical proofs. A detailed comparison of PCFS to existing access control systems based on PCA is provided in Section 7.

Figure 1 shows the PCFS workflow. Numbers are used to label steps in order of occurrence. Steps 1–6 create and store procaps which show that a user is allowed certain permissions in the file system. These steps are performed in advance of file access, and happen infrequently (usually when a user accesses a file for the first time). Once procaps are stored, they can be used repeatedly to perform file operations (steps 7–12). The solid black vertical line in the diagram separates parts that happen in user space, i.e., before and after a file system call (left side of the line) from those that happen during a file system call (right side of the line).

In the following we describe the steps of Figure 1 in some detail. Briefly, policy enforcement in PCFS follows the path:

$$\text{Policy} \rightarrow \text{Proof} \rightarrow \text{Procap} \rightarrow \text{File access}$$

Policy creation (Step 1). We define a *policy* as a set of rules and facts which determines access rights. An access right is a triple $\langle k, f, \eta \rangle$, which means that user k (Alice, Bob, etc) has permission η (read, write, etc) on file or directory f. We allow different rules and facts in a policy to be created by different individuals called *administrators* (this is necessary to faithfully represent separation of duty in many organizations). We require that each administrator write her portion of facts and rules as *logical formulas* in a text file and digitally sign the file with her private key. This signed file is called a *certificate.* In a concrete sense, therefore, a policy is a collection of certificates signed by different administrators. Abstractly, a policy is a collection of logical formulas that are contained in the certificates. We often denote this collection of logical formulas with the symbol Γ. Representing a policy as logical formulas as opposed to, say, natural language has the advantage that its meaning becomes unambiguous through the logic's inference rules. Logical representation is also amenable to automatic enforcement. PCFS provides its own logic, BL, for writing logical formulas. BL is more expressive than prior logics designed for similar purposes, and its syntax and proof system are described in Section 3.

PCFS provides a command line tool, `pcfs-cert`, to help administrators check formulas for adherence to logical syntax, to digitally sign them, and to convert them to a custom certificate format. (We could have used a standard certificate format like X.509 [23], but found it easier to create our own format.) Policy rules and facts in practice generally follow specific templates, and we expect that our command line tool can be replaced by GUIs. We do not assume a centralized store for certificates. Instead they are distributed to users to whom they grant permissions. Typically, some certificates are created once and used for many months or years, whereas others are created as events happen in the system. As a result of the latter, the policy itself is not static, but changes over time.

Proof generation (Steps 2–3). Once certificates have been created by administrators and given to users, the latter use them to show that they are allowed certain permissions in the file system. The basic tenet of PCFS (adapted from PCA) is that a user k is allowed permission η on resource f at time u, if and only if the user can provide a *formal logical proof* M which shows that the policies in effect (Γ) entail a fixed formula $\texttt{auth}(k, f, \eta, u)$, or in formal notation, $M :: \Gamma \vdash \texttt{auth}(k, f, \eta, u)$. The formula $\texttt{auth}(k, f, \eta, u)$ is defined in Section 3.

To help users construct the proof M, PCFS provides an automatic theorem prover, through the command line tool `pcfs-search`. This tool is based in logic programming [27] (see Section 4 for a brief description of our approach). Figure 1 shows the user giving the policy (certificates) to the proof search tool in step 2, and the proof search tool returning a proof in step 3. A typical proof construction in PCFS takes several hundred milliseconds. A salient point is that the proof search tool is *not* a trusted component of PCFS and it is perfectly alright for a user to create her own proof search tool or even use a heuristic-based method or decision procedure to construct proofs in specific cases.

Proof verification (Steps 4–5). Once the user has constructed a proof M, this proof, together with the certificates used to construct it, is given to a proof verifier, invoked using another command line program `pcfs-verify` (Step 4 in Figure 1). The code of the verifier is simpler than that of the prover and it must be trusted. The verifier checks that the logical structure of the proof M is correct, and that all certificates used in the proof are genuine, i.e., their digital signatures check correctly. If both these hold, then the verifier gives back to the user a procap, which is a capability that mentions the right $\langle k, f, \eta \rangle$ that the proof grants (Step 5). The procap also contains some conditions on which the proof depends and is signed using a shared symmetric key that is known only to the verifier and the file system interface (see Section 4 for details). A typical proof verification including creation of a procap takes several tens or a few hundred milliseconds, depending on the size of the proof.

Procap injection (Step 6). After receiving a procap, the user calls another command line tool which puts the procap in a central store marked "Procap Store" in Figure 1. This store is in a designated part of the PCFS file system, and is accessible to both users as well as the system interface. The system interface looks up this store to find relevant procaps when file system calls are made. The organization of the store is described in Section 5.

File system call (Step 7). A call to the PCFS file system is made through the usual POSIX file system API during the execution of a user program or through a shell command like `ls, cp, rm,` etc. The PCFS backend respects the standard POSIX interface, so user programs and shell commands don't need to change to work on it. However, before a program is started or a shell command is executed, the user must ensure that procaps granting the executing process all needed permissions have been created and injected using Steps 2–6. Alternatively, the program may be augmented to possibly create, and certainly inject, procaps on the fly.

Procap look up and checking (Steps 8–10). When a system call is made on a PCFS file system, it is redirected by the Linux kernel to a process server which we have written (Step 8 in Figure 1). Depending on the specific operation requested, this server looks up one or more procaps in the procap store (Steps 9 and 10). The exact procaps needed for each operation vary, and are listed in Section 5. If all relevant procaps are found, they are checked. Checking a typical procap takes only 10–100μs (*cf.* the time taken to check a proof, which is of the order of tens or hundreds of milliseconds). Details of procap checking are presented in Section 4.

Error (Steps 11a, 12). If any procap needed for performing the requested file operation is missing, or fails to check, an error code is returned to the user program.

File operation (Steps 11b, 11c, 12). If all relevant procaps needed to perform the requested file operation are found, and successfully check, then the underlying ext3 file system is used to perform the requested file operation (Step 11b). The result of the operation is returned to the user (Steps 11c and 12).

2.1 Implementation

The PCFS implementation can be roughly divided into two parts: (a) the **front end**, which comprises the command line tools for creating certificates, constructing proofs, checking proofs to create procaps, and injecting procaps into the central store (Steps 1–6 in Figure 1), and (b) the **backend** which handles the calls from the Fuse kernel module, looks up procaps in the store, checks them, and then makes calls on the underlying file system to perform disk operations (Steps 8–11c in Figure 1). The two parts interact via procaps which carry information from logical proofs into the file system's interface. The front end (with the exception of the procap injection tool) is based in logic, and the technical challenge there has been the development of a well-founded logic (BL) that is not only expressive, but that can also be efficiently implemented. Our implementation of the front end tools is written in Standard ML, and comprises nearly 7,000 lines of code. OpenSSL is used for all cryptographic operations. Because the front end tools are used less frequently than the backend, their efficiency is also less of a concern. The backend is the bottleneck for performance and needs to be extremely efficient. It is implemented in C++ using approximately 10,000 lines of code.

3 BL: The Authorization Logic

PCFS provides a logic for expressing policies, which we call BL, and outline in this section.[1] A detailed description of the logic's proof system and meta theory is deferred to Appendix A. BL is an extension of first-order intuitionistic logic with two modalities that have been studied in prior work [5, 16, 24]: k says s, which means that principal k states or believes formula s, and s @ $[u_1, u_2]$ which means that s holds from time u_1 to time u_2. The former is used to distinguish in the logic parts of the policy made by different individuals whereas the latter is needed to accurately represent time-dependent rules. The logical interpretation of k says s in BL is different from that in any existing work. This new interpretation is designed to facilitate fast proof search. In addition to these modalities, BL supports constraints, which are relations between terms decided using external decision procedures not formalized in the logic (e.g., the usual order $\leq$ on integers). BL also supports predicates that capture the state of the file system. Formulas in BL are denoted using the letters s and r. The syntax of BL is summarized below.

$$\begin{array}{llll}
\text{Sorts} & \sigma & ::= & \mathsf{principal} \mid \mathsf{time} \mid \mathsf{file} \mid \mathsf{perm} \mid \ldots \\
\text{Terms} & t & ::= & a \mid v \mid h(t_1, \ldots, t_n) \\
\text{I-Predicates} & I & & \text{(Interpreted Predicates)} \\
\text{U-Predicates} & P & & \text{(Uninterpreted Predicates)} \\
\text{I-Atoms} & i & ::= & I(t_1, \ldots, t_n) \\
\text{U-Atoms} & p, q & ::= & P(t_1, \ldots, t_n) \\
\text{Constraints} & c & ::= & u_1 \leq u_2 \mid k_1 \succeq k_2 \mid \ \ldots \\
\text{Formulas} & r, s & ::= & p \mid i \mid c \mid r \wedge s \mid r \vee s \mid r \supset s \mid \top \mid \bot \mid \forall x{:}\sigma.s \mid \exists x{:}\sigma.s \mid \\
 & & & k \ \mathsf{says}\ s \mid s \ @\ [u_1, u_2]
\end{array}$$

[1] BL stands for "Binder Logic", as a tribute to the trust management framework Binder [14] from which the logic draws inspiration.

As in first-order logic, subjects of predicates are called *terms.* They represent principals, files, time points, etc. Abstractly, terms can be either ground constants a, bound variables v, or applications of uninterpreted function symbols h to ground terms. Terms are classified into sorts σ (sometimes called types). We stipulate at least four sorts: $\mathsf{principal}$, whose elements are denoted by the letter k, time whose elements are denoted by the letter u, file whose elements are denoted by the letter f, and perm (for permission) whose elements are denoted by the letter η. Elements of time are called time points, and it is assumed that ground time points are integers. In the external syntax of the logic, we allow clock times written to second level accuracy as yyyy:mm:dd:hh:mm:ss, but internally they are represented as integers that measure seconds elapsed from a fixed clock time.

The symbol Σ denotes a partial map $v_1{:}\sigma_1, \ldots, v_n{:}\sigma_n$ from term variables to sorts. The judgment $\Sigma \vdash t : \sigma$ means that under the assignment of sorts Σ, term t is well formed with sort σ. (We assume that the sorts of all function symbols and constants are specified separately, but elide the details.)

Predicates in BL are divided into two categories: uninterpreted predicates, denoted P, which are defined using logical rules, and interpreted predicates, denoted I, which capture properties of the environment. By environment we mean the state of the file system, including, but not limited to, meta data contained in files. The environment is reflected in the logic as a set E of interpreted predicates that hold in it. We write $E \models i$ to mean that in the environment E, the interpreted atomic formula i holds (i.e., $i \in E$). In practice, we require a procedure to decide whether each interpreted predicate I holds for some terms in the prevailing state of the file system or not. We assume that the state is volatile, i.e., it may change unpredictably. We believe that the inclusion and enforcement of such interpreted predicates is novel, at least in the context of access control.

Finally, we assume a syntactic class of constraints, denoted c. Like interpreted predicates, constraints are also relations between terms whose satisfaction is determined by decision procedures external to the logic. However, unlike interpreted predicates, constraints are independent of the state of the system. We stipulate at least two types of constraints: $u_1 \leq u_2$ capturing the usual total order on time points, and a pre-order $k_1 \succeq k_2$, read *principal* k_1 *is stronger than principal* k_2. If $k_1 \succeq k_2$, then BL's inference rules force $(k_1 \mathsf{~says~} s) \supset (k_2 \mathsf{~says~} s)$ for every formula s. We also assume that there is a strongest principal ℓ, i.e., $\models \ell \succeq k$ for every k. In particular $(\ell \mathsf{~says~} s) \supset (k \mathsf{~says~} s)$ for every k and s. For this reason ℓ is called the "local authority", a principal whom everyone believes. (The term local authority is borrowed from the language SecPAL [3, 10].) A set of constraints is written Ψ. The decision procedure for checking constraints is reflected in the logic as the judgment $\Psi \models c$, which means that if all constraints in Ψ hold, then so does c.

3.1 Proof System

Next, we present a proof system for BL in the natural deduction style of Gentzen [19]. Our approach is based on the judgmental method [12, 29], where a syntactic category of judgments (distinct from formulas) is the subject of proofs and deductions. Using the judgmental method makes the meta-theory of the logic much easier. Our technical presentation closely follows prior work by DeYoung et al. done in the context of a related logic [16]. As in that work, we introduce two judgments: $s \circ [u_1, u_2]$ meaning that formula s is provably true in the interval $[u_1, u_2]$, and k claims $s \circ [u_1, u_2]$ meaning that principal k states that s holds from u_1 to u_2. The symbol $\circ$ is read "during". The judgment $s \circ [u_1, u_2]$ is internalized in the logic as the formula $s @ [u_1, u_2]$, whereas k claims $s \circ [u_1, u_2]$ is internalized as $(k \ \mathsf{says}\ s) @ [u_1, u_2]$.

Judgments	J	$::=$	$s \circ [u_1, u_2] \mid k \ \mathsf{claims}\ s \circ [u_1, u_2]$	
Sort Map	Σ	$::=$	$v_1{:}\sigma_1 \dots v_n{:}\sigma_n$	
Hypothetical Constraints	Ψ	$::=$	$c_1 \dots c_n$	
Abstract Environment	E			
Views	α	$::=$	k, u_b, u_e	
Hypotheses	Γ	$::=$	$x_1 : J_1 \dots x_n : J_n$	$(n \geq 0)$
Hypothetical Judgments			$\Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$	

Hypothetical judgments (which are established through proofs) have the form $\Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$. Γ is the set of assumed judgments (hypotheses or policy), and Σ, Ψ, and E have meanings described earlier. $x_1, \dots, x_n$ are distinct names that refer to the assumptions in Γ. A novel feature here is the triple $\alpha = k, u_b, u_e$ on the entailment arrow, which we call the *view* of the sequent. The view represents the principal and interval of time relative to which reasoning is being performed. It affects provability in the following manner: while reasoning in view k, u_b, u_e, an assumption of the form k' claims $s \circ [u_1', u_2']$ entails $s \circ [u_1', u_2']$ if $k' \succeq k$, $u_1' \leq u_b$, and $u_e \leq u_2'$. This entailment does not hold in general. Views are explained in greater detail in Appendix A.

A proof is represented compactly as *proof term*, denoted M. We write $M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$ to mean that M is a proof term that represents a proof of the hypothetical judgment that follows it. For each deduction rule in our proof system, there is a unique constructor for proof terms. Consequently, an entire proof can be reconstructed from its proof term and the hypotheses.

Figure 2 shows selected rules of the proof system. The remaining rules are shown in Appendix A. As usual, we have introduction and elimination rules for each connective (marked I and E respectively). For a syntactic entity R, $R[t/x]$ denotes the capture avoiding substitution of term t for variable x in R. The rule (hyp) states that the assumption $s \circ [u_1', u_2']$ entails $s \circ [u_1, u_2]$ if $u_1' \leq u_1$ and $u_2 \leq u_2'$, i.e., the interval $[u_1, u_2]$ is a subset of the interval $[u_1', u_2']$. This makes intuitive sense: if a formula s holds throughout an interval, it must hold on every subinterval as well. The proof term corresponding to this (trivial) derivation is x, where x is also the name for the assumption $s \circ [u_1', u_2']$. The rule (claims) is similar, except that it allows us to conclude $s \circ [u_1, u_2]$ from the assumption k' claims $s \circ [u_1', u_2']$. In this case, it must also be shown, among other things, that k' is stronger than the principal k in the view (premise $\models k' \succeq k$).

(saysI) is the only rule which changes the view. The notation $\Gamma|$ in this rule denotes the subset of Γ that contains exactly the claims of principals, i.e., the set $\{(x : k' \ \mathsf{claims}\ s' \circ [u_1', u_2']) \in \Gamma\}$. The rule means that $(k \ \mathsf{says}\ s) \circ [u_1, u_2]$ holds in any view α if $s \circ [u_1, u_2]$ holds in the view k, u_1, u_2 using only claims of principals. Assumptions of the form $s' \circ [u_1', u_2']$ are eliminated from Γ in the premise because they may have been added in the view α (using other rules not shown here), but may not hold in the view k, u_1, u_2.

$$\boxed{\Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]}$$

$$\frac{\Psi \models u_1' \leq u_1 \qquad \Psi \models u_2 \leq u_2'}{x :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]}\text{hyp}$$

$$\frac{\begin{array}{c}\alpha = k, u_b, u_e \\ \Psi \models u_1' \leq u_1 \qquad \Psi \models u_2 \leq u_2' \qquad \Psi \models u_1' \leq u_b \qquad \Psi \models u_e \leq u_2' \qquad \Psi \models k' \succeq k\end{array}}{x :: \Sigma; \Psi; E; \Gamma, x : k' \textsf{ claims } s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]}\text{claims}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma| \xrightarrow{k, u_1, u_2} s \circ [u_1, u_2]}{(\texttt{pf\_saysI } M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (k \textsf{ says } s) \circ [u_1, u_2]}\textsf{saysI}$$

$$\frac{\begin{array}{c}M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \supset s_2 \circ [u_1, u_2] \\ M_2 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1', u_2'] \qquad \Psi \models u_1 \leq u_1' \leq u_1'' \qquad \Psi \models u_2'' \leq u_2' \leq u_2\end{array}}{(\texttt{pf\_impE } M_1\ M_2\ u_1'\ u_2') :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1'', u_2'']}{\supset}\text{E}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} \forall v{:}\sigma.s \circ [u_1, u_2] \qquad \Sigma \vdash t : \sigma}{(\texttt{pf\_forallE } M\ t) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s[t/v] \circ [u_1, u_2]}\forall\text{E}$$

$$\frac{E \models i}{(\texttt{pf\_sinjI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} i \circ [u_1, u_2]}\text{interI} \qquad \frac{\Psi \models c}{(\texttt{pf\_cinjI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} c \circ [u_1, u_2]}\text{consI}$$

Figure 2: BL: Natural Deduction (Selected rules)

($\supset$E) is a variant of the common rule of modus ponens. It means that if $s_1 \supset s_2$ holds during an interval $[u_1, u_2]$, and s_1 holds during a *subinterval* $[u_1', u_2']$, then s_2 must hold during any interval $[u_1'', u_2'']$, which is contained in both. ($\forall$E) states that if $\forall x{:}\sigma.s$ holds during some interval $[u_1, u_2]$, then $s[t/x]$ holds during the same interval for any term t.

The rule (interI) is used to establish interpreted predicates. It states that an interpreted atomic formula i is provable if $E \models i$. The rule (consI) is similar but it is used to establish constraints.

Meta-theory. A meta-theorem is a theorem about the proof system in general. Meta-theorems not only increase confidence in the foundations of the logic, but also help in constructing automatic proof search tools. We state below two important meta-theorems about BL's proof system: substitution and subsumption. Structural theorems such as weakening for the hypotheses also hold, but we do not state them explicitly. $M[M'/x]$ denotes the capture-avoiding substitution of proof term M' for the name x in the proof term M.

Theorem 3.1 (Substitution)**.** *Suppose the following hold:*

1. $M' :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$
2. $M :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \xrightarrow{\alpha} r \circ [u_1', u_2']$

Then, $(M[M'/x]) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} r \circ [u'_1, u'_2]$

Proof. See Appendix A. □

Theorem 3.2 (Subsumption)**.** *Suppose the following hold:*

1. $M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$
2. $\Psi \models u_1 \leq u_n$ *and* $\Psi \models u_m \leq u_2$

Then, $M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_n, u_m]$

Proof. See Appendix A. □

3.2 Connection to Enforcement

Representation of files and principals. The logic BL does not mandate how files and users are concretely represented. However, from the perspective of an implementation, making this choice is important. In PCFS, files and directories are represented by their full pathnames, relative to the path where PCFS is mounted. Thus, if PCFS is mounted at `/path/to/mountpoint`, then the file `/foo/bar` in any formula refers to the file `/path/to/mountpoint/foo/bar` in the file system. Principals are represented in one of two ways: either as symbolic constants, or by their Linux user ids. The former representation is used for principals that do not correspond to any real users (e.g., organizational roles), while the latter is used for principals that do (e.g., users that run programs and access files). Permissions are given on a per-file (or per-directory) basis to real users.

Representation of policy in BL. If an administrator k creates a rule represented by formula s, and puts it in a certificate that is valid from time u_1 to time u_2, then this rule is reflected in BL as the assumption $k \;\mathsf{claims}\; s \circ [u_1, u_2]$. In addition, we require that each rule be accompanied by a unique name (a string), which is written in the certificate with the rule. This name is used to refer to the assumption in proofs. The whole policy has the general form $\Gamma = \{x_i : k_i \;\mathsf{claims}\; s_i \circ [u_i, u'_i] \mid 1 \leq i \leq n\}$, where k_i's are administrators, and x_i's are unique names for the rules of the policy.

What should be proved? We assume the existence of one distinguished administrator, symbolically denoted admin, who has the ultimate authority on access. In order to get permission η on file f at time u, user k must prove that the policy in effect entails the defined judgment $\mathtt{auth}(k, f, \eta, u)$, where:

$$\mathtt{auth}(k, f, \eta, u) \triangleq (\mathsf{admin}\;\mathsf{says}\;\mathtt{may}(k, f, \eta)) \circ [u, u]$$

$\mathtt{may}$ is a fixed uninterpreted predicate taking three arguments, and u is the time of access ($[u, u]$ is a singleton set containing exactly the time point u).

When we start constructing a proof in BL at the top level, the exact view α does not matter. Further the set Ψ is empty, and Σ is a fixed map provided externally. To get access it must be shown that: $\Sigma; \cdot; E; \Gamma \xrightarrow{\alpha} \mathtt{auth}(k, f, \eta, u)$, where α is a view made of fresh constants, Γ is the policy, u is the time of access, and E is the environment at time u.

Usually, admin delegates part of its authority to other administrators through rules. Also, in most policies, admin may have authority over the predicate may but not others. For this reason, it is advisable to keep admin distinct from ℓ, the strongest principal whom everyone believes on every predicate.

Interpreted Predicates. BL natively supports two interpreted predicates, although support for other predicates can be added easily. These two predicates are: **owner**(f, k), which means that file f has owner k, and **has_xattr**(f, a, v), which means that file f has value v for the extended attribute user.#pcfs.a. Extended attributes beginning with the prefix user.#pcfs. are specially protected by PCFS (a special permission called "govern" is needed to change them). These attributes can be used to label files in a secure manner, as we illustrate in the following example. Interpreted predicates are written in **boldface** to distinguish them from others.

Example 1. We present a fragment of a case study that uses BL to model policies for control and dissemination of classified files in the U.S. Consider a hypothetical intelligence agency where each file and each user is assumed to have a classification level, which is an element of the ordered set $\mathsf{confidential} < \mathsf{secret} < \mathsf{topsecret}$. The classification level of a file is assumed to be written in an extended attribute user.#pcfs.level on the file. We also assume one distinguished administrator (in addition to admin) called hr who is responsible for deciding attributes of users (e.g., giving them classification levels and employment certifications).

In order that principal k may read file f, three conditions must be satisfied: (a) k should be an employee of the intelligence organization (predicate $\mathtt{employee}(k)$), (b) k should have a classification level above the file (predicate $\mathtt{hasLevelForFile}(k, f)$), and (c) k should get permission from the owner of the file. Let us assume that this rule came in effect in 2000, and will remain in effect till 2010. The following rule (created by admin) captures this intent. For readability, we omit all sort annotations from quantifiers.

$$\begin{array}{ll} & \mathsf{admin\ claims}\ \forall k, k', f. \\ & \quad (((\mathsf{hr\ says}\ \mathtt{employee}(k)) \wedge \\ & \qquad \mathtt{hasLevelForFile}(k, f) \wedge \\ (1) & \qquad \mathbf{owner}(f, k') \wedge \\ & \qquad (k'\ \mathsf{says}\ \mathtt{may}(k, f, \mathsf{read}))) \supset \mathtt{may}(k, f, \mathsf{read})) \\ & \circ\, [2000, 2010] \end{array}$$

The predicate $\mathtt{hasLevelForFile}(k, f)$ may further be defined by admin in terms of classification levels of k and f.

$$\begin{array}{ll} & \mathsf{admin\ claims}\ \forall k, f, l, l'. \\ & \quad ((\mathbf{has\_xattr}(f, \mathtt{level}, l) \wedge \\ (2) & \qquad (\mathsf{hr\ says}\ \mathtt{levelPrin}(k, l')) \wedge \\ & \qquad \mathtt{below}(l, l')) \supset \mathtt{hasLevelForFile}(k, f)) \\ & \circ\, [2000, 2010] \end{array}$$

It is instructive to observe the use of the interpreted predicates **owner** and **has_xattr** in these rules. The predicate $\mathtt{below}(l, l')$ captures the order $l < l'$ between classification levels. We assume that all principals believe this order. Hence it is stated by the strongest principal ℓ.

(3) ℓ claims $\texttt{below}$(confidential, secret) $\circ$ [2000, 2010]

(4) ℓ claims $\texttt{below}$(secret, topsecret) $\circ$ [2000, 2010]

(5) ℓ claims $\texttt{below}$(confidential, topsecret) $\circ$ [2000, 2010]

As an illustration of the use of this policy, let us assume that file /secret.txt is owned by Alice (user id 1003) and classified at the level secret. Thus the following must hold in the prevailing file system state E:

(A) $E \models$ **owner**(/secret.txt, uid 1003)

(B) $E \models$ **has_xattr**(/secret.txt, $\texttt{level}$, secret)

Suppose further that Bob (user id 1500) is an employee cleared at level topsecret from 2007 to 2009, and that Alice wants to let Bob read file /secret.txt from 2008 to 2009. This information may be captured by the following formulas (signed by the respective principals).

(6) hr claims $\texttt{employee}$(uid 1500) $\circ$ [2007, 2009]

(7) hr claims $\texttt{levelPrin}$(uid 1500, topsecret) $\circ$ [2007, 2009]

(8) (uid 1003) claims $\texttt{may}$(uid 1500, secret.txt, read)
$\circ$ [2008, 2009]

Let Γ denote the set of policy rules (1)–(8) (with corresponding names $\texttt{p1}$–$\texttt{p8}$), and let Σ be a map that defines the constants used in the policy. Then using the rules of Figure 2 we can show that there is a proof term M such that $M :: \Sigma; \cdot; E; \Gamma \xrightarrow{\alpha}$ (admin says $\texttt{may}$(uid 1500, /secret.txt, read)) $\circ$ [2008, 2009], if E satisfies the conditions (A) and (B). From Theorem 3.2 it follows that $M :: \Sigma; \cdot; E; \Gamma \xrightarrow{\alpha}$ $\texttt{auth}$(uid 1500, /secret.txt, read, u) whenever $u \in [2008, 2009]$, and hence Bob should be able to read /secret.txt from 2008 to 2009. This is what we may intuitively expect because the intersection of the validities of all certificates issued here is exactly [2008, 2009].

4 PCFS Front End: Proof Search and Verification

Having discussed the syntax and proof system of BL, we now turn to its implementation in proof search and proof verification tools. We start by describing the proof search tool briefly, and then turn to the proof verification tool and the structure of procaps.

4.1 Automatic Proof Search

Even though users are free to construct proofs of access by any means they like, PCFS provides a command line tool called `pcfs-search` for performing this task automatically. As discussed in Section 3, the objective is to prove a judgment of the form $\Sigma; \cdot; E; \Gamma \xrightarrow{\alpha}$ (admin says $\texttt{may}(k, f, \eta)$) $\circ$ $[u, u]$, where u is the expected time of access, and E is the expected environment at time u. Of course, in almost all cases, it is unreasonable to expect that the time of access can be predicted in advance to the precision of seconds (which is the precision at which enforcement of time works in PCFS), so instead of an exact time u, the user provides a range of time $[u_1, u_2]$ during which she desires access. Similarly, since the environment E at time u may also be difficult to predict, the environment at the time of proof construction is used as a proxy. The prover can also be run in interactive mode, where it asks for user input about the expected environment if it fails to construct a proof in the prevailing one.

The user must also provide the parameters k, f, η and the policy Γ (in the form of certificates obtained from administrators). The output of the tool is the proof term M such that $M :: \Sigma; \cdot; E; \Gamma \xrightarrow{\alpha} (\mathsf{admin\ says\ may}(k, f, \eta)) \circ [u_1, u_2]$. By Theorem 3.2 it follows that $M :: \Sigma; \cdot; E; \Gamma \xrightarrow{\alpha} \mathtt{auth}(k, f, \eta, u)$ for every $u \in [u_1, u_2]$, so this proof term M can be used for access at any time point in the interval $[u_1, u_2]$.

Proof search in BL is in general an undecidable problem because BL extends first-order intuitionistic logic, which is itself undecidable. However, as past work on languages and logics for authorization shows [10, 11, 14, 26, 30], most access policies in practice fit into a restricted fragment of logic on which logic programming techniques can be used for proof construction. Although logic programming methods work fast, extending them from fragments of first-order logic (where they are well understood) to BL's additional constructs – k says s, s @ $[u_1, u_2]$, constraints, and interpreted predicates – is a challenging task. The @ modality is particularly difficult to handle since it interacts with all other connectives of BL in non-trivial ways. We omit a description of the proof search method, but refer the reader to prior work for details [17].

4.2 Proof Verification and Procaps

The proof verifier checks proofs that a user constructs and issues procaps in return. Since these procaps can be directly used for access, the proof verifier is a trusted piece of code. Briefly, the proof verifier is invoked with a command line tool `pcfs-verify`. It is given as input the policy Γ (in the form of signed certificates), the parameters k, f, η, and a proof term M. The verifier first checks that the policy is correct, i.e., all its certificates have authentic digital signatures. For this, the verifier must have access to some public key infrastructure (PKI) that maps public keys to principals that own them. We use a simple PKI, with a single certifying authority (CA) that certifies all keys. The public key of the CA is stored in a specially protected file in PCFS itself (see Section 5).

Second, the verifier checks the logical structure of the proof term, i.e., it makes sure that it is the case that $M :: \Sigma; \cdot; E; \Gamma \xrightarrow{\alpha} \mathtt{auth}(k, f, \eta, u)$. Checking a logical proof is mostly standard; it works on the observation that the proof term (together with the policy) is enough to reconstruct the structure of the entire proof. Once this step succeeds, the verifier outputs for the user a signed capability, which contains the tuple $\langle k, f, \eta \rangle$. There are three subtleties here.

1. How does the verifier get access to the secret key needed to sign the procap? (Or, what prevents users from accessing the key and forging procaps?)

2. What file system state E is the proof checked in? This is relevant because it should never be the case that a proof successfully checks in some state E but the resulting procap is used in a state where the proof verification would have failed.

3. How does the procap reflect the time interval over which the proof is valid?

To address problem (1), we use a simple method. The secret key is stored in a specially marked file in the PCFS file system. The file system interface ensures that *only a specific user id* (called pcfssystem) has read access to this file. The verification tool `pcfs-verify`'s disk file is owned by this user, and executes with a set-uid bit. As a result, when a user invokes this program, it runs with pcfssystem's user id, and hence gets access to this key.

Problem (2) is addressed by *never checking interpreted predicates* during proof verification. Instead, when the verifier encounters the proof term `pf_sinjI`, which corresponds to an application of the rule (interI) from Figure 2, the verifier writes the interpreted predicate i to be checked in the output procap. This predicate must then be checked by the file system backend when the procap is used. As a result, any interpreted predicates on which the validity of the proof depends are transferred unchanged to the procap, and are checked in the state prevalent at the time of access (see Appendix B for details).

To address problem (3), we use a *special symbolic constant* ctime, which has sort time, and is supposed to represent the actual time at which access is requested. The verifier tries to check that $M :: \Sigma, \mathsf{ctime{:}time}; \cdot; \cdot; \Gamma \xrightarrow{\alpha} \mathtt{auth}(k, f, \eta, \mathsf{ctime})$. Observe that the time of access u is replaced by this symbolic constant. During the verification, many judgments of the form $\Psi \models u_1 \leq u_2$ are encountered (e.g., in the rules (hyp), (claims), ($\supset$E), and (consI)) where either Ψ, u_1, or u_2 contains ctime. If this happens, then instead of verifying the judgment using the external decision procedure, it is written into the output procap. During file access, the file system backend *substitutes* the actual time of access for the constant ctime in the judgment and checks it (see Appendix B for details).

Symbolic constants similar to ctime have been used to represent access policies in the past [8, 10]. However, in each of these cases, the constant is a part of the logic and can be used within a policy (similar to our interpreted predicates). In contrast, we use the constant as an enforcement technique only; time in the logic is represented using the @ connective.

Procap structure. In summary, a procap contains four components $\langle \psi, \vec{i}, \vec{C}, \Xi \rangle$, where

- $\psi = \langle k, f, \eta \rangle$ is a three-tuple that lists the principal, file, and permission that the procap authorizes.
- $\vec{i}$ is a list of interpreted predicates on which the verified proof depends (point (2) above).
- $\vec{C}$ is a list of judgments $\Psi \models u_1 \leq u_2$ that contain the constant ctime, and on which the proof depends (point (3) above). In most cases Ψ is $\cdot$.
- Ξ is a cryptographic signature over the first three components. This guarantees the procap's authenticity.

Procap verification. Before admitting a procap, the file system backend must check not only its signature Ξ, but also the interpreted predicates $\vec{i}$ (in the state prevalent at the time of access) and the constraint judgments in the list $\vec{C}$ (with ctime substituted by the actual time of access). The following (informally stated) theorem shows that these checks guarantee that the proof in lieu of which the procap was obtained authorizes the operation at the actual time of access. A precise formalization of the verification procedure, a formal statement of this theorem, and its proof are presented in Appendix B.

Theorem 4.1 (Enforcement correctness)**.** *Suppose that the verification of a proof term M which establishes the right $\psi = \langle k, f, \eta \rangle$ from policy Γ results in a procap $\langle \psi, \vec{i}, \vec{C}, \Xi \rangle$. Further let E be a file system state, which occurs at some time u, and assume that:*

1. *For each $i \in \vec{i}$, $E \models i$*
2. *For each $(\Psi \models u_1 \leq u_2) \in \vec{C}$, $\Psi[u/\mathsf{ctime}] \models c[u/\mathsf{ctime}]$*

Then, $M :: \Sigma; \cdot; E; \Gamma \xrightarrow{\alpha} \mathtt{auth}(k, f, \eta, u)$.

Example 2. At the end of Example 1, we constructed a proof term M which established $E; \Gamma \xrightarrow{\alpha} (\mathsf{admin}\ \mathsf{says}\ \mathtt{may}(\mathsf{uid}\ 1500, /\mathsf{secret.txt}, \mathsf{read})) \circ [2008, 2009]$, where E was required to satisfy the two conditions (A) and (B). If we give this proof term to our proof verifier, the resulting procap has the structure $\langle \psi, \vec{i}, \vec{C}, \Xi \rangle$, where

- $\psi = \langle \mathsf{uid}\ 1500, /\mathsf{secret.txt}, \mathsf{read} \rangle$
- $\vec{i} = \mathbf{owner}(/\mathsf{secret.txt}, \mathsf{uid}\ 1003),\ \mathbf{has\_xattr}(/\mathsf{secret.txt}, \mathtt{level}, \mathsf{secret})$
- $\vec{C} = \cdot \models 2008{:}01{:}01{:}00{:}00{:}00 \leq \mathsf{ctime},\ \cdot \models \mathsf{ctime} \leq 2009{:}12{:}31{:}23{:}59{:}59$

The predicates in list $\vec{i}$ imply that the procap is valid only in a state where /secret.txt is owned by Alice, and it has extended attribute `user.#pcfs.level` set to secret. These correspond exactly to conditions (A) and (B) from Example 1, and are necessary for the proof term M to be valid. The list $\vec{C}$ means that the time of access ctime must lie in the interval $[2008, 2009]$, which is also what we may expect from the policy rules.

Certificate Revocation. A revocation refers to the withdrawal of a policy rule or fact after it has been created but before it expires. Revocations are an issue for enforcement because proofs and procaps depending on a revoked statement may already have been generated. There are two simple ways to enforce revocations using procaps, both of which we describe briefly. (The current implementation of PCFS does not implement revocation, but either of these methods is easy to add.)

- A list of unique ids of certificates on which a proof depends can be included in the procap generated from it. Before admitting a procap, the file system backend can compare the list of certificate ids in it to a list of revoked certificates provided by administrators. If there is an overlap, the procap can be rejected. Although this method would enforce revocation perfectly, it would also slow down file access because an additional check must be performed on each procap.
- Alternatively, the list of revoked certificates can be provided to the proof verifier instead of the file system backend. The verifier can then refuse to accept any proof that depends on revoked certificates. If the verifier issues a procap, it can be short lived, i.e., its validity can be restricted to a short duration T using a constraint on ctime. Although the effect of revocation in this method is not immediate (it can lag by a time T), the file system backend does not get involved, so its performance is not affected.

5 PCFS Backend

Whereas the front end of PCFS is used to generate procaps from proofs of access, the backend grants access to files and directories using the procaps to check access rights. The two ends are linked by procaps only; indeed the backend of the file system is entirely agnostic to the logic used. If we had a different logic for writing the policy, we could use the same backend, so long as the logic's proof verifier generated the same procaps. Since the backend is called at every single file access, it needs to be extremely efficient. In this section, we discuss its design and implementation.

Overall architecture. The PCFS backend is implemented as a process server, which listens to upcalls made by the kernel module Fuse. The latter happens whenever a process makes a system call to access a file or directory within the mount path of PCFS. Depending on the operation requested, a specific handling function is invoked. There is one function for every POSIX file system operation like open, read, write, stat, unlink, rmdir, mkdir, etc. This handling function looks up and checks procaps corresponding to permissions needed to perform the operation. If the checks succeed, it invokes an identical file system call, but on a different mount path, which is actually an ext3 file system. In order to bypass any access checks during the call to the ext3 file system, the process server runs with superuser privileges. To prevent users from directly using the ext3 file system to access data, we give ownership of the root directory on the ext3 file system to the superuser, and turn off all access on it.[2]

Organization of the file system. For the purpose of illustration let us assume that PCFS is mounted at `/pcfs`, and that it makes calls to the ext3 file system at `/src`. Then `/pcfs` mirrors the file system structure rooted at `/src`, except that all calls within the former are subject to procap based checks. A special directory `/pcfs/#config` contains configuration data for the file system, including procaps and the secret key used to sign them. This directory is protected by the file system with strict rules that do not use procaps. We list below some of the important files and directories in this special directory, as well their contents and protections.

> `/pcfs/#config/config-file`: File containing configuration options, including the user ids of the principals admin and pcfssystem. (Recall from Section 4 that pcfssystem is the only user who has access to the secret key needed to sign procaps.) Anyone can read this file, but only pcfssystem can change this file.
>
> `/pcfs/#config/shared-key`: Contains the shared key used to sign procaps. Only pcfssystem may read or write this file.
>
> `/pcfs/#config/ca-pubkey.pem`: Contains the public key of the certifying authority who signs associations between other public keys and users. Anyone may read this file, but only pcfssystem may write to it.

[2] A more secure method to prevent access via the underlying file system is to keep data encrypted on it, and to decrypt data in our process server. We have not implemented this design, since our objective here is to evaluate the performance of access checks.

`/pcfs/#config/procaps/`: This directory contains the procaps. Its organization is discussed next. pcfssystem has full access to this directory, and other users have access to specific subdirectories only.

The procap giving the right $\langle k, f, \eta \rangle$, subject to access-time conditions as discussed before, is stored in the file `/pcfs/#config/procaps/<`k`>/<`f`>.perm.<`η`>`. Here `<`k`>` is the user id of the user k, `<`f`>` is the path of the file f (relative to the mount point), and `<`η`>` is a textual representation of the permission. Thus each procap is stored in a separate file, and further for each right $\langle k, f, \eta \rangle$, there can be at most one procap that authorizes the right. While this may be restrictive, it makes look up extremely easy since the exact path where a procap is to be found can be determined simply by knowing the PCFS mount point and the right $\langle k, f, \eta \rangle$. To prevent denial of service attacks and to protect user privacy, the PCFS server ensures that only user k can access (read, write, or delete) files inside `/pcfs/#config/procaps/<`k`>/`.

Since pcfssystem has full access to all files and directories within `/pcfs/#config/`, it is a very attractive target for attack. In particular, if an attacker gains control of this user account, it can change the secret key used to sign and verify procaps, and then inject fake procaps to access other files. To prevent this, the PCFS process server denies pcfssystem all rights in *other* directories within the file system. Thus, to attack PCFS through this mechanism, the attacker must break into at least one more account in addition to pcfssystem.

Procap Cache. Since procaps are stored in files, and one or more of them must be read for every file system operation, it is helpful to cache commonly used procaps in memory to improve performance. To this end, PCFS uses a least recently used (LRU) in-memory cache, whose size can be adjusted at mount time. The cache stores parsed procaps, whose signatures have already been verified. The only cost involved in using a cached procap is checking its conditions (lists $\vec{C}$ and $\vec{\imath}$ from Section 4). This is extremely fast and usually takes only 10–100μs. As a result, PCFS obtains extremely high performance when the number of files in use is small. We evaluate the effect of the cache in Section 6.

Permissions. PCFS uses five distinct permissions on any file or directory: read, write, execute, identity, and govern. (In contrast, POSIX mandates only the first three permissions.) Permissions read and write are the obvious ones; they are needed to read and to change the contents of a file/directory respectively. As usual, execute is the permission to read the meta data of a file or directory. Permission identity is needed to delete a file or directory, or to rename it. This permission is separated from others because in many settings, administrators may not want to allow users to delete or rename shared files, but perform other operations on them (and their parent directories). The govern permission is needed to change the owner of a file and to change extended attributes starting with the prefix `user.#pcfs`. Because of this special protection, these attributes can be used by administrators to give classification or security labels to files, as in Example 1. Figure 3 lists the permissions needed to perform some common file system operations. During a file system call, procaps corresponding to the relevant entry in this table are looked up and checked. By separating the identity and govern permissions from write, we allow for the possibility of easily administering both mandatory and discretionary access

Operation	Permissions needed
stat `/foo`	execute on `/foo`
open `/foo` in read mode	read on `/foo`
open `/foo` in write mode	write on `/foo`
create `/bar/foo`	write on `/bar`
delete `/bar/foo`	identity on `/bar/foo`
rename `/bar` to `/foo`	identity on `/bar`, write on `/foo`
getxattr on `/foo`	execute on `/foo`
setxattr on `/foo`	govern on `/foo` if attribute starts with `user.#pcfs.`, write otherwise
chown on `/foo`	govern on `/foo`

Figure 3: Permissions needed to perform some operations

control in PCFS. This is difficult with POSIX permissions, where the owner always has all permissions on a file.

Default Permissions. When a program first creates a file, it cannot be assumed that any policy rules apply to it, since that (usually) requires creation of certificates by administrators. Yet, many programs create temporary files, to which they must have access in order to complete their tasks. To allow such programs to execute correctly, when a new file or directory is created, PCFS automatically creates and stores *default procaps* that give the creator of the file read, write, execute, and identity permissions for a fixed period of time (this period can be adjusted at mount time). In addition the user admin is given execute and govern rights on the new file. After this period elapses, the default procaps expire and the administrators must create policy rules to control access to the file.

6 Evaluation

We evaluate PCFS in two ways. First, we report the results of performance benchmarks on the backend of the file system. Second, we comment on the expressiveness of the framework through two case studies.

6.1 Performance of the Backend

Since we are primarily interested in measuring the overhead of access control checks due to procaps, our baseline for comparing performance is a Fuse-based file system that does not perform the corresponding checks, but is otherwise running a process server and using an underlying ext3 file system, just as PCFS does. We call this file system Fuse/Null. For macrobenchmarks we also compare with an ext3 file system. All measurements reported here were made on a 2.4GHz Core Duo 2 machine with 3GB RAM and a 7200RPM 100GB hard disk drive, running the Linux kernel 2.6.24-23.

Read and write throughput. By default, PCFS does not make any access checks when read or write operations are performed on a previously opened file. Instead access checks are made when the file is opened. As a result its read and write throughput is very close to that of Fuse/Null. The following table summarizes the read and write throughputs of PCFS and Fuse/Null based on reading and writing a 1GB file sequentially using the Bonnie++ test suite [1].

Operation	PCFS (MB/s)	Fuse/Null (MB/s)
Read	538.69	567.47
Write	73.18	76.05

It is possible, through a mount time option, to force PCFS to check procaps that authorize read and write access during read and write operations respectively. As long as the procaps checked are cached in memory, this does not affect performance at all since the time taken to check a cached procap is only a few microseconds.

File stats. Besides read and write, two other very common file operations are open and stat (reading a file's meta data). In terms of access checks, both are similar, since usually one procap must be checked in each case.[3] The table below shows the speed of the stat operation in PCFS with different hit rates in the procap cache. All measurements are reported in number of operations per second, as well as time taken per operation. The title $n\%$ indicates a measurement with a cache hit rate of $n\%$. For comparison, performance of Fuse/Null is also shown. The figures are based on choosing a random file 20,000 times in a directory containing exactly 20,000 files, and stating it. To get a hit rate of $n\%$, the cache size is set to $n/100 \times 20000$, and the cache is warmed a priori with random procaps. It is easy to prove that for an LRU cache this results in a hit rate of exactly $n\%$ when subsequent files (procaps) are also chosen at random.

Cache hit rate →	0%	50%	90%	95%	98%	100%	Fuse/Null
Stats per second	5774	7186	8871	9851	11879	23652	36042
Time per stat (μs)	173.2	139.2	112.7	101.5	84.2	42.2	27.7

As can be seen from this table, the procap cache is extremely helpful in attaining efficiency. The difference of the times in the last two columns is an estimate of the time it takes to check a cached procap (i.e., the time needed to check the conditions in a procap). In this case, this time is $42.2 - 27.7 = 14.5\mu$s. This estimate is rough, and the actual time varies with the complexity of the conditions in the procap. The procaps used here are default ones. In other experiments, we have found that this time varies from 10 to 100μs. By taking the difference of the times in first and last columns, we obtain an estimate of the time required to read a procap, check its signature, parse the procap, and check its conditions. In this experiment, this time is $173.2 - 27.7 = 145.5\mu$s. Additional time may be needed to seek to the procap on disk, which is not counted here. This suggests that, in general, procap checking is dominated by reading and parsing times. The signatures we use for procaps are message authentication codes or MACs, which can be verified in 1 to 2μs.

[3] Two procaps must be checked when a file is opened in read and write modes simultaneously.

File creation and deletion. The table below lists the number of create and delete operations per second that are supported by PCFS and Fuse/Null. These are measured by creating and deleting 10,000 files in a single directory.

Operation	PCFS (op/s)	Fuse/Null (op/s)
Create	1386	4738
Delete	1989	15429

PCFS is approximately 3.5 times slower than FUSE/Null in creating files. This is because PCFS also creates six default procaps for every file created. As a result, it creates seven times as many files in three separate directories. Deletion in PCFS is nearly 7.7 times slower than that in Fuse/Null. This is because when a file is deleted in PCFS, one procap must be looked up, parsed, and checked, and all procaps related to the file must later be deleted. This is done to avoid accumulating useless procaps; it can be turned off using a mount time option. In this case, each file deletion corresponds to seven file deletions on the ext3 file system in three different directories. The effect of the procap cache is negligible during create and delete operations.

In summary, assuming a low rate of cache misses, the performance of PCFS on common file operations like read, write, stat, and open is comparable to that of Fuse/Null. On the other hand, less common operations like create and delete are slower because default procaps must be managed.

Macrobenchmarks. To understand the performance of PCFS in practice, we also ran two simple macrobenchmarks. The first (called OpenSSL in the table below), untars the OpenSSL source code, compiles it and deletes it. The other (called Fuse in the table below), performs similar operations for the source of the fuse kernel module five times in sequence. As can be seen, the performance penalty for PCFS as compared to Fuse/Null is approximately 10% for OpenSSL, and 2.5% for Fuse. The difference arises because the OpenSSL benchmark depends more on file creation and deletion as compared to the Fuse benchmark.

Benchmark	PCFS (s)	Fuse/Null (s)	Ext3 (s)
OpenSSL	126	114	94
Fuse × 5	79	77	70

6.2 Case Studies

We have also completed two case studies using BL and PCFS. In each case, we expressed the policy from the case study in BL, and considered whether it could be enforced in PCFS.

Classified Information. Our first case study formalizes rules for control and dissemination of classified information among intelligence agencies in the U.S. (Examples 1 and 2 are based on this case study.) The enforcement of these rules was also the original motivation for building PCFS. We obtained information on these rules from public government documents, and through an industrial collaborator. This information was distilled to 35 formulas in BL. The study is interesting because it uses almost all features of BL. Extended attributes are used to represent the classification status of files (classified vs unclassified), and their classification level as in Example 1. Attributes of individuals are specified in certificates issued by administrators, many of which expire at fixed points of time. For example, some background checks expire every 5 years. These expirations are represented using the @ connective in BL. Also, one of the rules requires arithmetic over time – the owner of an unclassified file can access it for 90 days *after* its creation. (BL supports linear arithmetic over time, but we have not discussed it in this paper.)

Some of the proofs needed for access in this study are quite large; they contain as many as 1100 proof steps, and depend on 70 certificates. It takes nearly 100ms to verify these proofs. This strongly supports the case for performing proof verification ahead of access and using capabilities, as PCFS does. If such proof verifications were performed during file access, the file system interface would be limited to less that 10 operations a second.

Course administration. In our second case study we formalize the rules for controlling permissions on directories for storing class materials and assignments, based on current workflows at our university. Although these rules are much simpler than those in the previous study, we had to add support for a new kind of interpreted predicate: $\mathbf{member}(f, d)$, which means that file f is contained in directory d. This effort gave us an idea about the difficulty involved in extending PCFS (and BL) with new interpreted predicates. In all, it took us less than 10 minutes of programming effort to add support for this new predicate. (All parsers in our implementation already support parsing of unknown predicates, so we only had to define a procedure for verifying the predicate.)

7 Related Work

A lot of prior work is related to PCFS; here we describe only the most closely related work.

Relation to PCA. Proof-carrying authorization (PCA), the general architecture on which PCFS builds, has been implemented in several other systems [8, 9, 25]. However, PCFS differs from these systems in several ways. The most significant difference is that in all existing PCA-based systems, the proof that the user constructs is given directly to the system interface at the time of access. As a result, the proof verifier must be called *every time an access is requested.* This design works well only when the time taken to check certificates and the proof (typically several milliseconds) is not significant in comparison to the time taken to perform the actual operation. This has been the case in all implementations of PCA to date. In contrast, a file system access is a fast operation that takes of the order of a few micro or milli seconds only, and checking several certificates and a proof at each access results in visible delays for the user. We actually confirmed this hypothesis through an earlier implementation of PCFS that used the PCA architecture directly. As a result of this prior experience, in the present design of PCFS, proofs are verified in advance of performing operations, and capabilities issued in return are used to authorize access.

Second, the logic used in in PCFS (discussed in Section 3) contains explicit time. This allows accurate representation of expiration dates of policy rules in logical formulas and also in proofs. In contrast, logics used so far in PCA systems are unaware of time, and rule expiration is enforced using an extra-logical mechanism. Having time in the logic also allows more expressive rules, e.g., those that use arithmetic over time.

Third, in all existing implementations using PCA, the user is *authenticated* to the system interface using a challenge response protocol with a fresh nonce. This nonce must be embedded in the proof used to authorize access because the interface does not learn the identity of the user. This implies that the proof cannot be completed in advance of the access (although some parts of it that are independent of the nonce can be). Owing to concerns regarding efficiency, we do not consider this style of authentication a good design principle for PCA. Instead, we believe that the authentication protocol used should tell the system interface the identity of the user. In distributed settings a password or public key can be used for authentication, and in centralized settings like PCFS the system interface can learn the user id of the calling process through a system call like `getuid()`. This form of authentication allows proofs of access to be created and checked in advance of access, which is central to obtaining efficiency in PCFS.

Other related work. Many prior file systems have used capabilities to authorize access (e.g., [6, 20, 28, 31, 32]), although the use of proofs to generate capabilities is novel to our work. Prior work by Chaudhuri considers a formal analysis of correctness of an implementation of authorization through cryptographic capabilities in the face of dynamic policies [13]. That paper also considers many strategies for enforcing time-dependent and state-dependent policies, but the mechanism used to generate policies is treated abstractly (in contrast, in Theorem 4.1, we prove our enforcement correct with respect to a concrete logic and proof system).

Many logics and logic-based languages have been proposed in the past for representing access control policies (e.g., [4, 5, 10, 14, 18, 21, 30]). The k says s modality in BL is most closely related to a similar modality in Binder [14]. Our treatment of explicit time draws on work by DeYoung et al. [16]. We believe that the combination of time and interpreted predicates is novel to BL. The implementation of the proof search tool for BL builds upon work on uniform proofs for logic programming [27], and draws on ideas from the language Lolli [22].

8 Conclusion

PCFS combines strong logical foundations for access policies with an efficient enforcement based on proofs and cryptographic capabilities. Owing to a very expressive logic for policies, and conditions in capabilities, PCFS automatically enforces time-dependent policy rules, as well as policies that depend on file system state. A significant contribution of our work is Theorem 4.1 which shows that enforcement of policies using procaps is sound with respect to enforcement with proofs directly (as in PCA).

A number of interesting avenues remain for future work that we discuss here briefly. One interesting direction is to apply the PCFS architecture to build a networked file system, with the proof verifier and storage on separate nodes, and a decentralized store for procaps. Procaps already support decentralization, since their integrity is protected by the signature contained in them. Another interesting line of work may be to use capabilities to implement access control on devices that have little computational power (e.g., embedded devices), and support them with the existing front end from PCFS that runs on a separate machine. A third subject of interest is to consider more case studies of policies used in practice to see if they can be expressed and enforced in PCFS.

References

[1] Bonnie++. Available from http://www.coker.com.au/bonnie++/.

[2] FUSE: Filesystem in Userspace. Available from http://fuse.sourceforge.net/.

[3] SecPAL research release for .NET, 2007. Online at http://research.microsoft.com/en-us/projects/secpal/.

[4] Martín Abadi. Access control in a core calculus of dependency. *Electronic Notes in Theoretical Computer Science*, 172:5–31, April 2007. *Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin.*

[5] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[6] Marcos K. Aguilera, Minwen Ji, Mark Lillibridge, John MacCormick, Erwin Oertli, Dave Andersen, Mike Burrows, Timothy Mann, and Chandramohan A. Thekkath. Block-level security for network-attached disks. In *Proceedings of the 2nd Conference on File and Storage Technologies (FAST)*, pages 159–174, 2003.

[7] Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In G. Tsudik, editor, *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 52–62, Singapore, November 1999. ACM Press.

[8] Lujo Bauer. *Access Control for the Web via Proof-Carrying Authorization.* PhD thesis, Princeton University, November 2003.

[9] Lujo Bauer, Scott Garriss, Jonathan M. McCune, Michael K. Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the Grey system. In *Proceedings of the 8th International Conference on Information Security (ISC'05)*, pages 431–445. Springer LNCS 3650, September 2005.

[10] Moritz Y. Becker, Cédric Fournet, and Andrew D. Gordon. Design and semantics of a decentralized authorization language. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF-20)*, pages 3–15, 2007.

[11] Moritz Y. Becker and Peter Sewell. Cassandra: Flexible trust management applied to health records. In *Proceedings of 17th IEEE Computer Security Foundations Workshop (CSFW-17)*, pages 139–154, 2004.

[12] Bor-Yuh Evan Chang, Kaustuv Chaudhuri, and Frank Pfenning. A judgmental analysis of linear logic. Technical Report CMU-CS-03-131R, Carnegie Mellon University, 2003.

[13] Avik Chaudhuri. On secure distributed implementations of dynamic access control. In *Proceedings of the Joint Workshop on Foundations of Computer Security, Automated Reasoning for Security Protocol Analysis, and Issues in the Theory of Security (FCS-ARSPA-WITS)*, pages 93–107, 2008.

[14] John DeTreville. Binder, a logic-based security language. In M. Abadi and S. Bellovin, editors, *Proceedings of the Symposium on Security and Privacy (S&P'02)*, pages 105–113, Berkeley, California, May 2002. IEEE Computer Society Press.

[15] Henry DeYoung. A logic for reasoning about time-dependent access control policies. Technical Report CMU-CS-08-131, Computer Science Department, Carnegie Mellon University, December 2008.

[16] Henry DeYoung, Deepak Garg, and Frank Pfenning. An authorization logic with explicit time. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF-21)*, Pittsburgh, Pennsylvania, June 2008. Extended version available as Technical Report CMU-CS-07-166.

[17] Deepak Garg. Proof search in an authorization logic. Technical Report CMU-CS-09-121, Carnegie Mellon University, June 2009.

[18] Deepak Garg and Frank Pfenning. Non-interference in constructive authorization logic. In J. Guttman, editor, *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19)*, pages 283–293, Venice, Italy, July 2006.

[19] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131, North-Holland, 1969.

[20] H. Gobioff, G. Gibson, and D. Tygar. Security for network attached storage devices. Technical Report CMU-CS-97-185, Carnegie Mellon University, 1997.

[21] Yuri Gurevich and Itay Neeman. DKAL: Distributed-knowledge authorization language. In *Proceedings of the 21st IEEE Symposium on Computer Security Foundations (CSF-21)*, pages 149–162, June 2008.

[22] Joshua S. Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994.

[23] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 public key infrastructure. See http://www.ietf.org/rfc/rfc2459.txt, 1999.

[24] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.

[25] Chris Lesniewski-Laas, Bryan Ford, Jacob Strauss, Robert Morris, and M. Frans Kaashoek. Alpaca: Extensible authorization for distributed services. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07)*, Alexandria, VA, October 2007.

[26] Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages (PADL'03)*, pages 58–73. Springer LNCS 2562, 2003.

[27] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[28] Christopher Olson and Ethan L. Miller. Secure capabilities for a petabyte-scale object-based distributed file system. In *Proceedings of the ACM Workshop on Storage Security and Survivability (StorageSS'05)*, pages 64–73, 2005.

[29] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11:511–540, 2001.

[30] Andrew Pimlott and Oleg Kiselyov. Soutei, a logic-based trust-management system. In *Proceedings of the Eighth International Symposium on Functional and Logic Programming (FLOPS'06)*, pages 130–145, 2006.

[31] Benjamin C. Reed, Edward G. Chron, Randal C. Burns, and Darrell D. E. Long. Authenticating network-attached storage. *IEEE Micro*, 20(1):49–57, 2000.

[32] Jude T. Regan and Christian D. Jensen. Capability file names: Separating authorisation from user management in an internet file system. In *Proceedings of the 10th USENIX Security Symposium (SSYM'01)*, August 2001.

A Description of the Logic BL

This appendix describes the proof system of the logic BL, and its meta theory. The syntax of the logic was presented in Section 3. Proof terms M are summarized below:

$$\begin{array}{lll} M & ::= & x \mid \texttt{pf\_conjI}\ M_1\ M_2 \mid \texttt{pf\_conjE1}\ M \mid \texttt{pf\_conjE2}\ M \mid \texttt{pf\_disjI1}\ M \mid \\ & & \texttt{pf\_disjI2}\ M \mid \texttt{pf\_disjE}\ M\ ([x]M_1)\ ([y]M_2) \mid \texttt{pf\_topI} \mid \texttt{pf\_botE}\ M \mid \\ & & \texttt{pf\_impI}\ ([x][v_1][v_2]M) \mid \texttt{pf\_impE}\ M_1\ M_2\ u_1\ u_2 \mid \texttt{pf\_forallI}\ ([v]M) \mid \\ & & \texttt{pf\_forallE}\ t\ M \mid \texttt{pf\_existsI}\ t\ M \mid \texttt{pf\_existsE}\ M_1\ ([x][v]M_2) \mid \\ & & \texttt{pf\_atI}\ M \mid \texttt{pf\_atE}\ M_1\ ([x]M_2) \mid \texttt{pf\_saysI}\ M \mid \texttt{pf\_saysE}\ M_1\ ([x]M_2) \mid \\ & & \texttt{pf\_sinjI} \mid \texttt{pf\_sinjE}\ M_1\ M_2 \mid \texttt{pf\_cinjI} \mid \texttt{pf\_cinjE}\ M_1\ M_2 \end{array}$$

Variables x, v in square brackets $[x], [v]$ are binding occurrences. Bound variables may be α-renamed implicitly.

Figures 4 and 5 list the rules of the natural deduction system for BL. All rules in Figure 4 are similar to corresponding rules in prior work by DeYoung [15, Chapter 5], done in the context of η-logic [16]. There are only two minor differences: (a) Our rules contain the view α and the state E both of which remain unchanged in all rules of Figure 4, and (b) BL contains the connective $\bot$ (rule $\bot$E), which η-logic does not. For descriptions of the rules in Figure 4, we refer the reader to the prior work by DeYoung.

Rules in Figure 5 are peculiar to BL. Rule (hyp) states that the assumption $s \circ [u_1', u_2']$ entails $s \circ [u_1, u_2]$ if $u_1' \leq u_1$ and $u_2 \leq u_2'$, i.e., the interval $[u_1, u_2]$ is a subset of the interval $[u_1', u_2']$. This makes intuitive sense: if a formula s holds throughout an interval, it must hold on every subinterval as well. The proof term corresponding to this (trivial) derivation is x, where x is also the name for the assumption $s \circ [u_1', u_2']$. The rule (claims) is similar, except that it allows us to conclude $s \circ [u_1, u_2]$ from the assumption k' claims $s \circ [u_1', u_2']$. In this case, it must also be shown, among other things, that k' is stronger than the principal k in the view (premise $\Psi \models k' \succeq k$).

(saysI) is the only rule which changes the view. The notation $\Gamma|$ in this rule denotes the subset of Γ that contains exactly the claims of principals, i.e., the set $\{(k'\ \mathsf{claims}\ s' \circ [u_1', u_2']) \in \Gamma\}$. The rule means that $(k\ \mathsf{says}\ s) \circ [u_1, u_2]$ holds in any view α if $s \circ [u_1, u_2]$ holds in the view k, u_1, u_2 using only claims of principals. Assumptions of the form $s' \circ [u_1', u_2']$ are eliminated from Γ in the premise because they may have been added in the view α, but may not hold in the view k, u_1, u_2. Its dual rule (saysE) states

$$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1, u_2] \qquad M_2 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1, u_2]}{(\mathtt{pf\_conjI}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \wedge s_2 \circ [u_1, u_2]}\wedge\mathrm{I}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \wedge s_2 \circ [u_1, u_2]}{(\mathtt{pf\_conjE1}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1, u_2]}\wedge\mathrm{E1} \qquad \frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \wedge s_2 \circ [u_1, u_2]}{(\mathtt{pf\_conjE2}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1, u_2]}\wedge\mathrm{E2}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1, u_2]}{(\mathtt{pf\_disjI1}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \vee s_2 \circ [u_1, u_2]}\vee\mathrm{I1} \qquad \frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1, u_2]}{(\mathtt{pf\_disjI2}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \vee s_2 \circ [u_1, u_2]}\vee\mathrm{I2}$$

$$\frac{\begin{array}{c} M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \vee s_2 \circ [u_1, u_2] \\ M_1 :: \Sigma; \Psi; E; \Gamma, x : s_1 \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u_1', u_2'] \qquad M_2 :: \Sigma; \Psi; E; \Gamma, y : s_2 \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u_1', u_2'] \end{array}}{(\mathtt{pf\_disjE}\ M\ ([x]M_1)\ ([y]M_2)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']}\vee\mathrm{E}$$

$$\frac{}{(\mathtt{pf\_topI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} \top \circ [u_1, u_2]}\top\mathrm{I} \qquad \frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} \bot \circ [u_1, u_2]}{(\mathtt{pf\_botE}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1', u_2']}\bot\mathrm{E}$$

$$\frac{M :: \Sigma, v_1{:}\mathsf{time}, v_2{:}\mathsf{time}; \Psi, u_1 \leq v_1, v_2 \leq u_2; E; \Gamma, x : s_1 \circ [v_1, v_2] \xrightarrow{\alpha} s_2 \circ [v_1, v_2]}{(\mathtt{pf\_impI}\ ([x][v_1][v_2]M)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \supset s_2 \circ [u_1, u_2]}\supset\mathrm{I}$$

$$\frac{\begin{array}{c} M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \supset s_2 \circ [u_1, u_2] \\ M_2 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1', u_2'] \qquad \Psi \models u_1 \leq u_1' \leq u_1'' \qquad \Psi \models u_2'' \leq u_2' \leq u_2 \end{array}}{(\mathtt{pf\_impE}\ M_1\ M_2\ u_1'\ u_2') :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1'', u_2'']}\supset\mathrm{E}$$

$$\frac{M :: \Sigma, v{:}\sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]}{(\mathtt{pf\_forallI}\ ([v]M)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} \forall v{:}\sigma.s \circ [u_1, u_2]}\forall\mathrm{I}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} \forall v{:}\sigma.s \circ [u_1, u_2] \qquad \Sigma \vdash t : \sigma}{(\mathtt{pf\_forallE}\ M\ t) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s[t/v] \circ [u_1, u_2]}\forall\mathrm{E} \qquad \frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s[t/v] \circ [u_1, u_2] \qquad \Sigma \vdash t : \sigma}{(\mathtt{pf\_existsI}\ t\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} \exists v{:}\sigma.s \circ [u_1, u_2]}\exists\mathrm{I}$$

$$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} \exists v{:}\sigma.s \circ [u_1, u_2] \qquad M_2 :: \Sigma, v{:}\sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u_1', u_2']}{(\mathtt{pf\_existsE}\ M_1\ ([v][x]M_2)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']}\exists\mathrm{E}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]}{(\mathtt{pf\_atI}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (s \mathbin{@} [u_1, u_2]) \circ [u_1', u_2']}@\mathrm{I}$$

$$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (s \mathbin{@} [u_1, u_2]) \circ [u_1', u_2'] \qquad M_2 :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u_1'', u_2'']}{(\mathtt{pf\_atE}\ M_1\ ([x]M_2)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_1'', u_2'']}@\mathrm{E}$$

Figure 4: BL: Natural Deduction, Part 1

that a proof of k says $s \circ [u_1, u_2]$ can be used to justify an assumption of the equivalent judgmental form k claims $s \circ [u_1, u_2]$.

The rule (interI) is used to establish interpreted predicates. It states that an interpreted atomic formula i is provable if it holds in the abstract logical representation of the environment E. The proof term `sinjI` has no specific structure; it is merely a marker which means that a procedure must be invoked to check i in the prevailing environment. Its dual rule (interE) states that any proof of $i \circ [u_1, u_2]$ justifies the addition of i to the environment. In particular, the time interval $[u_1, u_2]$ associated with an interpreted predicate is meaningless. Rules (consI) and (consE) are similar but they are used to establish and eliminate constraints reified as formulas.

$$\frac{\Psi \models u_1' \leq u_1 \qquad \Psi \models u_2 \leq u_2'}{x :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]}\,\text{hyp}$$

$$\frac{\alpha = k, u_b, u_e \qquad \Psi \models u_1' \leq u_1 \qquad \Psi \models u_2 \leq u_2' \qquad \Psi \models u_1' \leq u_b \qquad \Psi \models u_e \leq u_2' \qquad \Psi \models k' \succeq k}{x :: \Sigma; \Psi; E; \Gamma, x : k' \; \mathsf{claims} \; s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]}\,\text{claims}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma| \xrightarrow{k, u_1, u_2} s \circ [u_1, u_2]}{(\texttt{pf\_saysI} \; M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (k \; \mathsf{says} \; s) \circ [u_1, u_2]}\,\mathsf{saysI}$$

$$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (k \; \mathsf{says} \; s) \circ [u_1, u_2] \qquad M_2 :: \Sigma; \Psi; E; \Gamma, x : k \; \mathsf{claims} \; s \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u_1', u_2']}{(\texttt{pf\_saysE} \; M_1 \; ([x] M_2)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']}\,\text{saysE}$$

$$\frac{E \models i}{(\texttt{pf\_sinjI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} i \circ [u_1, u_2]}\,\text{interI}$$

$$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} i \circ [u_1, u_2] \qquad M_2 :: \Sigma; \Psi; E, i; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']}{(\texttt{pf\_sinjE} \; M_1 \; M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']}\,\text{interE}$$

$$\frac{\Psi \models c}{(\texttt{pf\_cinjI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} c \circ [u_1, u_2]}\,\text{consI}$$

$$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} c \circ [u_1, u_2] \qquad M_2 :: \Sigma; \Psi, c; E; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']}{(\texttt{pf\_cinjE} \; M_1 \; M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']}\,\text{consE}$$

Figure 5: BL: Natural Deduction, Part 2

Explanation of views. The use of views α is unique to BL, at least in the context of authorization logics, and we would like to explain it further. In general, a view $\alpha = k, u_b, u_e$ can be thought of as consisting of two components: the principal k and the interval $[u_b, u_e]$. Since a view changes only in the rule (saysI), the view k, u_b, u_e on any sequent in a proof is determined by the most recently encountered goal judgment $(k \; \mathsf{says} \; s) \circ [u_b, u_e]$. The importance of the view is that it prevents the use of any assumptions of the form $k' \; \mathsf{claims} \; s' \circ [u_b', u_e']$ unless either (a) the view changes due to another (saysI) rule, or (b) k' is stronger than k and $[u_b', u_e']$ is a superset of the interval $[u_b, u_e]$. These are forced by the premises of the rule (claims).

Restriction (b) is important in practice. Suppose we try to establish the goal $(\mathsf{admin} \; \mathsf{says} \; \texttt{may}(\mathsf{Alice}, /\mathsf{secret.txt}, \mathsf{read})) \circ [2009, 2011]$ to allow Alice to read file /secret.txt from 2009 to 2011. The use of views ensures that (unless a (saysI) is encountered in a subgoal), this proof will *only* depend on credentials and policies that are (1) created by principals stronger than admin, and (2) valid on intervals that include $[2009, 2011]$. If we omit principals from views, we might violate (1), allowing principals without proper authority to influence an authorization. If we omit intervals from views, we run the risk of allowing expired credentials or those that are applicable in the future to affect an authorization. Neither of these is desirable. On a more formal note, we expect that the use of views in BL will lead to strong non-interference theorems, like those established for an earlier logic without explicit time [18], allowing us to formalize these intuitions.

A.1 Meta-Theory

We prove some interesting meta-theoretic properties of BL, including those mentioned in Section 3. In order to prove these properties we make the following assumptions about decision procedures for constraints and interpreted predicates. In particular, we allow free parameters (constants) in the judgments $\Psi \models c$ and $E \models i$, and assume that the decision procedures treat these parameters universally, i.e, truth of judgments is closed under substitution of these parameters by ground terms. This is formalized by the assumptions (Substitution-cons) and (Substitution-state) below.

1. (Hypothesis) $\Psi, c \models c$.
2. (Weakening-cons) $\Psi \models c$ implies $\Psi, c' \models c$.
3. (Cut-cons) $\Psi \models c$ and $\Psi, c \models c'$ imply $\Psi \models c'$.
4. (Substitution-cons) $\Psi \models c$ implies $\Psi[t/v] \models c[t/v]$.
5. (Refl-time) $\Psi \models u \leq u$.
6. (Trans-time) $\Psi \models u \leq u'$ and $\Psi \models u' \leq u''$ imply $\Psi \models u \leq u''$.
7. (Refl-prin) $\Psi \models k \succeq k$.
8. (Trans-prin) $\Psi \models k \succeq k'$ and $\Psi \models k' \succeq k''$ imply $\Psi \models k \succeq k''$.
9. (Weakening-state) $E \models i$ implies $E, E' \models i$
10. (Cut-state) $E \models i$ and $E, i \models i'$ imply $E \models i'$.
11. (Substitution-state) $E \models i$ implies $E[t/v] \models i[t/v]$.

Definition A.1. *Let $\alpha = k, u_b, u_e$ and $\alpha' = k', u_b', u_e'$ be two views. We say that α is stronger than α' under constraints Ψ, written $\Psi \models \alpha \geq \alpha'$ if the following hold:*

1. $\Psi \models k \succeq k'$
2. $\Psi \models u_b \leq u_b'$
3. $\Psi \models u_e' \leq u_e$

Lemma A.1 (View Subsumption)**.** *Let $M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$ and suppose $\Psi \models \alpha \geq \alpha'$. Then $M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha'} s \circ [u_1, u_2]$ by a derivation of equal or smaller depth.*[4]

Proof. By induction on the given derivation of $M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$, case analyzing the last rule.

Case. $$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1, u_2] \qquad M_2 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1, u_2]}{(\texttt{pf\_conjI}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \wedge s_2 \circ [u_1, u_2]}\wedge\mathrm{I}$$

We have:

[4] The depth of a derivation is defined as the maximum number of rules of Figures 4 and 5 on the unique path from the final sequent to any leaf.

1. $M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha'} s_1 \circ [u_1, u_2]$ (i.h. on 1st premise)
2. $M_2 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha'} s_2 \circ [u_1, u_2]$ (i.h. on 2nd premise)
3. $(\texttt{pf\_conjI}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha'} s_1 \wedge s_2 \circ [u_1, u_2]$ (Rule $\wedge$I on 1 and 2)

Case. Rules ($\wedge$E1)–(@E) from Figure 4 are treated as in the case above. For each of these, we apply the induction hypothesis to any premise that contains α, and then reapply the rule.

Case. $$\frac{\Psi \models u_1' \leq u_1 \qquad \Psi \models u_2 \leq u_2'}{x :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]}\text{hyp}$$

1. $\Psi \models u_1' \leq u_1$ and $\Psi \models u_2 \leq u_2'$ (premises)
2. $x :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1', u_2'] \xrightarrow{\alpha'} s \circ [u_1, u_2]$ (Rule (hyp))

Case. $$\frac{\begin{array}{c}\alpha = k, u_b, u_e \\ \Psi \models u_1' \leq u_1 \qquad \Psi \models u_2 \leq u_2' \qquad \Psi \models u_1' \leq u_b \qquad \Psi \models u_e \leq u_2' \qquad \Psi \models k' \succeq k\end{array}}{x :: \Sigma; \Psi; E; \Gamma, x : k' \ \textsf{claims}\ s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]}\text{claims}$$

Let $\alpha' = k'', u_b'', u_e''$, and $\Psi \models \alpha \geq \alpha'$. We have:

1. $\Psi \models u_b \leq u_b''$ (defn of $\Psi \models \alpha \succeq \alpha'$)
2. $\Psi \models u_1' \leq u_b$ (3rd premise)
3. $\Psi \models u_1' \leq u_b''$ (Assumption (Trans-time) on 1 and 2)
4. $\Psi \models u_e'' \leq u_e$ (defn of $\Psi \models \alpha \succeq \alpha'$)
5. $\Psi \models u_e \leq u_2'$ (4th premise)
6. $\Psi \models u_e'' \leq u_2'$ (Assumption (Trans-time) on 4 and 5)
7. $\Psi \models k \succeq k''$ (defn of $\Psi \models \alpha \succeq \alpha'$)
8. $\Psi \models k' \succeq k$ (5th premise)
9. $\Psi \models k' \succeq k''$ (Assumption (Trans-prin) on 7 and 8)
10. $\Psi \models u_1' \leq u_1$ (1st premise)
11. $\Psi \models u_2 \leq u_2'$ (2nd premise)
12. $x :: \Sigma; \Psi; E; \Gamma, x : k' \ \textsf{claims}\ s \circ [u_1', u_2'] \xrightarrow{\alpha'} s \circ [u_1, u_2]$

(Rule (claims) on 10,11,3,6,9)

Case. $$\frac{M :: \Sigma; \Psi; E; \Gamma | \xrightarrow{k, u_1, u_2} s \circ [u_1, u_2]}{(\texttt{pf\_saysI}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (k\ \textsf{says}\ s) \circ [u_1, u_2]}\textsf{saysI}$$

1. $M :: \Sigma; \Psi; E; \Gamma | \xrightarrow{k, u_1, u_2} s \circ [u_1, u_2]$ (premise)

2. $(\texttt{pf\_saysI}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha'} (k\ \textsf{says}\ s) \circ [u_1, u_2]$ (Rule (saysI))

Case. $$\frac{\begin{array}{c} M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (k\ \textsf{says}\ s) \circ [u_1, u_2] \\ M_2 :: \Sigma; \Psi; E; \Gamma, x : k\ \textsf{claims}\ s \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u_1', u_2'] \end{array}}{(\texttt{pf\_saysE}\ M_1\ ([x]M_2)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']}\textsf{saysE}$$

1. $M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha'} (k\ \textsf{says}\ s) \circ [u_1, u_2]$ (i.h. on 1st premise)
2. $M_2 :: \Sigma; \Psi; E; \Gamma, x : k\ \textsf{claims}\ s \circ [u_1, u_2] \xrightarrow{\alpha'} s' \circ [u_1', u_2']$ (i.h. on 2nd premise)
3. $(\texttt{pf\_saysE}\ M_1\ ([x]M_2)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha'} s' \circ [u_1', u_2']$ (Rule (saysE) on 1 and 2)

Case. $$\frac{E \models i}{(\texttt{pf\_sinjI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} i \circ [u_1, u_2]}\text{interI}$$

1. $E \models i$ (premise)
2. $(\texttt{pf\_sinjI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha'} i \circ [u_1, u_2]$ (Rule (interI) on 1)

Case. $$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} i \circ [u_1, u_2] \qquad M_2 :: \Sigma; \Psi; E, i; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']}{(\texttt{pf\_sinjE}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']}\text{interE}$$

1. $M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha'} i \circ [u_1, u_2]$ (i.h. on 1st premise)
2. $M_2 :: \Sigma; \Psi; E, i; \Gamma \xrightarrow{\alpha'} s' \circ [u_1', u_2']$ (i.h. on 2nd premise)
3. $(\texttt{pf\_sinjE}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha'} s' \circ [u_1', u_2']$ (Rule (interE) on 1 and 2)

Case. Rule (consI): similar to (sinjI)
Case. Rule (consE): similar to (sinjE)

□

Lemma A.2 (Constraint substitution)**.** *Suppose* $\Psi \models c_0$ *and* $M :: \Sigma; \Psi, c_0; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$*. Then,* $M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$ *by a derivation of shorter or equal depth.*

Proof. By induction on the given derivation of $M :: \Sigma; \Psi, c_0; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$, and case analysis of its last rule. The interesting cases are those rules where $\Psi \models \cdot$ is used in one of the premises. In such cases, we appeal to the assumption (Cut-cons). In the remaining rules, we just apply the induction hypothesis to the premises, and reapply the rule to the modified premises. For illustration, we show one example of the latter (rule ($\wedge$I)), and then turn to the more interesting cases.

Case. $$\frac{M_1 :: \Sigma; \Psi, c_0; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1, u_2] \qquad M_2 :: \Sigma; \Psi, c_0; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1, u_2]}{(\texttt{pf\_conjI}\ M_1\ M_2) :: \Sigma; \Psi, c_0; E; \Gamma \xrightarrow{\alpha} s_1 \wedge s_2 \circ [u_1, u_2]}\wedge\text{I}$$

1. $M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1, u_2]$ (i.h. on 1st premise)
2. $M_2 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1, u_2]$ (i.h. on 2nd premise)

3. $(\texttt{pf\_conjI}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \wedge s_2 \circ [u_1, u_2]$ (Rule ($\wedge$I) on 1 and 2)

Case. $$\frac{\begin{array}{c} M_1 :: \Sigma; \Psi, c_0; E; \Gamma \xrightarrow{\alpha} s_1 \supset s_2 \circ [u_1, u_2] \\ M_2 :: \Sigma; \Psi, c_0; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1', u_2'] \qquad \Psi, c_0 \models u_1 \leq u_1' \leq u_1'' \qquad \Psi, c_0 \models u_2'' \leq u_2' \leq u_2 \end{array}}{(\texttt{pf\_impE}\ M_1\ M_2\ u_1'\ u_2') :: \Sigma; \Psi, c_0; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1'', u_2'']}\supset\text{E}$$

1. $M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \supset s_2 \circ [u_1, u_2]$ (i.h. on 1st premise)
2. $M_2 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1', u_2']$ (i.h. on 2nd premise)
3. $\Psi \models c_0$ (Assumption)
4. $\Psi \models u_1 \leq u_1' \leq u_1''$ ((Cut-cons) on 3 and 3rd premise)
5. $\Psi \models u_2'' \leq u_2' \leq u_2$ ((Cut-cons) on 3 and 4th premise)
6. $(\texttt{pf\_impE}\ M_1\ M_2\ u_1'\ u_2') :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1'', u_2'']$ (Rule ($\supset$E) on 1,2,4,5)

Case. $$\frac{\Psi, c_0 \models u_1' \leq u_1 \qquad \Psi, c_0 \models u_2 \leq u_2'}{x :: \Sigma; \Psi, c_0; E; \Gamma, x : s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]}\text{hyp}$$

1. $\Psi \models c_0$ (Assumption)
2. $\Psi \models u_1' \leq u_1$ ((Cut-cons) on 1 and 1st premise)
3. $\Psi \models u_2 \leq u_2'$ ((Cut-cons) on 1 and 2nd premise)
4. $x :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]$ (Rule (hyp) on 2 and 3)

Case. $$\frac{\begin{array}{c} \alpha = k, u_b, u_e \qquad \Psi, c_0 \models u_1' \leq u_1 \\ \Psi, c_0 \models u_2 \leq u_2' \qquad \Psi, c_0 \models u_1' \leq u_b \qquad \Psi, c_0 \models u_e \leq u_2' \qquad \Psi, c_0 \models k' \succeq k \end{array}}{x :: \Sigma; \Psi, c_0; E; \Gamma, x : k' \textsf{ claims } s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]}\text{claims}$$

1. $\Psi \models c_0$ (Assumption)
2. $\Psi \models u_1' \leq u_1$ ((Cut-cons) on 1 and 1st premise)
3. $\Psi \models u_2 \leq u_2'$ ((Cut-cons) on 1 and 2nd premise)
4. $\Psi \models u_1' \leq u_b$ ((Cut-cons) on 1 and 3rd premise)
5. $\Psi \models u_e \leq u_2'$ ((Cut-cons) on 1 and 4th premise)
6. $\Psi \models k' \succeq k$ ((Cut-cons) on 1 and 5th premise)
7. $x :: \Sigma; \Psi; E; \Gamma, x : k' \textsf{ claims } s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]$ (Rule (claims) on 2–6)

Case. $$\frac{\Psi, c_0 \models c}{(\texttt{pf\_cinjI}) :: \Sigma; \Psi, c_0; E; \Gamma \xrightarrow{\alpha} c \circ [u_1, u_2]}\text{consI}$$

1. $\Psi \models c_0$ (Assumption)
2. $\Psi \models c$ ((Cut-cons) on 1 and 1st premise)

3. $(\texttt{pf\_cinjI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} c \circ [u_1, u_2]$ (Rule (consI) on 2)

□

Theorem A.3 (Subsumption; Theorem 3.2). *Suppose the following hold:*

1. $M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$
2. $\Psi \models u_1 \leq u_n$ *and* $\Psi \models u_m \leq u_2$

Then, $M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_n, u_m]$

Proof. By induction on the depth of the given derivation of $M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$, and case analysis of its last rule. Some representative cases are shown below.

Case. $$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1, u_2] \qquad M_2 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1, u_2]}{(\texttt{pf\_conjI}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \wedge s_2 \circ [u_1, u_2]}\wedge\mathrm{I}$$

1. $M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_n, u_m]$ (i.h. on 1st premise)
2. $M_2 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_n, u_m]$ (i.h. on 2nd premise)
3. $(\texttt{pf\_conjI}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \wedge s_2 \circ [u_n, u_m]$ (Rule ($\wedge$I) on 1 and 2)

Case. $$\frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \wedge s_2 \circ [u_1, u_2]}{(\texttt{pf\_conjE1}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1, u_2]}\wedge\mathrm{E1}$$

1. $M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \wedge s_2 \circ [u_n, u_m]$ (i.h. on premise)
2. $(\texttt{pf\_conjE1}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_n, u_m]$ (Rule ($\wedge$E1) on 1)

Case. $$\frac{}{(\texttt{pf\_topI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} \top \circ [u_1, u_2]}\top\mathrm{I}$$

1. $(\texttt{pf\_topI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} \top \circ [u_n, u_m]$ (Rule ($\top$I))

Case. $$\frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} \bot \circ [u_1, u_2]}{(\texttt{pf\_botE}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1', u_2']}\bot\mathrm{E}$$

1. $(\texttt{pf\_botE}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_n, u_m]$ (Rule ($\bot$E) on premise)

Case. $$\frac{M :: \Sigma, v_1{:}\mathsf{time}, v_2{:}\mathsf{time}; \Psi, u_1 \leq v_1, v_2 \leq u_2; E; \Gamma, x : s_1 \circ [v_1, v_2] \xrightarrow{\alpha} s_2 \circ [v_1, v_2]}{(\texttt{pf\_impI}\ ([x][v_1][v_2]M)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \supset s_2 \circ [u_1, u_2]}\supset\mathrm{I}$$

1. $\Psi \models u_1 \leq u_n$ (Assumption)
2. $\Psi, u_n \leq v_1 \models u_1 \leq u_n$ ((Weakening-cons) on 1)
3. $\Psi, u_n \leq v_1 \models u_n \leq v_1$ (Refl-time)
4. $\Psi, u_n \leq v_1 \models u_1 \leq v_1$ ((Trans-time) on 2,3)
5. $M :: \Sigma, v_1{:}\mathsf{time}, v_2{:}\mathsf{time}; \Psi, u_1 \leq v_1, v_2 \leq u_2; E; \Gamma, x : s_1 \circ [v_1, v_2] \xrightarrow{\alpha} s_2 \circ [v_1, v_2]$

(premise)

6. $M :: \Sigma, v_1{:}\mathsf{time}, v_2{:}\mathsf{time}; \Psi, u_n \leq v_1, v_2 \leq u_2; E; \Gamma, x : s_1 \circ [v_1, v_2] \xrightarrow{\alpha} s_2 \circ [v_1, v_2]$

(Lemma A.2 on 4,5)

7. $\Psi, v_2 \leq u_m \models v_2 \leq u_2$ (Similar to 4)

8. $M :: \Sigma, v_1{:}\mathsf{time}, v_2{:}\mathsf{time}; \Psi, u_n \leq v_1, v_2 \leq u_m; E; \Gamma, x : s_1 \circ [v_1, v_2] \xrightarrow{\alpha} s_2 \circ [v_1, v_2]$

(Lemma A.2 on 7,6)

9. $(\mathtt{pf\_impI}\ ([x][v_1][v_2]M)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \supset s_2 \circ [u_n, u_m]$ (Rule ($\supset$I) on 8)

Case.
$$\frac{\begin{array}{c} M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \supset s_2 \circ [u_1, u_2] \\ M_2 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1', u_2'] \qquad \Psi \models u_1 \leq u_1' \leq u_1'' \qquad \Psi \models u_2'' \leq u_2' \leq u_2 \end{array}}{(\mathtt{pf\_impE}\ M_1\ M_2\ u_1'\ u_2') :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1'', u_2'']} \supset\text{E}$$

1. $\Psi \models u_1'' \leq u_n$ (Assumption)

2. $\Psi \models u_1 \leq u_1' \leq u_n$ ((Trans-time) on 1 and premise 3)

3. $\Psi \models u_m \leq u_2''$ (Assumption)

4. $\Psi \models u_m \leq u_2' \leq u_2$ ((Trans-time) on 3 and premise 4)

5. $(\mathtt{pf\_impE}\ M_1\ M_2\ u_1'\ u_2') :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s_2 \circ [u_n, u_m]$

(Rule ($\supset$E) on 1st,2nd premise and 2,4)

Case.
$$\frac{M :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]}{(\mathtt{pf\_atI}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (s \mathbin{@} [u_1, u_2]) \circ [u_1', u_2']} @\text{I}$$

1. $(\mathtt{pf\_atI}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (s \mathbin{@} [u_1, u_2]) \circ [u_n, u_m]$ (Rule (@I) on premise)

Case.
$$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (s \mathbin{@} [u_1, u_2]) \circ [u_1', u_2'] \qquad M_2 :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u_1'', u_2'']}{(\mathtt{pf\_atE}\ M_1\ ([x]M_2)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_1'', u_2'']} @\text{E}$$

1. $M_2 :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u_n, u_m]$ (i.h. on 2nd premise)

2. $(\mathtt{pf\_atE}\ M_1\ ([x]M_2)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_n, u_m]$

(Rule (@E) on 1st premise and 1)

Case.
$$\frac{\Psi \models u_1' \leq u_1 \qquad \Psi \models u_2 \leq u_2'}{x :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]} \text{hyp}$$

1. $\Psi \models u_1 \leq u_n$ (Assumption)

2. $\Psi \models u_1' \leq u_n$ ((Trans-time) on 1st premise and 1)

3. $\Psi \models u_m \leq u_2$ (Assumption)

4. $\Psi \models u_m \leq u_2'$ ((Trans-time) on 2nd premise and 3)

5. $x :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_n, u_m]$ (Rule (hyp) on 2,4)

Case. $$\frac{\begin{array}{c}\alpha = k, u_b, u_e \\ \Psi \models u_1' \leq u_1 \qquad \Psi \models u_2 \leq u_2' \qquad \Psi \models u_1' \leq u_b \qquad \Psi \models u_e \leq u_2' \qquad \Psi \models k' \succeq k\end{array}}{x :: \Sigma; \Psi; E; \Gamma, x : k' \ \mathsf{claims}\ s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_1, u_2]}\text{claims}$$

1. $\Psi \models u_1 \leq u_n$ (Assumption)

2. $\Psi \models u_1' \leq u_n$ ((Trans-time) on 1st premise and 1)

3. $\Psi \models u_m \leq u_2$ (Assumption)

4. $\Psi \models u_m \leq u_2'$ ((Trans-time) on 2nd premise and 3)

5. $x :: \Sigma; \Psi; E; \Gamma, x : k' \ \mathsf{claims}\ s \circ [u_1', u_2'] \xrightarrow{\alpha} s \circ [u_n, u_m]$

(Rule (claims) on 2,4 and 3rd–5th premises)

Case. $$\frac{M :: \Sigma; \Psi; E; \Gamma | \xrightarrow{k, u_1, u_2} s \circ [u_1, u_2]}{(\mathtt{pf\_saysI}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (k\ \mathsf{says}\ s) \circ [u_1, u_2]}\mathsf{saysI}$$

1. $\Psi \models k \succeq k$ (Refl-prin)

2. $\Psi \models u_1 \leq u_n$ (Assumption)

3. $\Psi \models u_m \leq u_2$ (Assumption)

4. $\Psi \models (k, u_1, u_2) \geq (k, u_n, u_m)$ (Definition on 1–3)

5. $M :: \Sigma; \Psi; E; \Gamma | \xrightarrow{k, u_n, u_m} s \circ [u_1, u_2]$ (Lemma A.1 on 4 and premise)

6. $M :: \Sigma; \Psi; E; \Gamma | \xrightarrow{k, u_n, u_m} s \circ [u_n, u_m]$ (i.h. on 5)

7. $(\mathtt{pf\_saysI}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (k\ \mathsf{says}\ s) \circ [u_n, u_m]$ (Rule (saysI) on 6)

Case. $$\frac{\begin{array}{c}M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (k\ \mathsf{says}\ s) \circ [u_1, u_2] \\ M_2 :: \Sigma; \Psi; E; \Gamma, x : k\ \mathsf{claims}\ s \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u_1', u_2']\end{array}}{(\mathtt{pf\_saysE}\ M_1\ ([x]M_2)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']}\mathsf{saysE}$$

1. $M_2 :: \Sigma; \Psi; E; \Gamma, x : k\ \mathsf{claims}\ s \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u_n, u_m]$ (i.h. on 2nd premise)

2. $(\mathtt{pf\_saysE}\ M_1\ ([x]M_2)) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_n, u_m]$

(Rule (saysE) on 1st premise and 1)

Case. $$\frac{E \models i}{(\mathtt{pf\_sinjI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} i \circ [u_1, u_2]}\text{interI}$$

1. $(\mathtt{pf\_sinjI}) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} i \circ [u_n, u_m]$ (Rule (interI) on premise)

Case. $$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} i \circ [u_1, u_2] \qquad M_2 :: \Sigma; \Psi; E, i; \Gamma \xrightarrow{\alpha} s' \circ [u'_1, u'_2]}{(\texttt{pf\_sinjE}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u'_1, u'_2]}\text{interE}$$

1. $M_2 :: \Sigma; \Psi; E, i; \Gamma \xrightarrow{\alpha} s' \circ [u_n, u_m]$ (i.h. on 2nd premise)
2. $(\texttt{pf\_sinjE}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u_n, u_m]$

   (Rule (interE) on 1st premise and 1)

□

Substitution. $M[M'/x]$ denotes the capture avoiding substitution of proof term M' for proof variable x in proof term M. The substitution is defined by induction on M. Since it follows a standard template, we show below only some selected clauses of the definition. $x \notin M$ means that x does not occur free in M.

$$\begin{array}{rcll}
x[M'/x] & = & M' & \\
y[M'/x] & = & y & (x \neq y) \\
(\texttt{pf\_conjE1}\ M)[M'/x] & = & \texttt{pf\_conjE1}\ (M[M'/x]) & \\
(\texttt{pf\_impI}\ ([y][v_1][v_2]M))[M'/x] & = & \texttt{pf\_impI}\ ([y][v_1][v_2]M[M'/x]) & (y \neq x \text{ and } y \notin M)
\end{array}$$

Theorem A.4 (Substitution; Theorem 3.1)**.** *The following hold:*

1. $M' :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$ *and* $M :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u'_1, u'_2]$ *imply that* $M[M'/x] :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u'_1, u'_2]$
2. $M' :: \Sigma; \Psi; E; \Gamma| \xrightarrow{k,u_1,u_2} s \circ [u_1, u_2]$ *and* $M :: \Sigma; \Psi; E; \Gamma, x : k\ \mathsf{claims}\ s \circ [u_1, u_2] \xrightarrow{\alpha} s' \circ [u'_1, u'_2]$ *imply that* $M[M'/x] :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s' \circ [u'_1, u'_2]$

Proof. By simultaneous induction on given derivations containing M, and case analysis of the last rule in the derivations. We show some interesting cases for both statements above. The other cases are straightforward: the induction hypothesis is applied to the premises and the rule is applied again.

Proof of (1)

Case. $$\frac{\Psi \models u_1 \leq u'_1 \qquad \Psi \models u'_2 \leq u_2}{x :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \xrightarrow{\alpha} s \circ [u'_1, u'_2]}\text{hyp}$$

1. $M' :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$ (Assumption)
2. $\Psi \models u_1 \leq u'_1$ and $\Psi \models u'_2 \leq u_2$ (Premises)
3. $M' :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u'_1, u'_2]$ (Lemma A.3 on 2 and 1)
4. $M' = x[M'/x]$ (Definition)

Case. $$\frac{\Psi \models u''_1 \leq u'_1 \qquad \Psi \models u'_2 \leq u''_2}{y :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2], y : s' \circ [u''_1, u''_2] \xrightarrow{\alpha} s' \circ [u'_1, u'_2]}\text{hyp}$$

1. $y :: \Sigma; \Psi; E; \Gamma, y : s' \circ [u_1'', u_2''] \xrightarrow{\alpha} s' \circ [u_1', u_2']$ (Rule (hyp) on premises)
2. $y[M'/x] = y$ (Definition)

$$\textbf{Case.}\quad \dfrac{\overset{\alpha = k, u_b, u_e}{\Psi \models u_1' \le u_1'' \qquad \Psi \models u_2'' \le u_2' \qquad \Psi \models u_1' \le u_b \qquad \Psi \models u_e \le u_2' \qquad \Psi \models k' \succeq k}}{y :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2], y : k' \;\mathsf{claims}\; r \circ [u_1', u_2'] \xrightarrow{\alpha} r \circ [u_1'', u_2'']}\text{claims}$$

1. $y :: \Sigma; \Psi; E; \Gamma, y : k' \;\mathsf{claims}\; r \circ [u_1', u_2'] \xrightarrow{\alpha} r \circ [u_1'', u_2'']$ (Rule (claims) on premises)
2. $y[M'/x] = y$ (Definition)

$$\textbf{Case.}\quad \dfrac{M :: \Sigma; \Psi; E; (\Gamma, x : s \circ [u_1, u_2])| \xrightarrow{k, u_1', u_2'} r \circ [u_1', u_2']}{(\texttt{pf\_saysI}\ M) :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \xrightarrow{\alpha} (k \;\mathsf{says}\; r) \circ [u_1', u_2']}\mathsf{saysI}$$

1. $(\Gamma, x : s \circ [u_1, u_2])| = \Gamma|$ (Definition)
2. $M :: \Sigma; \Psi; E; \Gamma| \xrightarrow{k, u_1', u_2'} r \circ [u_1', u_2']$ (Premise and 1)
3. $(\texttt{pf\_saysI}\ M) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (k \;\mathsf{says}\; r) \circ [u_1', u_2']$ (Rule (saysI) on 2)
4. $(\texttt{pf\_saysI}\ M)[M'/x] = (\texttt{pf\_saysI}\ M)$ ($x \notin M$ from 2)

$$\textbf{Case.}\quad \dfrac{\begin{array}{c} M_1 :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \xrightarrow{\alpha} (k \;\mathsf{says}\; r) \circ [u_1', u_2'] \\ M_2 :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2], y : k \;\mathsf{claims}\; r \circ [u_1', u_2'] \xrightarrow{\alpha} r' \circ [u_1'', u_2''] \end{array}}{(\texttt{pf\_saysE}\ M_1\ ([y]M_2)) :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \xrightarrow{\alpha} r' \circ [u_1'', u_2'']}\mathsf{saysE}$$

1. $M_1[M'/x] :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (k \;\mathsf{says}\; r) \circ [u_1', u_2']$ (i.h. on 1st premise)
2. $M_2[M'/x] :: \Sigma; \Psi; E; \Gamma, y : k \;\mathsf{claims}\; r \circ [u_1', u_2'] \xrightarrow{\alpha} r' \circ [u_1'', u_2'']$

   (i.h. on 2nd premise)
3. $(\texttt{pf\_saysE}\ (M_1[M'/x])\ ([y]M_2[M'/x])) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} r' \circ [u_1'', u_2'']$

   (Rule (saysE) on 1 and 2)
4. $(\texttt{pf\_saysE}\, M_1\ ([y]M_2))[M'/x] = \texttt{pf\_saysE}\ (M_1[M'/x])\ ([y]M_2[M'/x])$ (Definition)

Proof of (2)

$$\textbf{Case.}\quad \dfrac{\overset{\alpha = k', u_b, u_e}{\Psi \models u_1 \le u_1' \qquad \Psi \models u_2' \le u_2 \qquad \Psi \models u_1 \le u_b \qquad \Psi \models u_e \le u_2 \qquad \Psi \models k \succeq k'}}{x :: \Sigma; \Psi; E; \Gamma, x : k \;\mathsf{claims}\; s \circ [u_1, u_2] \xrightarrow{\alpha} s \circ [u_1', u_2']}\text{claims}$$

1. $M' :: \Sigma; \Psi; E; \Gamma| \xrightarrow{k, u_1, u_2} s \circ [u_1, u_2]$ (Assumption)
2. $M' :: \Sigma; \Psi; E; \Gamma \xrightarrow{k, u_1, u_2} s \circ [u_1, u_2]$ (Weakening on 1)
3. $\Psi \models (k, u_1, u_2) \geq \alpha$ (Premises 3–5)

4. $M' :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$ (Lemma A.1 on 2 and 3)

5. $\Psi \models u_1 \leq u_1'$ and $\Psi \models u_2' \leq u_2$ (Premises 1 and 2)

6. $M' :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} s \circ [u_1', u_2']$ (Lemma A.3 on 4 and 5)

7. $x[M'/x] = M'$ (Definition)

Case. $$\frac{M :: \Sigma; \Psi; E; \Gamma|, x : k \mathsf{\ claims\ } s \circ [u_1, u_2] \xrightarrow{k', u_1', u_2'} r \circ [u_1', u_2']}{(\texttt{pf\_saysI}\ M) :: \Sigma; \Psi; E; \Gamma, x : k \mathsf{\ claims\ } s \circ [u_1, u_2] \xrightarrow{\alpha} (k' \mathsf{\ says\ } r) \circ [u_1', u_2']}\mathsf{saysI}$$

1. $(\Gamma|)| = \Gamma|$ (Definition)

2. $M' :: \Sigma; \Psi; E; (\Gamma|)| \xrightarrow{k, u_1, u_2} s \circ [u_1, u_2]$ (Assumption and 1)

3. $M[M'/x] :: \Sigma; \Psi; E; \Gamma| \xrightarrow{k', u_1', u_2'} r \circ [u_1', u_2']$ (i.h. on premise and 2)

4. $(\texttt{pf\_saysI}\ (M[M'/x])) :: \Sigma; \Psi; E; \Gamma \xrightarrow{\alpha} (k' \mathsf{\ says\ } r) \circ [u_1', u_2']$ (Rule (saysI) on 3)

5. $(\texttt{pf\_saysI}\ M)[M'/x] = \texttt{pf\_saysI}\ (M[M'/x])$ (Definition)

□

B Proof Verification and Generation of Procaps

This appendix formalizes the verifier for BL's proofs that PCFS uses. Let C denote a judgment of the form $\Psi \models c$ (the judgment may or may not hold) and $\vec{C}$ denote a set of such judgments. $\models \vec{C}$ means that for each $(\Psi \models c) \in \vec{C}$, it is the case that $\Psi \models c$ holds. Further let $\vec{i}$ denote a set of interpreted predicates. $E \models \vec{i}$ means that for each $i \in \vec{i}$, $E \models i$. In the following, we first discuss a general proof verification procedure for BL's proofs and then show how it is specialized to PCFS.

B.1 A General Proof Verifier for BL

We construct a bidirectional proof verifier for BL formalized using two verification judgments: $M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}$ (checking judgment), and $M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}$ (synthesis judgment). The intent of both judgments is that if $\models \vec{C}$ and $E' \models \vec{i}$, then $M :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$ in BL's natural deduction system (Figures 4 and 5). However, $s \circ [u_1, u_2]$ is an *input* to the verification procedure in the checking judgment, and an *output* of the procedure in the synthesis judgment. $M, \Sigma, \Psi, E, \Gamma, \alpha$ are inputs in both cases, whereas $\vec{C}, \vec{i}$ are always outputs.

The output $\vec{C}$ is a non-deterministically chosen subset of the constraint judgments on which the correctness of the proof depends. The non-determinism is deliberate; in Section B.2 we show how the non-determinism is resolved when the verifier is used in PCFS, and use theorems from this section in that specific context. Figures 6 and 7 list the rules for the checking judgment, whereas Figure 8 lists the rules for the synthesis judgment. The notation $\vec{C_1} \bowtie \vec{C_2} = \vec{C}$ means that $\vec{C_1}$ and $\vec{C_2}$ are a non-deterministically chosen disjoint partition of $\vec{C}$. All rules are implemented backwards: the proof term

$$\dfrac{M_1 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_1 \circ [u_1, u_2] \searrow \vec{C_1}; \vec{i_1} \qquad M_2 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_2 \circ [u_1, u_2] \searrow \vec{C_2}; \vec{i_2}}{(\mathtt{pf\_conjI}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_1 \wedge s_2 \circ [u_1, u_2] \searrow \vec{C_1}, \vec{C_2}; \vec{i_1}, \vec{i_2}}\wedge\mathrm{I}$$

$$\dfrac{M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_1 \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}{(\mathtt{pf\_disjI1}\ M) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_1 \vee s_2 \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}\vee\mathrm{I1}$$

$$\dfrac{M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_2 \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}{(\mathtt{pf\_disjI2}\ M) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_1 \vee s_2 \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}\vee\mathrm{I2}$$

$$\dfrac{\begin{array}{c} M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s_1 \vee s_2 \circ [u_1, u_2] \searrow \vec{C_1}; \vec{i_1} \qquad M_1 :: \Sigma; \Psi; E; \Gamma, x : s_1 \circ [u_1, u_2] \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_2}; \vec{i_2} \\ M_2 :: \Sigma; \Psi; E; \Gamma, y : s_2 \circ [u_1, u_2] \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_3}; \vec{i_3} \end{array}}{(\mathtt{pf\_disjE}\ M\ ([x]M_1)\ ([y]M_2)) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_1}, \vec{C_2}, \vec{C_3}; \vec{i_1}, \vec{i_2}, \vec{i_3}}\vee\mathrm{E}$$

$$\dfrac{}{(\mathtt{pf\_topI}) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} \top \circ [u_1, u_2] \searrow \cdot; \cdot}\top\mathrm{I} \qquad \dfrac{M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} \bot \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}{(\mathtt{pf\_botE}\ M) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s \circ [u_1', u_2'] \searrow \vec{C}; \vec{i}}\bot\mathrm{E}$$

$$\dfrac{M :: \Sigma, v_1{:}\mathsf{time}, v_2{:}\mathsf{time}; \Psi, u_1 \leq v_1, v_2 \leq u_2; E; \Gamma, x : s_1 \circ [v_1, v_2] \overset{\alpha}{\Longrightarrow} s_2 \circ [v_1, v_2] \searrow \vec{C}; \vec{i}}{(\mathtt{pf\_impI}\ ([x][v_1][v_2]M)) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_1 \supset s_2 \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}\supset\mathrm{I}$$

$$\dfrac{M :: \Sigma, v{:}\sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}{(\mathtt{pf\_forallI}\ ([v]M)) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} \forall v{:}\sigma.s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}\forall\mathrm{I}$$

$$\dfrac{M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s[t/v] \circ [u_1, u_2] \searrow \vec{C}; \vec{i} \qquad \Sigma \vdash t : \sigma}{(\mathtt{pf\_existsI}\ t\ M) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} \exists v{:}\sigma.s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}\exists\mathrm{I}$$

$$\dfrac{M_1 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} \exists v{:}\sigma.s \circ [u_1, u_2] \searrow \vec{C_1}; \vec{i_1} \qquad M_2 :: \Sigma, v{:}\sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_2}; \vec{i_2}}{(\mathtt{pf\_existsE}\ M_1\ ([v][x]M_2)) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_1}, \vec{C_2}; \vec{i_1}, \vec{i_2}}\exists\mathrm{E}$$

Figure 6: BL: Proof verification checking rules, Part 1

to be verified is matched with the conclusion of a rule, and the premises established recursively. Observe that, with the exception of the rule (CS) in Figure 7, only one other rule will apply to any given proof term and hypotheses.

Since bidirectional proof checking is standard, we describe only some of the rules briefly. Checking rules for proof terms that introduce connectives follow a similar template. We explain only one representative case here: ($\wedge$I) from Figure 6. This rule states that in order check that `pf_conjI` M_1 M_2 establishes the judgment $s_1 \wedge s_2 \circ [u_1, u_2]$ (conclusion of the rule), we must check that M_1 establishes $s_1 \circ [u_1, u_2]$ (first premise) and that M_2 establishes $s_2 \circ [u_1, u_2]$ (second premise). The outputs of the premises $\vec{C_1}; \vec{i_1}$ and $\vec{C_2}; \vec{i_2}$ are combined to form the output of the whole verification: $\vec{C_1}, \vec{C_2}; \vec{i_1}, \vec{i_2}$. Checking rules for proof terms that eliminate a connective are more interesting. In these cases the principal judgment is synthesized, not checked. For example, in the rule ($\vee$E) from Figure 6, the first premise synthesizes the judgment $s_1 \vee s_2 \circ [u_1, u_2]$ from the proof term M. The obtained formulas s_1 and s_2 are then used to check the two proof terms M_1 and M_2.

The rule (interI) from Figure 5 corresponds to two checking rules in the verifier, named (interI1) and (interI2) in Figure 7. If the interpreted predicate i checks in E,

$$\frac{M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}{(\texttt{pf\_atI}\ M) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} (s \ @\ [u_1, u_2]) \circ [u_1', u_2'] \searrow \vec{C}; \vec{i}}\text{@I}$$

$$\frac{\begin{array}{c} M_1 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} (s \ @\ [u_1, u_2]) \circ [u_1', u_2'] \searrow \vec{C_1}; \vec{i_1} \\ M_2 :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \overset{\alpha}{\Longrightarrow} s' \circ [u_1'', u_2''] \searrow \vec{C_2}; \vec{i_2} \end{array}}{(\texttt{pf\_atE}\ M_1\ ([x]M_2)) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u_1'', u_2''] \searrow \vec{C_1}, \vec{C_2}; \vec{i_1}, \vec{i_2}}\text{@E}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma| \overset{k,u_1,u_2}{\Longrightarrow} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}{(\texttt{pf\_saysI}\ M) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} (k \textsf{ says } s) \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}\textsf{saysI}$$

$$\frac{\begin{array}{c} M_1 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} (k \textsf{ says } s) \circ [u_1, u_2] \searrow \vec{C_1}; \vec{i_1} \\ M_2 :: \Sigma; \Psi; E; \Gamma, x : k \textsf{ claims } s \circ [u_1, u_2] \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_2}; \vec{i_2} \end{array}}{(\texttt{pf\_saysE}\ M_1\ ([x]M_2)) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_1}, \vec{C_2}; \vec{i_1}, \vec{i_2}}\textsf{saysE}$$

$$\frac{E \models i}{(\texttt{pf\_sinjI}) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} i \circ [u_1, u_2] \searrow \cdot; \cdot}\text{interI1} \qquad \frac{}{(\texttt{pf\_sinjI}) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} i \circ [u_1, u_2] \searrow \cdot; i}\text{interI2}$$

$$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} i \circ [u_1, u_2] \searrow \vec{C_1}; \vec{i_1} \qquad M_2 :: \Sigma; \Psi; E, i; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_2}; \vec{i_2}}{(\texttt{pf\_sinjE}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_1}, \vec{C_2}; \vec{i_1}, \vec{i_2}}\text{interE}$$

$$\frac{\vec{C_1} \bowtie \vec{C_2} = (\Psi \models c) \qquad \models \vec{C_1}}{(\texttt{pf\_cinjI}) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} c \circ [u_1, u_2] \searrow \vec{C_2}; \cdot}\text{consI}$$

$$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} c \circ [u_1, u_2] \searrow \vec{C_1}; \vec{i_1} \qquad M_2 :: \Sigma; \Psi, c; E; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_2}; \vec{i_2}}{(\texttt{pf\_cinjE}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_1}, \vec{C_2}; \vec{i_1}, \vec{i_2}}\text{consE}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i} \qquad C_1 \bowtie C_2 = (\Psi \models u_1 \leq u_1'), (\Psi \models u_2' \leq u_2) \qquad \models \vec{C_1}}{M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s \circ [u_1', u_2'] \searrow \vec{C}, \vec{C_2}; \vec{i}}\text{CS}$$

Figure 7: BL: Proof verification checking rules, Part 2

(interI1) is used and the output is empty. Otherwise, i is written to the output (rule (interI2)). When the verifier is used in PCFS, E is generally empty, so in practice, i always gets written to the output (and then to the procap).

Synthesis rules (Figure 8) apply to some proof terms that eliminate connectives and to hypotheses (rules (hyp) and (claims)). The only non-trivial rule here is ($\supset$E). It states that in order to synthesize the judgment proved by a proof term `pf_impE` M_1 M_2 u_1' u_2', we should first synthesize the judgment $s_1 \supset s_2 \circ [u_1, u_2]$ proved by M_1 (first premise). Next, we should *check* that M_2 proves $s_1 \circ [u_1', u_2']$ (second premise) and that $[u_1', u_2']$ is a subset of $[u_1, u_2]$ (constraints $(\Psi \models u_1 \leq u_1')$ and $(\Psi \models u_2' \leq u_2)$). If all these hold, then `pf_impE` M_1 M_2 u_1' u_2' proves the judgment $s_2 \circ [u_1', u_2']$.

In both the checking and synthesis rules, if a set of constraint judgments needs to be checked in a rule, then this set is split non-deterministically into two sets, one of which is checked immediately, and the other of which is written to the output. This happens in rules (consI), (CS), (claims) and ($\supset$E). It is expected that the output judgments will be checked later (e.g., in PCFS, the output judgments are written in a procap and checked whenever the procap is checked).

The rule (CS) in Figure 7 allows the verifier to move from a checking judgment to a synthesis judgment without changing the proof term. This shift can be, and must

$$\frac{}{x :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \overset{\alpha}{\Longleftarrow} s \circ [u_1, u_2] \searrow \cdot; \cdot}\,\text{hyp}$$

$$\frac{\alpha = k, u_b, u_e \qquad \vec{C_1} \bowtie \vec{C_2} = (\Psi \models u_1 \leq u_b), (\Psi \models u_e \leq u_2), (\Psi \models k' \succeq k) \qquad \models \vec{C_1}}{x :: \Sigma; \Psi; E; \Gamma, x : k' \;\textsf{claims}\; s \circ [u_1, u_2] \overset{\alpha}{\Longleftarrow} s \circ [u_1, u_2] \searrow \vec{C_2}; \cdot}\,\text{claims}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s_1 \wedge s_2 \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}{(\texttt{pf\_conjE1}\ M) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s_1 \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}\,\wedge\text{E1}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s_1 \wedge s_2 \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}{(\texttt{pf\_conjE2}\ M) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s_2 \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}\,\wedge\text{E2}$$

$$\frac{\begin{array}{c} M_1 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s_1 \supset s_2 \circ [u_1, u_2] \searrow \vec{C_1}; \vec{i_1} \\ M_2 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_1 \circ [u_1', u_2'] \searrow \vec{C_2}; \vec{i_2} \qquad \vec{C_3} \bowtie \vec{C_4} = (\Psi \models u_1 \leq u_1'), (\Psi \models u_2' \leq u_2) \qquad \models \vec{C_3} \end{array}}{(\texttt{pf\_impE}\ M_1\ M_2\ u_1'\ u_2') :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s_2 \circ [u_1', u_2'] \searrow \vec{C_1}, \vec{C_2}, \vec{C_4}; \vec{i_1}, \vec{i_2}}\,\supset\text{E}$$

$$\frac{M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} \forall v{:}\sigma.s \circ [u_1, u_2] \searrow \vec{C}; \vec{i} \qquad \Sigma \vdash t : \sigma}{(\texttt{pf\_forallE}\ M\ t) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s[t/v] \circ [u_1, u_2] \searrow \vec{C}; \vec{i}}\,\forall\text{E}$$

Figure 8: BL: Proof verification synthesis rules

be, applied when the proof term M matches the proof term in the conclusion of some rule in Figure 8. Generally, bidirectional proof verifiers also allow a shift in the other direction – from synthesis to checking – via an explicit proof term constructor. In our implementation we do not allow this coercion. The consequence of this restriction is that our verifier can only check *normal proof terms* (a proof term is normal if it does not contain any elimination constructor immediately outside an introduction constructor). While it is easy to avoid this restriction by allowing the explicit coercion, we do not do so here since our proof search tool constructs normal proofs only. Further, normal proofs are complete: any hypothetical judgment that has a proof also has a normal proof. Proof of the latter result is based on a sequent calculus presentation of BL, which is beyond the scope of this paper.

Another important point concerns the scope of parameters in the checking rules ($\forall$I) and ($\exists$E) of Figure 6. In each case, the premise introduces a new parameter $v{:}\sigma$, which may appear free in the output – $\vec{C}; \vec{i}$ for ($\forall$I) and $\vec{C_1}, \vec{C_2}; \vec{i_1}, \vec{i_2}$ for ($\exists$E). As mentioned in Appendix A, decision procedures for solving constraints and interpreted predicates treat such parameters universally, e.g., in the case of ($\forall$I), if $\vec{C}$ is found to hold then $\vec{C}[t/v]$ would also hold for every ground term t. To ensure that parameters introduced in different branches of the verification do not conflict in the outputs, we also assume implicitly that all parameters introduced during verification are unique.

The following lemma establishes that this verification procedure is sound, i.e., any proof it verifies is valid in BL's natural deduction system subject to some conditions.

Lemma B.1 (Verification Soundness)**.** *Suppose that* $\models \vec{C}$ *and* $E' \models \vec{i}$*. Then*

1. $M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}$ *implies* $M :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}$

2. $M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}$ *implies* $M :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}$

Proof. The proof follows by a simultaneous induction on the given derivations. We show some representative cases below.

Case. $$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_1 \circ [u_1, u_2] \searrow \vec{C_1}; \vec{i_1} \qquad M_2 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_2 \circ [u_1, u_2] \searrow \vec{C_2}; \vec{i_2}}{(\texttt{pf\_conjI}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_1 \wedge s_2 \circ [u_1, u_2] \searrow \vec{C_1}, \vec{C_2}; \vec{i_1}, \vec{i_2}} \wedge\mathrm{I}$$

1. $\models \vec{C_1}, \vec{C_2}$ (Assumption)
2. $\models \vec{C_1}$ and $\models \vec{C_2}$ (Definition)
3. $E' \models \vec{i_1}, \vec{i_2}$ (Assumption)
4. $E' \models \vec{i_1}$ and $E' \models \vec{i_2}$ (Definition)
5. $M_1 :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1, u_2]$ (i.h. on 1st premise, 2, 4)
6. $M_2 :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1, u_2]$ (i.h. on 2nd premise, 2, 4)
7. $(\texttt{pf\_conjI}\ M_1\ M_2) :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s_1 \wedge s_2 \circ [u_1, u_2]$ (Rule ($\wedge$I) on 5,6)

Case. $$\frac{E \models i}{(\texttt{pf\_sinjI}) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} i \circ [u_1, u_2] \searrow \cdot; \cdot} \mathrm{interI1}$$

1. $E \models i$ (premise)
2. $E, E' \models i$ (Weakening-state)
3. $(\texttt{pf\_sinjI}) :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} i \circ [u_1, u_2]$ (Rule (interI) on 2)

Case. $$\frac{}{(\texttt{pf\_sinjI}) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} i \circ [u_1, u_2] \searrow \cdot; i} \mathrm{interI2}$$

1. $E' \models i$ (Assumption)
2. $E, E' \models i$ (Weakening-state)
3. $(\texttt{pf\_sinjI}) :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} i \circ [u_1, u_2]$ (Rule (interI) on 2)

Case. $$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} i \circ [u_1, u_2] \searrow \vec{C_1}; \vec{i_1} \qquad M_2 :: \Sigma; \Psi; E, i; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_2}; \vec{i_2}}{(\texttt{pf\_sinjE}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u_1', u_2'] \searrow \vec{C_1}, \vec{C_2}; \vec{i_1}, \vec{i_2}} \mathrm{interE}$$

1. $\models \vec{C_1}, \vec{C_2}$ (Assumption)
2. $E' \models \vec{i_1}, \vec{i_2}$ (Assumption)
3. $M_1 :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} i \circ [u_1, u_2]$ (i.h. on 1st premise,1,2)
4. $M_2 :: \Sigma; \Psi; E, i, E'; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']$ (i.h. on 2nd premise,1,2)
5. $(\texttt{pf\_sinjE}\ M_1\ M_2) :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s' \circ [u_1', u_2']$ (Rule (interE) on 3,4)

Case. $$\frac{\vec{C_1} \bowtie \vec{C_2} = (\Psi \models c) \qquad \models \vec{C_1}}{(\texttt{pf\_cinjI}) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} c \circ [u_1, u_2] \searrow \vec{C_2}; \cdot} \mathrm{consI}$$

1. $\models \vec{C}_1$ (2nd premise)
2. $\models \vec{C}_2$ (Assumption)
3. $\vec{C}_1 \bowtie \vec{C}_2 = (\Psi \models c)$ (1st premise)
4. $\Psi \models c$ (1,2,3)
5. $(\texttt{pf\_cinjI}) :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} c \circ [u_1, u_2]$ (Rule (consI) on 4)

Case.
$$\frac{M_1 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} c \circ [u_1, u_2] \searrow \vec{C}_1; \vec{i}_1 \qquad M_2 :: \Sigma; \Psi, c; E; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u'_1, u'_2] \searrow \vec{C}_2; \vec{i}_2}{(\texttt{pf\_cinjE}\ M_1\ M_2) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s' \circ [u'_1, u'_2] \searrow \vec{C}_1, \vec{C}_2; \vec{i}_1, \vec{i}_2}\,\text{consE}$$

1. $\models \vec{C}_1, \vec{C}_2$ (Assumption)
2. $E' \models \vec{i}_1, \vec{i}_2$ (Assumption)
3. $M_1 :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} c \circ [u_1, u_2]$ (i.h. on 1st premise,1,2)
4. $M_2 :: \Sigma; \Psi, c; E, E'; \Gamma \xrightarrow{\alpha} s' \circ [u'_1, u'_2]$ (i.h. on 2nd premise,1,2)
5. $(\texttt{pf\_cinjE}\ M_1\ M_2) :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s' \circ [u'_1, u'_2]$ (Rule (consE) on 3,4)

Case.
$$\frac{\begin{array}{c} M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i} \\ C_1 \bowtie C_2 = (\Psi \models u_1 \leq u'_1), (\Psi \models u'_2 \leq u_2) \qquad \models \vec{C}_1 \end{array}}{M :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s \circ [u'_1, u'_2] \searrow \vec{C}, \vec{C}_2; \vec{i}}\,\text{CS}$$

1. $\models \vec{C}, \vec{C}_2$ (Assumption)
2. $E' \models \vec{i}$ (Assumption)
3. $M :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s \circ [u_1, u_2]$ (i.h. on 1st premise,1,2)
4. $\models \vec{C}_1$ (3rd premise)
5. $C_1 \bowtie C_2 = (\Psi \models u_1 \leq u'_1), (\Psi \models u'_2 \leq u_2)$ (2nd premise)
6. $\Psi \models u_1 \leq u'_1$ and $\Psi \models u'_2 \leq u_2$ (1,4,5)
7. $M :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s \circ [u'_1, u'_2]$ (Theorem A.3 on 3,6)

Case.
$$\frac{}{x :: \Sigma; \Psi; E; \Gamma, x : s \circ [u_1, u_2] \overset{\alpha}{\Longleftarrow} s \circ [u_1, u_2] \searrow \cdot; \cdot}\,\text{hyp}$$

1. $\Psi \models u_1 \leq u_1$ and $\Psi \models u_2 \leq u_2$ (Refl-time)
2. $x :: \Sigma; \Psi; E, E'; \Gamma, x : s \circ [u_1, u_2] \xrightarrow{\alpha} s \circ [u_1, u_2]$ (Rule (hyp) on 1)

Case.
$$\frac{\begin{array}{c} M_1 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s_1 \supset s_2 \circ [u_1, u_2] \searrow \vec{C}_1; \vec{i}_1 \qquad M_2 :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s_1 \circ [u'_1, u'_2] \searrow \vec{C}_2; \vec{i}_2 \\ \vec{C}_3 \bowtie \vec{C}_4 = (\Psi \models u_1 \leq u'_1), (\Psi \models u'_2 \leq u_2) \qquad \models \vec{C}_3 \end{array}}{(\texttt{pf\_impE}\ M_1\ M_2\ u'_1\ u'_2) :: \Sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s_2 \circ [u'_1, u'_2] \searrow \vec{C}_1, \vec{C}_2, \vec{C}_4; \vec{i}_1, \vec{i}_2}\,{\supset}\text{E}$$

1. $\models \vec{C}_1, \vec{C}_2, \vec{C}_4$ (Assumption)

2. $\models \vec{C_3}$ (4th premise)

3. $\models \vec{C_3}, \vec{C_4}$ (1,2)

4. $\vec{C_3} \bowtie \vec{C_4} = (\Psi \models u_1 \leq u_1'), (\Psi \models u_2' \leq u_2)$ (3rd premise)

5. $\Psi \models u_1 \leq u_1' \leq u_1'$ and $\Psi \models u_2' \leq u_2' \leq u_2$ (3,4,Refl-time)

6. $E' \models \vec{i_1}, \vec{i_2}$ (Assumption)

7. $M_1 :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s_1 \supset s_2 \circ [u_1, u_2]$ (i.h. on 1st premise, 1, 6)

8. $M_2 :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s_1 \circ [u_1', u_2']$ (i.h. on 2nd premise, 1, 6)

9. $(\texttt{pf\_impE}\ M_1\ M_2\ u_1'\ u_2') :: \Sigma; \Psi; E, E'; \Gamma \xrightarrow{\alpha} s_2 \circ [u_1', u_2']$ (Rule ($\supset$E) on 7,8,5)

□

We also need the following substitution lemma in order to prove Theorem 4.1.

Lemma B.2 (Term substitution)**.**

1. *If* $M :: \Sigma, v{:}\sigma; \Psi; E; \Gamma \overset{\alpha}{\Longrightarrow} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}$ *and* $\Sigma \vdash t : \sigma$, *then* $M[t/v] :: \Sigma; \Psi[t/v]; E[t/v]; \Gamma[t/v] \overset{\alpha[t/v]}{\Longrightarrow} s[t/v] \circ [u_1[t/v], u_2[t/v]] \searrow \vec{C}[t/v]; \vec{i}[t/v]$.

2. *If* $M :: \Sigma, v{:}\sigma; \Psi; E; \Gamma \overset{\alpha}{\Longleftarrow} s \circ [u_1, u_2] \searrow \vec{C}; \vec{i}$ *and* $\Sigma \vdash t : \sigma$, *then* $M[t/v] :: \Sigma; \Psi[t/v]; E[t/v]; \Gamma[t/v] \overset{\alpha[t/v]}{\Longleftarrow} s[t/v] \circ [u_1[t/v], u_2[t/v]] \searrow \vec{C}[t/v]; \vec{i}[t/v]$.

Proof. By simultaneous induction on the given derivations, and case analysis of the last rules. For the rules ($\supset$E), (claims), and (consI), the assumption (Substitution-cons) is needed. For the rule (interI), (Substitution-state) is needed. □

B.2 Proof Verification in PCFS

PCFS uses a specialized version of BL's non-deterministic verifier described above. As mentioned in Section 4, in PCFS, the problem is to check that $M :: \Sigma; \cdot; E; \Gamma \xrightarrow{\alpha} \texttt{auth}(k, f, \eta, u)$, where α is a view made of fresh constants, u is the time of access, and E is the environment at time u. Since neither u nor E is known when verification is done, the verifier instead tries to check that $M :: \Sigma, \mathsf{ctime}{:}\mathsf{time}; \cdot; \cdot; \Gamma \xrightarrow{\alpha} \texttt{auth}(k, f, \eta, \mathsf{ctime})$, where ctime is a symbolic constant that represents the actual time of access.

Any constraints containing ctime encountered during verification are output from the verification procedure (on the right of $\searrow$); others are checked immediately. This method is sound because Lemma B.1 shows that any strategy for deciding which constraints to check during verification, and which to output is correct. The constraints written to the output are then also written in the procap produced, and get checked when the procap is verified. (At that time, ctime is substituted by the actual time of access.) More precisely, whenever the verifier needs to construct $\vec{C_1}$ and $\vec{C_2}$ such that $\vec{C_1} \bowtie \vec{C_2} = \vec{C}$ (rules ($\supset$E), (claims), and (consI)), it sets $\vec{C_1}$ to those judgments $\Psi \models c$ in $\vec{C}$ that *do not* contain ctime. $\vec{C_2}$ contains the remaining judgments. $\vec{C_1}$ is checked immediately by the verifier, whereas $\vec{C_2}$ is written to the output procap.

Summary of proof verification in PCFS. Proof verification in PCFS can be summarized as follows. The verifier is given a proof term M, Σ (from a trusted file), Γ (in the form of certificates), k, f, and η. It tries to check the proof by establishing the judgment $M :: \Sigma, \mathsf{ctime{:}time}; \cdot; \cdot; \Gamma \overset{\alpha}{\Longrightarrow} \mathtt{auth}(k, f, \eta, \mathsf{ctime}) \searrow \vec{C}; \vec{i}$ for some $\vec{C}$ and $\vec{i}$, resolving non-determinism in splitting constraint judgments as described above. If this succeeds, it issues the procap $\langle \psi, \vec{C}, \vec{i}, \Xi \rangle$ where $\psi = \langle k, f, \eta \rangle$ and Ξ is a cryptographic signature.

The procap can be checked during access at time u in environment E by ensuring that $E \models \vec{i}$ and that $\models \vec{C}[u/\mathsf{ctime}]$. We now show that these checks are sufficient to show that $M :: \Sigma; \cdot; E; \Gamma \xrightarrow{\alpha} \mathtt{auth}(k, f, \eta, u)$, i.e., the authorization is valid *at the time of access.*

Theorem B.3 (Soundness of enforcement; Theorem 4.1)**.** *Suppose* $M :: \Sigma, \mathsf{ctime{:}time}; \cdot; \cdot; \Gamma \overset{\alpha}{\Longrightarrow} \mathtt{auth}(k, f, \eta, \mathsf{ctime}) \searrow \vec{C}; \vec{i}$, *where* ctime *is a new constant. Let u be a (later) point of time at which the environment is E, and suppose that:*

1. $\models \vec{C}[u/\mathsf{time}]$

2. $E \models \vec{i}$

Then, $M :: \Sigma; \cdot; E; \Gamma \xrightarrow{\alpha} \mathtt{auth}(k, f, \eta, u)$.

Proof. We reason as follows:

1. $M :: \Sigma, \mathsf{ctime{:}time}; \cdot; \cdot; \Gamma \overset{\alpha}{\Longrightarrow} \mathtt{auth}(k, f, \eta, \mathsf{ctime}) \searrow \vec{C}; \vec{i}$ (Assumption)

2. $M :: \Sigma; \cdot; \cdot; \Gamma \overset{\alpha}{\Longrightarrow} \mathtt{auth}(k, f, \eta, u) \searrow \vec{C}[u/\mathsf{ctime}]; \vec{i}$ (Lemma B.2; ctime is fresh)

3. $\models \vec{C}[u/\mathsf{time}]$ (Assumption)

4. $E \models \vec{i}$ (Assumption)

5. $M :: \Sigma; \cdot; E; \Gamma \xrightarrow{\alpha} \mathtt{auth}(k, f, \eta, u)$ (Lemma B.1 on 2,3,4)

□