Journal of Logic and Computation Advance Access published June 19, 2015

Justification logic and audited computation

FRANCISCO BAVERA, *UNRC and CONICET, Argentina.*
E-mail: fbavera@gmail.com

EDUARDO BONELLI, *UNQ, CONICET and ITBA, Argentina.*
E-mail: ebonelli@unq.edu.ar

Abstract

Justification Logic (**JL**) is a refinement of modal logic in which assertions of knowledge and belief are accompanied by justifications: the formula $[\![s]\!]A$ states that s is a 'reason' for knowing/believing A. We study the computational interpretation of **JL** via the Curry–Howard isomorphism in which the modality $[\![s]\!]A$ is interpreted as: s is a type derivation justifying the validity of A. The resulting lambda calculus is such that its terms are aware of the reduction sequence that gave rise to them. This serves as a basis for understanding systems, many of which belong to the security domain, in which computation is history-aware.

Keywords: Modal Logic, Justification Logic, Curry-Howard, Lambda Calculus, History-based computation.

1 Introduction

This article is concerned with the computational interpretation of *Justification Logic* (more precisely, the initial Justification Logic system introduced under the name of *Logic of Proofs*) [4–6] (**JL**). **JL** is a refinement of modal logic in which assertions of knowledge and belief are accompanied by *justifications*: the modality $[\![s]\!]A$ states that s is a 'reason' for knowing/believing A. The starting point of this work is the observation that if s is understood as a typing derivation of a term of type A, then a term of type $[\![s]\!]A$ should include some encoding of s. If this typing derivation is seen as a logical derivation, then any normalization steps applied to it would produce a new typing derivation for A. Moreover, its relation to s would have to be made explicit in order for derivations to be closed under normalization (in type systems parlance: for Subject Reduction (SR) to hold). This suggests that the computational interpretation of **JL** should be a programming language that records its computation history. This work is an attempt at making these ideas precise.

1.1 JL and Hypothetical JL

In **JL** proofs are represented as combinatory terms (*proof polynomials*). Proof polynomials are constructed from proof variables and constants using two operations: application '$\cdot$' and introspection '!'. The usual propositional connectives are augmented by a new one: given a proof polynomial s and a proposition A build $[\![s]\!]A$. The intended reading is: '*s is a proof of A*'. The axioms and inference schemes of **JL** are as follows where, in this work, we restrict the first axiom to minimal logic rather than classical logic:

A0. Axiom schemes of minimal logic in the language of **JL**	
A1. $[\![s]\!]A \supset A$	*"verification"*
A2. $[\![s]\!](A \supset B) \supset ([\![t]\!]A \supset [\![s \cdot t]\!]B)$	*"application"*
A3. $[\![s]\!]A \supset [\![!s]\!][\![s]\!]A$	*"introspection"*
R1. $\Gamma \vdash A \supset B$ and $\Gamma \vdash A$ implies $\Gamma \vdash B$	*"modus ponens"*
R2. If **A** is an axiom **A0**-**A3**, and c is a proof constant, then $\vdash [\![c]\!]\mathbf{A}$	*"necessitation"*

doi:10.1093/logcom/exv037

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi_1}{\Delta; a : A \vdash B \mid s}}{\Delta; \cdot \vdash A \supset B \mid \lambda a^A.s}\supset \mathsf{I} \qquad \dfrac{\pi_2}{\Delta; \cdot \vdash A \mid t}}{\Delta; \cdot \vdash B \mid (\lambda a^A.s) \cdot t}\supset \mathsf{E}}{\Delta; \Gamma \vdash [\![(\lambda a^A.s) \cdot t]\!] B \mid !(\lambda a^A.s) \cdot t}\Box \mathsf{I}_0 \qquad \dfrac{\dfrac{\pi_3}{\Delta; \cdot \vdash B \mid s^a_t}}{\Delta; \Gamma \vdash [\![(\lambda a^A.s) \cdot t]\!] B \mid !(\lambda a^A.s) \cdot t}\Box \mathsf{I}$$

FIGURE 1. Failure of SR for the naive modal introduction scheme.

In [3] a Natural Deduction presentation of this minimal fragment of **JL** is studied. It is based on work on *judgemental reconstruction* [18] of intuitionistic **S4** [13–15]; judgements are introduced in which a distinction is made between propositions whose *truth* is assumed from those whose *validity* is assumed. Moreover, the notion of *proof code* is incorporated:

$$\Delta; \Gamma \vdash A \mid s.$$

This judgement reads as: '*proof code s is evidence that A is true, assuming validity of hypothesis in* Δ *and truth of hypothesis in* Γ'. In such hypothetical judgements with proof codes, evidence s is a constituent part of it without which the proposed reading is no longer possible. Its importance is reflected in the following introduction scheme for the modality, where '$\cdot$' stands for an empty context:

$$\dfrac{\Delta; \cdot \vdash A \mid s}{\Delta; \Gamma \vdash [\![s]\!] A \mid !s}\Box \mathsf{I}_0.$$

This scheme internalizes logical derivations of validity: If s is evidence that A is unconditionally true, then it is true that s is a proof of A. The new witness to this fact is registered as the evidence $!s$. The '!' operator is reminiscent of that of proof polynomials. However, in our article, proof terms such as s encode Natural Deduction derivations and thus are no longer the proof polynomials of **JL**.

Unfortunately, the system which includes $\Box \mathsf{I}_0$ is not closed under substitution of derivations. Eg. the derivation in Figure 1 (left) would produce, after a step of normalization, the derivation of Figure 1 (right) where π_3 is obtained from π_1, π_2 and an appropriate substitution principle (see Section 2.2). The latter derivation is invalid since proof codes s^a_t (replace all free occurences of a in s by t) and $(\lambda a^A.s) \cdot t$ do not coincide. SR may be regained, however, by introducing a judgement stating *compatibility* of proof codes, where e below is called a *compatibility code*[1] *or, more suggestively*, trail and is witness to the compatibility between the derivations denoted by the proof codes s and t:

$$\Delta; \Gamma \vdash \mathsf{Eq}(A, s, t) \mid e \tag{1}$$

together with a new scheme:

$$\dfrac{\Delta; \Gamma \vdash A \mid s \quad \Delta; \Gamma \vdash \mathsf{Eq}(A, s, t) \mid e}{\Delta; \Gamma \vdash A \mid t}\mathsf{Eq}.$$

Examples of trails are $\mathtt{r}(s)$ if $\mathsf{Eq}(A, s, s)$ is proved using reflexivity, $\mathtt{ba}(s_1, s_2)$ (for appropriate proof codes s_1 and s_2) if $\mathsf{Eq}(A, s, t)$ is proved using β-equality, $\mathtt{app}(e_1, e_2)$ if s and t encode eliminations of implication and $\mathsf{Eq}(A, s, t)$ is proved by proving the compatibility of the arguments to these

[1] They are essentially 'proof terms' in rewriting theory [25, Ch. 8].

eliminations (with e_1 and e_2), and $\mathfrak{t}(e_1, e_2)$ if it is proved using transitivity. Details are provided in Section 2.2, however, the interested reader can take a peek at Figure 3.

Normalization of derivations gives rise to numerous instances of Eq. These may be permuted past any inference scheme *except* a box introduction $\Box\mathsf{I}_0$, as may be easily verified from the full set of inference schemes introduced in Section 2. This suggests a modified introduction scheme for $\Box$:

$$\frac{\Delta; \cdot \vdash A \,|\, s \quad \Delta; \cdot \vdash \mathsf{Eq}(A, s, t) \,|\, e}{\Delta; \Gamma \vdash [\![t]\!] A \,|\, t}\ \Box\mathsf{I}_1$$

and a normalization procedure over *canonical* derivations where occurrences of Eq are permuted until they reach the innermost instance $\Box\mathsf{I}_1$ at which point they permute to the upper right-hand hypothesis (cf. Section 3). This procedure spells out a natural notion of *locality* of trails given in terms of *scope*: trails are local to the innermost box introduction scheme. The resulting system, may be shown strongly normalizing and confluent by applying properties of higher-order rewrite systems [3].

Although the resulting system in which reduction and their corresponding trails are confined to local computation units is satisfying from a purely logical point of view, from a programming language point of view it would be desirable for trails themselves to be *reifiable*. We are aware of two such examples. The first is the need to inspect trails in order to determine whether access to resources is granted or not [1] (see Section 1.2). Another is a study on information confidentiality based on incorporating permissions into the information flow analysis and then using history-based access control to determine the current available permissions [7]. Both examples modify the control flow of the program depending on the inspection of trails. We therefore incorporate trail inspection and perform an analysis of normalization in the resulting logical system. Normalization in this extended setting suggests how to approach non-trivial operations, such as trail variable renaming and trail persistence, of the run-time environment of programming languages modelled on these ideas, that must be taken into account in order to obtain sound implementations (in the sense that SR is upheld). Access to trails is achieved through *trail variables* together with an associated recursion scheme:

$$\frac{\alpha : \mathsf{Eq}(A) \in \Sigma \quad \Delta; \cdot; \cdot \vdash \mathcal{T}^B \,|\, \theta}{\Delta; \Gamma; \Sigma \vdash B \,|\, \alpha\theta}\ \mathsf{TI}\cdot$$

Here Σ is a context of trail variables. $\mathcal{T}^B$ is a fixed mapping from trail constructors (e.g. $\mathfrak{r}, \mathfrak{ba}, \mathfrak{app}, \mathfrak{t}$) to logical propositions while θ is a mapping from trail constructors to proof codes, parameterized over a proposition B. This mapping is such that if c is a trail constructor, then $\theta(c)$ is the proof code associated with the proposition $\mathcal{T}^B(c)$. Some sample defining cases, where "$\triangleq$" stands for definitional equality, are $\mathcal{T}^B(\mathfrak{ba}) \triangleq B$, the type of the branch of the recursor in case that a β-step took place and $\mathcal{T}^B(\mathfrak{app}) \triangleq B \supset B \supset B$, the type of the branch of the recursor in case a reduction step took place under an application. Full details are developed in Section 2.

Finally, the local (as mentioned above) scope of trail variables is reflected in the, now final, introduction scheme for $\Box$. In this scheme, the proof code $\Sigma.t$ binds all free occurrences of the trail

variables in Σ and the context Σ' of the judgement in the conclusion allows weakening of trail variables:

$$\frac{\Delta;\cdot;\Sigma \vdash A \,|\, s \quad \Delta;\cdot;\Sigma \vdash \mathsf{Eq}(A,s,t) \,|\, e}{\Delta;\Gamma;\Sigma' \vdash [\![\, \Sigma.t \,]\!] A \,|\, \Sigma.t}\ \Box\mathsf{I}\,.$$

The associated principal reduction is when an instance of TI is used in a derivation of the top left judgement of $\Box\mathsf{I}$ (see Section 3).

Following this informal discussion of *hypothetical* **JL** (**JL**$^{\mathfrak{h}}$), we next briefly describe its term assignment $\lambda^{\mathfrak{h}}$, the *calculus of audited units*, through an example. The formal presentation of **JL**$^{\mathfrak{h}}$ is given in Section 2.

1.2 A calculus of audited units

We describe a sample program in $\lambda^{\mathfrak{h}}$ that models Abadi and Fournet's [1] mechanism for access control based on execution history. First we recall the mechanism itself. Proposed as an enhanced alternative to Java stack-inspection, it consists in controlling access to objects depending on which functions were invoked previously in time. Each function is (statically) assigned a set of permissions. At any moment, the current set of permissions is computed by taking the intersection of the permissions of all function calls in the history of the computation. For example, assuming we have untrusted user-defined function **bad**, trusted user-defined function **eraseFile**, a system function **Win32Delete** that deletes a file and that **eraseFile** calls **Win32Delete**, we might have the following assigned permissions to each of these: *perms*(**bad**) $\triangleq \emptyset$, *perms*(**Win32Delete**) $\triangleq \{\mathit{FileIOPerm}\}$ and *perms*(**eraseFile**) $\triangleq \{\mathit{FileIOPerm}\}$. Here the function called **bad** has no permissions, whereas **eraseFile** is assigned the permission to perform file input/output. The only valid executions of a program involving the above-mentioned functions is when **bad** was never invoked.

In $\lambda^{\mathfrak{h}}$ computation which is to be audited takes place in so-called *audited (computation) units* which are typed using $\Box\mathsf{I}$. Such expressions are of the form $!_e^{\alpha_1,\ldots,\alpha_n} M$ where M is the *body*, e the *trail* of computation having produced M and α_i, $i \in 1..n$, the *trail variables* that are used for consulting the computation trail. The scope of these variables is M. The expression e is part of the run-time environment, hence not explicitly declared by the programmer, and is local to the audited unit in the sense that different audited units will have their own trails. Consider the following top-level function:

$$\begin{aligned}\textbf{eraseFile} \doteq\ !^{\alpha}\lambda a^{\mathit{Str}}.\mathsf{if}\ \mathit{FileIOPerm} \in \alpha\vartheta\ \mathsf{then}\ \textbf{Win32Delete}\, a \\ \mathsf{else}\ \textbf{securityException};\end{aligned} \tag{2}$$

This audited unit uses only one trail variable, namely α. Also, the trail e is not depicted since, as mentioned, it is not necessary for the programmer to specify it given that it will be maintained by the run-time system. We mention its existence all the same since it will appear during the term reduction process. The body of the audited unit consists of a function which, once it has been supplied with an appropriate argument for a of string type, either deletes the file if it has the appropriate permission (*FileIOPerm*) or raises an exception if it does not.

The trail variable α supplies the function with access to the computation trail which will be extant when it is evaluated. Evaluation of trail variables inside an audited unit consists in first looking up the trail and then immediately traversing it, replacing each constructor of the trail with a term of the appropriate type.[2] The mapping ϑ that replaces trail constructors with terms is called a *trail*

[2]In the same way as one recurs over lists using fold in functional programming, replacing nil and cons by appropriate terms.

replacement. All instances of trail inspection are thus written $\alpha\vartheta$ where α is a (local) trail variable and ϑ a trail replacement. In our example, the trail replacement should produce the intersection of the permissions of all functions invoked during the history of the computation. It thus takes the form:

$$\begin{array}{rlrl} \vartheta(\mathfrak{r}) = \vartheta(\mathfrak{ti}) & \triangleq \emptyset & \vartheta(\mathfrak{trpl}) & \triangleq \lambda\vec{a}^{\mathbb{N}}.a_1 \cap .. \cap a_{11} \\ \vartheta(\mathfrak{s}) = \vartheta(\mathfrak{abs}) & \triangleq \lambda a^{\mathbb{N}}.a & \vartheta(\mathfrak{ba}) = \vartheta(\mathfrak{bb}) & \triangleq \emptyset \\ \vartheta(\mathfrak{t}) = \vartheta(\mathfrak{app}) = \vartheta(\mathfrak{let}) & \triangleq \lambda a^{\mathbb{N}}.\lambda b^{\mathbb{N}}.a \cap b & \vartheta(\mathfrak{def}_{\mathbf{f}}) & \triangleq \{perms(\mathbf{f})\} \end{array} .$$

Here $\mathbb{N}$ is the type of natural numbers; $\lambda\vec{a}^{\mathbb{N}}$ abbreviates $\lambda a_1^{\mathbb{N}}.....\lambda a_{11}^{\mathbb{N}}$, 11 being the total number of trail constructors in $\lambda^{\mathfrak{h}}$; the expressions $\mathfrak{r}, \mathfrak{ba}, \mathfrak{app}, \mathfrak{t}$, etc., are the trail constructors (cf. Section 2.1) which are maintained by the run-time system. In particular, $\mathfrak{def}_{\mathbf{f}}$ is registered in the trail when an unfolding of the user-defined function **f** takes place. Therefore, as we will see, evaluation of $FileIOPerm \in \alpha\vartheta$ shall return false if e includes the trail constructor $\mathfrak{def}_{\mathbf{bad}}$.

We revisit this example in Section 4.2 in order to illustrate its reduction behaviour, once the formal definitions of $\lambda^{\mathfrak{h}}$ are in place.

1.3 Contributions and related work

We develop a proof theoretical analysis of a λ-calculus which produces a trail of its execution. This builds on ideas stemming from **JL**, a judgemental analysis of modal logic [13–15, 18] and Contextual Modal Type Theory [23]. More precisely, we argue how a fragment of **JL** whose notion of validity is relative to a context of trail variables may be seen, via the Curry–Howard interpretation, as a type system for a calculus that records its computation history.

S. Artemov introduced **JL** in [4–6]. For Natural Deduction and Sequent Calculus presentations consult [3, 5, 12]. Computational interpretation of proofs in **JL** is studied in [2, 3, 10]; however, none of these address computation trails. From a type theoretic perspective, we should mention the theory of dependent types [8] where types may depend on terms, in much the same way that a type $[\![s]\!]A$ depends on the proof code s. However, dependent type theory lacks a notion of internalization of derivations as is available in **JL**.

Contributions of **JL** towards mobile code interpretations of intuitionistic S4 [17, 19–22] have also been explored. The result is a calculus of *certified mobile units* in which a unit consists of mobile code and a certificate [10]. The type system guarantees that when these mobile units are composed to form new ones, the certificates that are constructed out of those of the composed units, are correct.

Finally, a computational interpretation via the Curry–Howard isomorphism for **JL**, that is based on classical logic and including the plus proof polynomial constructor, has been developed too [11].

1.4 Structure of the article

Section 2 introduces $\mathbf{JL}^{\mathfrak{h}}$. Section 3 studies normalization in this system. We then introduce a term assignment for this logic in order to obtain a lambda calculus with computation history trails, $\lambda^{\mathfrak{h}}$. This calculus is endowed with a call-by-value operational semantics and type safety of this semantics w.r.t. the type system is proved. Section 5 addresses strong normalization. Finally, we conclude and suggest further avenues for research. This work is a revision and extension of [9].

2 Hypothetical JL

JL is a modal logic of provability which has a sound and complete arithmetical semantics. This section introduces $\mathbf{JL}^{\mathfrak{h}}$ a Natural Deduction presentation for a fragment of **JL**, namely the restriction of **JL** to minimal propositional **JL** without the 'plus' polynomial proof constructor. The inference schemes we shall define give meaning to *hypothetical judgements with proof codes* $\Delta;\Gamma;\Sigma\vdash A\,|\,s$ whose intended reading is: '*s is evidence that A is true under validity hypothesis* Δ*, truth hypothesis* Γ *and trail hypothesis* Σ.'

We assume given term variables $a,b,\ldots$, audited unit variables $u,v,\ldots$ and trail variables $\alpha,\beta,\ldots$. A proposition in $\mathbf{JL}^{\mathfrak{h}}$ is either a propositional variable P, an implication $A\supset B$ or a modality $[\![\,\Sigma.s\,]\!]A$:

$$A ::= P\,|\,A\supset A\,|\,[\![\,\Sigma.s\,]\!]A.$$

In $[\![\,\Sigma.s\,]\!]A$, 'Σ.' binds all occurrences of trail variables in proof code s and hence may be renamed at will. Contexts are given by the following grammar:

$$\begin{array}{lll}\Delta ::= \cdot\,|\,\Delta,u{:}A[\Sigma] & \text{Validity context}\\ \Gamma ::= \cdot\,|\,\Gamma,a{:}A & \text{Truth context}\\ \Sigma ::= \cdot\,|\,\Sigma,\alpha{:}\mathsf{Eq}(A) & \text{Trail context}\end{array}.$$

Contexts are considered multisets; '$\cdot$' denotes the empty context. Variables in Σ are assigned a type of the form $\mathsf{Eq}(A)$. The type $\mathsf{Eq}(A)$ in an assignment $\alpha:\mathsf{Eq}(A)$ may informally be understood as $\exists x,y.\mathsf{Eq}(A,x,y)$ (where x,y stand for arbitrary type derivations of propositions of type A) since α stands for a proof of compatibility of two type derivations of propositions of type A about which nothing more may be assumed. In the subexpression of a judgement of the form $\Delta;\Gamma;\Sigma$ we assume all variables to be fresh. Proof codes are defined as:

$$\begin{array}{lll}s ::= a & & \text{truth hypothesis}\\ \quad|\quad \lambda a^{A}.s & & \text{abstraction}\\ \quad|\quad s\cdot s & & \text{application}\\ \quad|\quad \langle u;\sigma\rangle & & \text{validity hypothesis}\\ \quad|\quad \Sigma.s & & \text{box introduction}\\ \quad|\quad \textsc{let}(u^{A[\Sigma]}.s,s) & & \text{box elimination}\\ \quad|\quad \alpha\theta & & \text{trail inspection}\end{array}$$

They bear witness to proofs of propositions, encoding instances of the inference schemes of $\mathbf{JL}^{\mathfrak{h}}$ which we shall introduce shortly. Proof codes rely on the following supporting notions:

$$\begin{array}{rl}\text{Renaming } \sigma ::= & \{\alpha_1/\beta_1,\ldots,\alpha_n/\beta_n\}\\ \text{Trail replacement } \theta ::= & \{\mathfrak{r}/s,\mathfrak{s}/s,\mathfrak{t}/s,\mathfrak{ba}/s,\mathfrak{bb}/s,\mathfrak{ti}/s,\mathfrak{abs}/s,\mathfrak{app}/s,\mathfrak{let}/s,\mathfrak{trp}/s\}\end{array}.$$

A *renaming* σ is a bijective map over trail variables. A trail replacement is a mapping θ from the set of trail constructors to proof codes. Free truth variables of s ($\mathsf{fvT}(s)$), free validity variables of s ($\mathsf{fvV}(s)$) and free trail variables of s ($\mathsf{fvTrl}(s)$) are defined as expected.

2.1 Inference schemes

The meaning of hypothetical judgements with proof codes is given by the axiom and inference schemes of Figure 2 and determine the $\mathbf{JL}^{\mathfrak{h}}$ *system*. We briefly comment on the schemes. First note that Σ and (hence also) Γ are affine hypothesis whereas those in Δ are intuitionistic (cf. Rem. 4). The axiom scheme Var states that judgement '$\Delta;\Gamma;\Sigma\vdash A\,|\,a$' is evident in itself: if we assume a

$$\frac{a : A \in \Gamma}{\Delta;\Gamma;\Sigma \vdash A \mid a}\ \mathsf{Var} \qquad \frac{\Delta;\Gamma, a : A;\Sigma \vdash B \mid s}{\Delta;\Gamma;\Sigma \vdash A \supset B \mid \lambda a^{A}.s}\ \supset \mathsf{I}$$

$$\frac{\Delta;\Gamma_1;\Sigma_1 \vdash A \supset B \mid s \quad \Delta;\Gamma_2;\Sigma_2 \vdash A \mid t}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash B \mid s \cdot t}\ \supset \mathsf{E} \qquad \frac{u : A[\Sigma] \in \Delta \quad \Sigma\sigma \subseteq \Sigma'}{\Delta;\Gamma;\Sigma' \vdash A \mid \langle u;\sigma \rangle}\ \mathsf{mVar}$$

$$\frac{\Delta;\cdot;\Sigma \vdash A \mid s}{\Delta;\cdot;\Sigma \vdash \mathsf{Eq}(A,s,t) \mid e}\ \Box\mathsf{I} \qquad \frac{\Delta;\Gamma_1;\Sigma_1 \vdash [\![\Sigma.r]\!]A \mid s \quad \Delta, u : A[\Sigma];\Gamma_2;\Sigma_2 \vdash C \mid t}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash C^{u}_{\Sigma.r} \mid \mathrm{LET}(u^{A[\Sigma]}.t, s)}\ \Box\mathsf{E}$$

$$\frac{\alpha : \mathsf{Eq}(A) \in \Sigma \quad \Delta;\cdot;\cdot \vdash \mathcal{T}^{B} \mid \theta}{\Delta;\Gamma;\Sigma \vdash B \mid \alpha\theta}\ \mathsf{TI} \qquad \frac{\Delta;\Gamma;\Sigma \vdash A \mid s \quad \Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,s,t) \mid e}{\Delta;\Gamma;\Sigma \vdash A \mid t}\ \mathsf{Eq}$$

FIGURE 2. Inference schemes for hypothetical judgements with proof codes.

is evidence that proposition A is true, then we may immediately conclude that A is true with proof code a. Similarly, the assumption that A is valid allows us to conclude that it is true, as indicated by mVar. However, since the validity of A depends on trail assumptions Σ, they must be renamed to fit the current context. The schemes for introduction and elimination of implication need no further explanation. The introduction scheme for the modality has already been motivated in the introduction. The compatibility code e of $\Box\mathsf{I}$ can take one of the following forms, where $\mathfrak{trpl}(e_1,\ldots,e_{10})$[3] is usually abbreviated $\mathfrak{trpl}(\overline{e})$:

$e ::=$	$\mathfrak{r}(s)$	reflexivity
$\mid$	$\mathfrak{s}(e)$	symmetry
$\mid$	$\mathfrak{t}(e,e)$	transitivity
$\mid$	$\mathfrak{ba}(a^{A}.s,s)$	β
$\mid$	$\mathfrak{bb}(u^{A[\Sigma]}.s,\Sigma.s)$	$\beta_{\Box}$
$\mid$	$\mathfrak{ti}(\theta,\alpha)$	trail inspection
$\mid$	$\mathfrak{abs}(a^{A}.e)$	abstraction compatibility
$\mid$	$\mathfrak{app}(e,e)$	application compatibility
$\mid$	$\mathfrak{let}(u^{A[\Sigma]}.e,e)$	let compatibility
$\mid$	$\mathfrak{trpl}(e_1,\ldots,e_{10})$	replacement compatibility

The set of compatibility code constructors is denoted $\mathfrak{C} \triangleq \{\mathfrak{r},\mathfrak{s},\mathfrak{t},\mathfrak{ba},\mathfrak{bb},\mathfrak{ti},\mathfrak{abs},\mathfrak{app},\mathfrak{let},\mathfrak{trpl}\}$.

The schemes defining the judgement (1) are given in Figure 3. There are four proof code compatibility axioms (EqRefl, Eqβ, Eq$\beta_{\Box}$ and EqTI) and six inference schemes (the rest). The axioms are used for recording principal contractions (Section 3) at the root of a derivation and schemes EqAbs, EqApp, EqLet and EqTI enable the same recording inside subderivations. Regarding trail inspection (EqTI in Figure 3) recall from the introduction that we append each reference to a trail variable with a trail replacement. Therefore, the trail replacement has to be accompanied by proof codes, one for each term that is to replace a trail constructor. The proof code for each of these proofs is grouped as θ and is called a *proof code trail replacement*: $\Delta;\cdot;\cdot \vdash \mathcal{T}^{B} \mid \theta$ which is a shorthand for

[3]We have dropped the code for top-level function declarations used in the introduction, hence there are a total of 10.

$$\frac{\Delta;\Gamma;\Sigma \vdash A \mid s}{\Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,s,s) \mid \mathfrak{r}(s)}\ \mathsf{EqRefl} \qquad \frac{\Delta;\Gamma_1, a:A;\Sigma_1 \vdash B \mid s \quad \Delta;\Gamma_2;\Sigma_2 \vdash A \mid t}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash \mathsf{Eq}(B, s_t^a, (\lambda a^A.s)\cdot t) \mid \mathfrak{ba}(a^A.s,t)}\ \mathsf{Eq}\beta$$

$$\frac{\Delta;\cdot;\Sigma_1 \vdash A \mid r \quad \Delta;\cdot;\Sigma_1 \vdash \mathsf{Eq}(A,r,s) \mid e \quad \Delta, u:A[\Sigma_1];\Gamma_2;\Sigma_2 \vdash C \mid t \quad \Gamma_2 \subseteq \Gamma_3 \quad \Sigma_2 \subseteq \Sigma_3}{\Delta;\Gamma_3;\Sigma_3 \vdash \mathsf{Eq}(C^u_{\Sigma_1.s}, t^u_{\Sigma_1.s}, \mathrm{LET}(u^{A[\Sigma_1]}.t, \Sigma_1.s)) \mid \mathfrak{bb}(u^{A[\Sigma_1]}.t, \Sigma_1.s)}\ \mathsf{Eq}\beta_\Box$$

$$\frac{\Delta;\cdot;\Sigma_1 \vdash \mathsf{Eq}(A,s,t) \mid e \quad \Delta;\cdot;\cdot \vdash \mathcal{T}^B \mid \theta \quad \alpha : \mathsf{Eq}(A) \in \Sigma_2}{\Delta;\Gamma;\Sigma_2 \vdash \mathsf{Eq}(B, e\theta, \alpha\theta) \mid \mathfrak{ti}(\theta,\alpha)}\ \mathsf{EqTI}$$

$$\frac{\Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,s,t) \mid e}{\Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,t,s) \mid \mathfrak{s}(e)}\ \mathsf{EqSym} \qquad \frac{\Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,s_1,s_2) \mid e_1 \quad \Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,s_2,s_3) \mid e_2}{\Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,s_1,s_3) \mid \mathfrak{t}(e_1,e_2)}\ \mathsf{EqTrans}$$

$$\frac{\Delta;\Gamma, a:A;\Sigma \vdash \mathsf{Eq}(B,s,t) \mid e}{\Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A \supset B, \lambda a^A.s, \lambda a^A.t) \mid \mathfrak{abs}(a^A.e)}\ \mathsf{EqAbs} \qquad \frac{\Delta;\Gamma_1;\Sigma_1 \vdash \mathsf{Eq}(A \supset B, s_1, s_2) \mid e_1 \quad \Delta;\Gamma_2;\Sigma_2 \vdash \mathsf{Eq}(A,t_1,t_2) \mid e_2}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash \mathsf{Eq}(B, s_1\cdot t_1, s_2\cdot t_2) \mid \mathfrak{app}(e_1,e_2)}\ \mathsf{EqApp}$$

$$\frac{\Delta;\Gamma_1;\Sigma_1 \vdash \mathsf{Eq}([\![\Sigma.r]\!]A, s_1, s_2) \mid e_1 \quad \Delta, u:A[\Sigma];\Gamma_2;\Sigma_2 \vdash \mathsf{Eq}(C,t_1,t_2) \mid e_2}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash \mathsf{Eq}(C^u_{\Sigma.r}, \mathrm{LET}(u^{A[\Sigma]}.t_1, s_1), \mathrm{LET}(u^{A[\Sigma]}.t_2, s_2)) \mid \mathfrak{let}(u^{A[\Sigma]}.e_2, e_1)}\ \mathsf{EqLet}$$

$$\frac{\alpha : \mathsf{Eq}(A) \in \Sigma \quad \Delta;\cdot;\cdot \vdash \mathsf{Eq}(\mathcal{T}^B(c), \theta'(c), \theta(c)) \mid e_c \quad \text{for each } c \in \mathfrak{C}}{\Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(B, \alpha\theta', \alpha\theta) \mid \mathfrak{trpl}(\overline{e})}\ \mathsf{EqRpl}$$

FIGURE 3. Schemes defining proof code compatibility judgement.

$\Delta;\cdot;\cdot \vdash \mathcal{T}^B(c) \mid \theta(c)$, for each $c \in \mathfrak{C}$, where $\mathcal{T}^B(c)$ is the type of term that replaces the trail constructor c. These types are defined as follows:

$$\begin{array}{rlrl} \mathcal{T}^B(\mathfrak{r}) &\triangleq B & \mathcal{T}^B(\mathfrak{abs}) &\triangleq B \supset B \\ \mathcal{T}^B(\mathfrak{s}) &\triangleq B \supset B & \mathcal{T}^B(\mathfrak{app}) &\triangleq B \supset B \supset B \\ \mathcal{T}^B(\mathfrak{t}) &\triangleq B \supset B \supset B & \mathcal{T}^B(\mathfrak{let}) &\triangleq B \supset B \supset B \\ \mathcal{T}^B(\mathfrak{ba}) = \mathcal{T}^B(\mathfrak{bb}) &\triangleq B & \mathcal{T}^B(\mathfrak{trpl}) &\triangleq \underbrace{B \supset \ldots \supset B}_{10 \text{ copies}} \supset B. \\ \mathcal{T}^B(\mathfrak{ti}) &\triangleq B & & \end{array}$$

Note that there are no congruence schemes for the modality in Figure 3. This follows from our discussion in the introduction motivating $\Box\mathsf{I}_1$, and consequently also $\Box\mathsf{I}$. All instances of Eq resulting from applications of principal contractions will be absorbed by the upper right hypothesis of $\Box\mathsf{I}$, after being permuted towards its innermost instance of $\Box\mathsf{I}$ (cf. Section 3.2; case Eq - $\Box\mathsf{I}$).

REMARK 1
It should be noted that the proof code s in a derivable judgement $\Delta;\Gamma;\Sigma \vdash A \mid s$ does not encode the derivation of this judgement due to the presence of Eq. The term assignment for $\mathbf{JL}^\mathfrak{h}$ is the topic of Section 4.

2.2 *Basic metatheoretic results*

Some basic meta-theoretic results about $\mathbf{JL}^\mathfrak{h}$ are presented next. The judgements in the statement of these results are decorated with terms (such as M, N and $M^a_{N,t}$ below) which may safely be ignored for the time being since they are introduced in Section 4 when we discuss the term assignment for $\mathbf{JL}^\mathfrak{h}$. The first result states that hypothesis may be added without affecting derivability.

LEMMA 2.1 (Weakening)

(1) If $\Delta;\Gamma;\Sigma\vdash M:A\,|\,s$ is derivable, then so is $\Delta';\Gamma';\Sigma'\vdash M:A\,|\,s$, where $\Delta\subseteq\Delta'$, $\Gamma\subseteq\Gamma'$ and $\Sigma\subseteq\Sigma'$.
(2) If $\Delta;\Gamma;\Sigma\vdash \mathsf{Eq}(A,s,t)\,|\,e$ is derivable, then so is $\Delta';\Gamma';\Sigma'\vdash \mathsf{Eq}(A,s,t)\,|\,e$, where $\Delta\subseteq\Delta'$, $\Gamma\subseteq\Gamma'$ and $\Sigma\subseteq\Sigma'$.

The second and third results address two substitution principles. The first principle is about substitution of truth variables. We abbreviate Γ_1,Γ_2 with $\Gamma_{1,2}$. If $\Gamma=\Gamma_1,a{:}A,\Gamma_3$, we write $\Gamma^a_{\Gamma_2}$ for $\Gamma_{1,2,3}$. Also, we write s^a_t for the substitution of all free occurrences of a in s by t and define it as follows (and similarly for e^a_s):

$$
\begin{array}{rcl@{\qquad}rcl}
a^a_s & \triangleq & s & \mathfrak{r}(t)^a_s & \triangleq & \mathfrak{r}(t^a_s)\\
b^a_s & \triangleq & b & \mathfrak{s}(e)^a_s & \triangleq & \mathfrak{s}(e^a_s)\\
(\lambda b^A.t)^a_s & \triangleq & \lambda b^A.t^a_s & \mathfrak{t}(e_1,e_2)^a_s & \triangleq & \mathfrak{t}(e_1{}^a_s,e_2{}^a_s)\\
(t_1\cdot t_2)^a_s & \triangleq & t_1{}^a_s\cdot t_2{}^a_s & \mathfrak{ba}(b^A.r,t)^a_s & \triangleq & \mathfrak{ba}(b^A.r^a_s,t^a_s)\\
\langle u;\sigma\rangle^a_s & \triangleq & \langle u;\sigma\rangle & \mathfrak{bb}(u^{A[\Sigma]}.r,\Sigma.t)^a_s & \triangleq & \mathfrak{bb}(u^{A[\Sigma]}.r^a_s,\Sigma.t)\\
(\Sigma.t)^a_s & \triangleq & \Sigma.t & \mathfrak{ti}(\theta,\alpha)^a_s & \triangleq & \mathfrak{ti}(\theta,\alpha)\\
\mathrm{LET}(u^{A[\Sigma]}.t_2,t_1)^a_s & \triangleq & \mathrm{LET}(u^{A[\Sigma]}.t_2{}^a_s,t_1{}^a_s) & \mathfrak{abs}(b^A.e)^a_s & \triangleq & \mathfrak{abs}(b^A.e^a_s)\\
(\alpha\theta)^a_s & \triangleq & \alpha\theta & \mathfrak{app}(e_1,e_2)^a_s & \triangleq & \mathfrak{app}(e_1{}^a_s,e_2{}^a_s)\\
 & & & \mathfrak{let}(u^{A[\Sigma]}.e_1,e_2)^a_s & \triangleq & \mathfrak{let}(u^{A[\Sigma]}.e_1{}^a_s,e_2{}^a_s)\\
 & & & \mathfrak{trpl}(\overline{e})^a_s & \triangleq & \mathfrak{trpl}(\overline{e^a_s}).
\end{array}
$$

Note that in the third clause of the definition of s^a_t we assume that b has been renamed away from s. Also, in the sixth clause, t is assumed to have no free occurrences of truth variables.

LEMMA 2.2 (Subst. Principle for Truth Hypothesis)
Suppose $\Delta;\Gamma_2;\Sigma_2\vdash N:A\,|\,t$ is derivable and $a{:}A\in\Gamma_1$.

(1) If $\Delta;\Gamma_1;\Sigma_1\vdash M:B\,|\,s$, then $\Delta;\Gamma_1{}^a_{\Gamma_2};\Sigma_{1,2}\vdash M^a_{N,t}:B\,|\,s^a_t$.
(2) If $\Delta;\Gamma_1;\Sigma_1\vdash \mathsf{Eq}(B,s_1,s_2)\,|\,e$, then $\Delta;\Gamma_1{}^a_{\Gamma_2};\Sigma_{1,2}\vdash \mathsf{Eq}(B,(s_1)^a_t,(s_2)^a_t)\,|\,e^a_t$.

The second substitution principle is about substitution of validity variables. Substitution of validity variables in proof codes is denoted $s^u_{\Sigma.t}$. It is defined as follows:

$$
\begin{array}{rcl}
a^u_{\Sigma.s} & \triangleq & a\\
\langle u;\sigma\rangle^u_{\Sigma.s} & \triangleq & s\sigma\\
\langle v;\sigma\rangle^u_{\Sigma.s} & \triangleq & \langle v;\sigma\rangle\\
(\lambda b^A.t)^u_{\Sigma.s} & \triangleq & \lambda b^A.t^u_{\Sigma.s}\\
(t_1\cdot t_2)^u_{\Sigma.s} & \triangleq & t_1{}^u_{\Sigma.s}\cdot t_2{}^u_{\Sigma.s}\\
(\Sigma'.t)^u_{\Sigma.s} & \triangleq & \Sigma'.t^u_{\Sigma.s}\\
\mathrm{LET}(v^{A[\Sigma']}.t_2,t_1)^u_{\Sigma.s} & \triangleq & \mathrm{LET}(v^{A[\Sigma']}.t_2{}^u_{\Sigma.s},t_1{}^u_{\Sigma.s})\\
(\alpha\theta)^u_{\Sigma.s} & \triangleq & \alpha(\theta)^u_{\Sigma.s}.
\end{array}
$$

Note that C^u_s stands for the proposition resulting from replacing all occurrences of u in C with s; in the second clause the domain of σ is Σ; and all trail variables in Σ are renamed by means of the renaming σ. Also, in the last clause, $(\theta)^u_{\Sigma.s}$ denotes the trail replacement where $((\theta)^u_{\Sigma.s})(c)=\theta(c)^u_{\Sigma.s}$, for all $c\in\mathfrak{C}$.

LEMMA 2.3 (Subst. Principle for Validity Hypothesis)
Suppose judgements $\Delta_{1,2};\cdot;\Sigma_1 \vdash M:A\,|\,s$ and $\Delta_{1,2};\cdot;\Sigma_1 \vdash \mathsf{Eq}(A,s,t)\,|\,e_1$ are derivable. Let $\Delta \triangleq \Delta_1, u{:}A[\Sigma_1], \Delta_2$. Then:

(1) If $\Delta;\Gamma;\Sigma_2 \vdash N:C\,|\,r$, then $\Delta_{1,2};\Gamma;\Sigma_2 \vdash N^u_{\Sigma_1.(M,t,e_1)} : C^u_{\Sigma_1.t}\,|\,r^u_{\Sigma_1.t}$.
(2) If $\Delta;\Gamma;\Sigma_2 \vdash \mathsf{Eq}(C,s_1,s_2)\,|\,e_2$, then $\Delta_{1,2};\Gamma;\Sigma_2 \vdash \mathsf{Eq}(C^u_{\Sigma_1.t}, s_1{}^u_{\Sigma_1.t}, s_2{}^u_{\Sigma_1.t})\,|\,e_2{}^u_{\Sigma_1.t}$.

REMARK 2
In this substitution principle, substitution of $u:A[\Sigma_1]$ requires not only a derivation of $\Delta_{1,2};\cdot;\Sigma_1 \vdash M:A\,|\,s$, but *also* its computation history $\Delta_{1,2};\cdot;\Sigma_1 \vdash \mathsf{Eq}(A,s,t)\,|\,e_1$. This logical requirement is reflected in the form of trail persistence in the programming language (cf. Definition 4.1, in particular the clause for $\langle u;\sigma\rangle$).

The last ingredient we require before discussing normalization is the following lemma which is used for computing the results of trail inspection. Similarly to the trail replacement θ already introduced we shall also make use of a mapping ϑ from trail constructors to *terms*: $e\vartheta$ produces a *term* by replacing each trail constructor in e by its associated term via ϑ. An example of such a trail replacement was given in the introduction (cf. Section 1.2). Note that although we refer to both θ and ϑ as trail replacements, $e\theta$ produces a *proof code* (rather than a term) by replacing each trail constructor in e with its associated proof code via θ. If ϑ agrees with θ, then they may be used for eliminating a trail e.

LEMMA 2.4
$\Delta;\cdot;\cdot \vdash \vartheta : \mathcal{T}^B\,|\,\theta$ and $\Delta;\cdot;\Sigma \vdash \mathsf{Eq}(A,s,t)\,|\,e$ implies $\Delta;\cdot;\cdot \vdash e\vartheta : B\,|\,e\theta$.

3 Normalization

Normalization equates derivations and since $\mathbf{JL}^\natural$ internalizes its own derivations by means of proof codes, normalization steps must explicitly relate proof codes in order for SR to hold. Normalization is modelled as a two step process. First a *principal contraction* is applied, then a series of *permutation conversions* follow. Principal contractions introduce witnesses of derivation compatibility. Permutation conversions standardize derivations by moving these witnesses to the innermost $\Box$ introduction scheme. They permute instances of Eq past any of the inference schemes in $\{\supset\mathsf{I}, \supset\mathsf{E}, \Box\mathsf{E}, \mathsf{Eq}, \mathsf{TI}\}$. Eq just above the left hypothesis of an instance of $\Box\mathsf{I}$ is composed with the trail of the corresponding unit (cf. end of Section 3.2). In this sense, instances of $\Box\mathsf{I}$ determine the scope of the audited unit.

3.1 Principal contractions

There are three principal contractions (β, $\beta_\Box$ and TI-contraction), the first two of which rely on the substitution principles discussed earlier.

3.1.1 Case β

The first replaces a derivation of the form:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Delta;\Gamma_1,a{:}A;\Sigma_1 \vdash B\,|\,s}}{\Delta;\Gamma_1;\Sigma_1 \vdash A\supset B\,|\,\lambda a^A.s}\supset\mathsf{I} \qquad \dfrac{\pi_2}{\Delta;\Gamma_2;\Sigma_2 \vdash A\,|\,t}}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash B\,|\,(\lambda a^A.s)\cdot t}\supset\mathsf{E}$$

by the following, where π_3 is a derivation of $\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash B\,|\,s_t^a$ resulting from π_1 and π_2 and the Substitution Principle for Truth Hypothesis:

$$\dfrac{\pi_3\quad\dfrac{\dfrac{\pi_1}{\Delta;\Gamma_1,a{:}A;\Sigma_1\vdash B\,|\,s}\qquad\dfrac{\pi_2}{\Delta;\Gamma_2;\Sigma_2\vdash A\,|\,t}}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash \mathsf{Eq}(B,s_t^a,(\lambda a^A.s)\cdot t)\,|\,\mathfrak{ba}(a^A.s,t)}}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash B\,|\,(\lambda a^A.s)\cdot t}\,\mathsf{Eq}\,.$$

3.1.2 Case $\beta_\Box$

The second contraction replaces:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Delta;\cdot;\Sigma\vdash A\,|\,s}\qquad\dfrac{\pi_2}{\Delta;\cdot;\Sigma\vdash \mathsf{Eq}(A,s,t)\,|\,e_1}}{\Delta;\Gamma_1;\Sigma_1\vdash[\![\,\Sigma.t\,]\!]A\,|\,\Sigma.t}\,\Box\mathsf{I}\qquad\dfrac{\pi_3}{\Delta,u{:}A[\Sigma];\Gamma_2;\Sigma_2\vdash C\,|\,r}}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash C^u_{\Sigma.t}\,|\,\mathrm{LET}(u^{A[\Sigma]}.r,\Sigma.t)}\,\Box\mathsf{E}$$

with the following derivation where π_4 is a derivation of $\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash C^u_{\Sigma.t}\,|\,t^u_{\Sigma.t}$ resulting from π_1, π_2, π_3 and the Substitution Principle for Validity Hypothesis followed by weakening (of Γ_1 and Σ_1):

$$\dfrac{\pi\quad\dfrac{\dfrac{\pi_1}{\Delta;\cdot;\Sigma\vdash A\,|\,s}\qquad\dfrac{\pi_2}{\Delta;\cdot;\Sigma\vdash \mathsf{Eq}(A,s,t)\,|\,e_1}\qquad\dfrac{\pi_3}{\Delta,u{:}A[\Sigma];\Gamma_2;\Sigma_2\vdash C\,|\,r}}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash \mathsf{Eq}(C^u_{\Sigma.t},r^u_{\Sigma.t},\mathrm{LET}(u^{A[\Sigma]}.r,\Sigma.t))\,|\,\mathfrak{bb}(u^{A[\Sigma]}.r,\Sigma.t)}\,\mathsf{Eq}\beta_\Box}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash C^u_{\Sigma.t}\,|\,\mathrm{LET}(u^{A[\Sigma]}.r,\Sigma.t)}\,\mathsf{Eq}\,.$$

3.1.3 Case TI

TI-contraction models audit trail inspection. Consider the following derivation, where $\Sigma_1\subseteq\Sigma_2$, $\Delta'\subseteq\Delta$ and the branch from the depicted instance of TI in π_1 to its conclusion has no instances of $\Box$I:

$$\dfrac{\dfrac{\alpha:\mathsf{Eq}(A)\in\Sigma_1\qquad\dfrac{\vec{\pi}_2}{\Delta;\cdot;\cdot\vdash\mathcal{T}^B\,|\,\theta}}{\Delta;\Gamma;\Sigma_1\vdash B\,|\,\alpha\theta}\,\mathsf{TI}\\ \vdots\,\pi_1\\ \Delta';\cdot;\Sigma_2\vdash A\,|\,s\qquad\dfrac{\pi_3}{\Delta';\cdot;\Sigma_2\vdash\mathsf{Eq}(A,s,t)\,|\,e}}{\Delta';\Gamma';\Sigma'\vdash[\![\,\Sigma_2.t\,]\!]A\,|\,\Sigma_2.t}\,\Box\mathsf{I}\,.$$

The above depicted instance of TI in π_1 is replaced by the following derivation where π_4 is obtained from all the $\vec{\pi}_2$ and π_3 by resorting to Lemma 2.4 and weakening (Lemma 2.1).

Also, $\Delta;\cdot;\Sigma_2 \vdash \mathsf{Eq}(A,s,t)\,|\,e$ is obtained from $\Delta';\cdot;\Sigma_2 \vdash \mathsf{Eq}(A,s,t)\,|\,e$ by weakening (Lemma 2.1).

$$\dfrac{\dfrac{\pi_4}{\Delta;\Gamma;\Sigma_1 \vdash B\,|\,e\theta} \qquad \dfrac{\Delta;\cdot;\Sigma_2 \vdash \mathsf{Eq}(A,s,t)\,|\,e \qquad \dfrac{\vec{\pi}_2}{\Delta;\cdot;\cdot \vdash \mathcal{T}^B\,|\,\theta}\,\mathsf{EqTI}}{\Delta;\Gamma;\Sigma_1 \vdash \mathsf{Eq}(B,e\theta,\alpha\theta)\,|\,\mathrm{ti}(\theta,\alpha)}}{\Delta;\Gamma;\Sigma_1 \vdash B\,|\,\alpha\theta}\,\mathsf{Eq}.$$

3.2 Permutation conversions

As for the permutation conversions, they indicate how Eq is permuted past any of the inference schemes in $\{\supset\mathsf{I}, \supset\mathsf{E}, \Box\mathsf{E}, \mathsf{Eq}, \mathsf{TI}\}$.

3.2.1 Case $\supset\mathsf{I}$

In the first case, Eq permutes past $\supset\mathsf{I}$ by replacing:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Delta;\Gamma,a:A;\Sigma \vdash B\,|\,s} \qquad \dfrac{\pi_2}{\Delta;\Gamma,a:A;\Sigma \vdash \mathsf{Eq}(B,s,t)\,|\,e}\,\mathsf{Eq}}{\Delta;\Gamma,a:A;\Sigma \vdash B\,|\,t}}{\Delta;\Gamma;\Sigma \vdash A \supset B\,|\,\lambda a^A.t}\,\supset\mathsf{I}$$

with the following where π_3 is a derivation of $\Delta;\Gamma;\Sigma \vdash A \supset B\,|\,\lambda a^A.s$ obtained from π_1 and $\supset\mathsf{I}$:

$$\dfrac{\pi_3 \qquad \dfrac{\dfrac{\pi_2}{\Delta;\Gamma,a:A;\Sigma \vdash \mathsf{Eq}(B,s,t)\,|\,e}}{\Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A \supset B, \lambda a^A.s, \lambda a^A.t)\,|\,\mathfrak{abs}(a^A.e)}\,\mathsf{EqAbs}}{\Delta;\Gamma;\Sigma \vdash A \supset B\,|\,\lambda a^A.t}\,\mathsf{Eq}.$$

3.2.2 Case $\supset\mathsf{E}$

There are two permutation conversions associated with the $\supset\mathsf{E}$ inference scheme depending on whether the instance of Eq is the last scheme in the left or the right hypothesis. In the former, the permutation conversion consists in replacing:

$$\dfrac{\dfrac{\dfrac{\Delta;\Gamma_1;\Sigma_1 \vdash A_1 \supset A_2\,|\,s}{\Delta;\Gamma_1;\Sigma_1 \vdash \mathsf{Eq}(A_1 \supset A_2,s,t)\,|\,e}\,\mathsf{Eq}}{\Delta;\Gamma_1;\Sigma_1 \vdash A_1 \supset A_2\,|\,t} \qquad \Delta;\Gamma_2;\Sigma_2 \vdash A_1\,|\,r}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash A_2\,|\,t\cdot r}\,\supset\mathsf{E}$$

with the derivation:

$$\dfrac{\dfrac{\Delta;\Gamma_1;\Sigma_1 \vdash A_1 \supset A_2\,|\,s \quad \Delta;\Gamma_2;\Sigma_2 \vdash A_1\,|\,r}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash A_2\,|\,s\cdot r}\,\supset\mathsf{E} \qquad \dfrac{\pi_1}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash \mathsf{Eq}(A_2,s\cdot r,t\cdot r)\,|\,\mathfrak{app}(e,\mathfrak{r}(r))}\,\mathsf{EqApp}}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash A_2\,|\,t\cdot r}\,\mathsf{Eq},$$

where π_1 is:

$$\dfrac{\Delta;\Gamma_1;\Sigma_1\vdash \mathsf{Eq}(A_1\supset A_2,s,t)\,|\,e \qquad \dfrac{\Delta;\Gamma_2;\Sigma_2\vdash A_1\,|\,r}{\Delta;\Gamma_2;\Sigma_2\vdash \mathsf{Eq}(A_1,r,r)\,|\,\mathfrak{r}(r)}\,\mathsf{EqRefl}}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash \mathsf{Eq}(A_2,s\cdot r,t\cdot r)\,|\,\mathfrak{app}(e,\mathfrak{r}(r))}\,\mathsf{EqApp}.$$

In the latter the permutation conversion consists in replacing:

$$\dfrac{\Delta;\Gamma_1;\Sigma_1\vdash A_1\supset A_2\,|\,r \qquad \dfrac{\Delta;\Gamma_2;\Sigma_2\vdash A_1\,|\,s \quad \Delta;\Gamma_2;\Sigma_2\vdash \mathsf{Eq}(A_1,s,t)\,|\,e}{\Delta;\Gamma_2;\Sigma_2\vdash A_1\,|\,t}\,\mathsf{Eq}}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash A_2\,|\,r\cdot t}\,\supset\mathsf{E}$$

with the derivation:

$$\dfrac{\dfrac{\Delta;\Gamma_1;\Sigma_1\vdash A_1\supset A_2\,|\,r \quad \Delta;\Gamma_2;\Sigma_2\vdash A_1\,|\,s}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash A_2\,|\,r\cdot s}\,\supset\mathsf{E} \qquad \dfrac{\pi_1}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash \mathsf{Eq}(A_2,r\cdot s,r\cdot t)\,|\,\mathfrak{app}(\mathfrak{r}(r),e)}\,\mathsf{EqApp}}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash A_2\,|\,r\cdot t}\,\mathsf{Eq},$$

where π_1 is:

$$\dfrac{\dfrac{\Delta;\Gamma_1;\Sigma_1\vdash A_1\supset A_2\,|\,r}{\Delta;\Gamma_1;\Sigma_1\vdash \mathsf{Eq}(A_1\supset A_2,r,r)\,|\,\mathfrak{r}(r)}\,\mathsf{EqRefl} \qquad \Delta;\Gamma_2;\Sigma_2\vdash \mathsf{Eq}(A_1,s,t)\,|\,e}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash \mathsf{Eq}(A_2,r\cdot s,r\cdot t)\,|\,\mathfrak{app}(\mathfrak{r}(r),e)}\,\mathsf{EqApp}.$$

3.2.3 Case $\Box\mathsf{E}$

For the same reasons that there are two permutation conversions associated with $\supset\mathsf{E}$, there are also two associated with $\Box\mathsf{E}$. The first consists in replacing:

$$\dfrac{\dfrac{\Delta;\Gamma_1;\Sigma_1\vdash [\![\,\Sigma.s_1\,]\!]A\,|\,s_2}{\Delta;\Gamma_1;\Sigma_1\vdash \mathsf{Eq}([\![\,\Sigma.s_1\,]\!]A,s_2,r)\,|\,e}\,\mathsf{Eq} \qquad \Delta,u{:}A[\Sigma];\Gamma_2;\Sigma_2\vdash C\,|\,t}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash C^u_{\Sigma.s_1}\,|\,\mathrm{LET}(u^{A[\Sigma]}.t,r)}\,\Box\mathsf{E}$$

with:

$$\dfrac{\dfrac{\pi_1}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash C^u_{\Sigma.s_1}\,|\,q}\,\Box\mathsf{E} \qquad \dfrac{\pi_2}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash \mathsf{Eq}(C^u_{\Sigma.s_1},q,q')\,|\,e'}\,\mathsf{EqLet}}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash C^u_{\Sigma.s_1}\,|\,\mathrm{LET}(u^{A[\Sigma]}.t,r)}\,\mathsf{Eq},$$

where q is the proof code $\mathrm{LET}(u^{A[\Sigma]}.t,s_2)$ and π_1 is:

$$\dfrac{\Delta;\Gamma_1;\Sigma_1\vdash [\![\,\Sigma.s_1\,]\!]A\,|\,s_2 \quad \Delta,u{:}A[\Sigma];\Gamma_2;\Sigma_2\vdash C\,|\,t}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash C^u_{\Sigma.s_1}\,|\,\mathrm{LET}(u^{A[\Sigma]}.t,s_2)}\,\Box\mathsf{E}$$

and π_2 is the following derivation where q' is $\textsc{let}(u^{A[\Sigma]}.t, r)$, $e' \triangleq \mathfrak{let}(u^{A[\Sigma]}.\mathfrak{r}(t), e)$:

$$\dfrac{\Delta;\Gamma_1;\Sigma_1 \vdash \mathsf{Eq}([\![\,\Sigma.s_1\,]\!]A, s_2, r) \mid e \qquad \dfrac{\Delta, u{:}A[\Sigma];\Gamma_2;\Sigma_2 \vdash C \mid t}{\Delta, u{:}A[\Sigma];\Gamma_2;\Sigma_2 \vdash \mathsf{Eq}(C,t,t) \mid \mathfrak{r}(t)}\,\mathsf{EqRefl}}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash \mathsf{Eq}(C^u_{\Sigma.s_1}, q, q') \mid e'}\,\mathsf{EqLet}.$$

The second permutation conversion associated with $\Box\mathsf{E}$ consists in replacing:

$$\dfrac{\Delta;\Gamma_1;\Sigma_1 \vdash [\![\,\Sigma.r\,]\!]A \mid s \qquad \dfrac{\Delta, u{:}A[\Sigma];\Gamma_2;\Sigma_2 \vdash C \mid t_1 \qquad \Delta, u{:}A[\Sigma];\Gamma_2;\Sigma_2 \vdash \mathsf{Eq}(C,t_1,t_2) \mid e}{\Delta, u{:}A[\Sigma];\Gamma_2;\Sigma_2 \vdash C \mid t_2}\,\mathsf{Eq}}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash C^u_{\Sigma.r} \mid \textsc{let}(u^{A[\Sigma]}.t_2, s)}\,\Box\mathsf{E}$$

with the derivation:

$$\dfrac{\dfrac{\Delta;\Gamma_1;\Sigma_1 \vdash [\![\,\Sigma.r\,]\!]A \mid s \qquad \Delta, u{:}A[\Sigma];\Gamma_2;\Sigma_2 \vdash C \mid t_1}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash C^u_{\Sigma.r} \mid \textsc{let}(u^{A[\Sigma]}.t_1, s)}\,\Box\mathsf{E} \qquad \pi_1}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash C^u_{\Sigma.r} \mid \textsc{let}(u^{A[\Sigma]}.t_2, s)}\,\mathsf{Eq},$$

where, π_1 is:

$$\dfrac{\pi_2 \qquad \Delta, u{:}A[\Sigma];\Gamma_2;\Sigma_2 \vdash \mathsf{Eq}(C, t_1, t_2) \mid e}{\Delta;\Gamma_{1,2};\Sigma_{1,2} \vdash \mathsf{Eq}(C^u_{\Sigma.r}, q, q') \mid \mathfrak{let}(\mathfrak{r}(s), u^{A[\Sigma]}.e)}\,\mathsf{EqLet},$$

where $q \triangleq \textsc{let}(u^{A[\Sigma]}.t_1, s)$ and $q' \triangleq \textsc{let}(u^{A[\Sigma]}.t_2, s)$ and π_2 is:

$$\dfrac{\Delta;\Gamma_1;\Sigma_1 \vdash [\![\,\Sigma.r\,]\!]A \mid s}{\Delta;\Gamma_1;\Sigma_1 \vdash \mathsf{Eq}([\![\,\Sigma.r\,]\!]A, s, s) \mid \mathfrak{r}(s)}\,\mathsf{EqRefl}.$$

3.2.4 Case Eq

In this case the derivation:

$$\dfrac{\dfrac{\Delta;\Gamma;\Sigma \vdash A \mid r \qquad \Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,r,s) \mid e_1}{\Delta;\Gamma;\Sigma \vdash A \mid s}\,\mathsf{Eq} \qquad \Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,s,t) \mid e_2}{\Delta;\Gamma;\Sigma \vdash A \mid t}\,\mathsf{Eq}$$

is replaced by:

$$\dfrac{\Delta;\Gamma;\Sigma \vdash A \mid r \qquad \dfrac{\Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,r,s) \mid e_1 \qquad \Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,s,t) \mid e_2}{\Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(A,r,t) \mid \mathfrak{t}(e_1,e_2)}\,\mathsf{EqTrans}}{\Delta;\Gamma;\Sigma \vdash A \mid t}\,\mathsf{Eq}.$$

3.2.5 Case TI

This case is depicted as follows where c is some fixed trail constructor:

$$\dfrac{\alpha : \mathsf{Eq}(A) \in \Sigma \qquad \dfrac{\Delta;\cdot;\cdot \vdash \mathcal{T}^B(c) \mid r \qquad \dfrac{\Delta;\cdot;\cdot \vdash \mathsf{Eq}(\mathcal{T}^B(c), r, \theta(c)) \mid e}{\Delta;\cdot;\cdot \vdash \mathcal{T}^B(c) \mid \theta(c)}\,\mathsf{Eq}}{\Delta;\cdot;\cdot \vdash \mathcal{T}^B(c) \mid \theta(c)} \qquad \Delta;\cdot;\cdot \vdash \mathcal{T}^B(d) \mid \theta(d) \quad d \in \mathfrak{C} \setminus c}{\Delta;\Gamma;\Sigma \vdash B \mid \alpha\theta}\,\mathsf{TI}$$

is replaced by:

$$\dfrac{\dfrac{\alpha : \mathsf{Eq}(A) \in \Sigma \quad \Delta;\cdot;\cdot \vdash \mathcal{T}^B \mid \theta'}{\Delta;\Gamma;\Sigma \vdash B \mid \alpha\theta'}\,\mathsf{TI} \qquad \dfrac{\alpha : \mathsf{Eq}(A) \in \Sigma \quad \Delta;\cdot;\cdot \vdash \mathsf{Eq}(\mathcal{T}^B(c), \theta'(c), \theta(c)) \mid e_c \quad \forall c \in \mathfrak{C}}{\Delta;\Gamma;\Sigma \vdash \mathsf{Eq}(B, \alpha\theta', \alpha\theta) \mid \mathfrak{trpl}(\overline{e})}\,\mathsf{EqRpl}}{\Delta;\Gamma;\Sigma \vdash B \mid \alpha\theta}\,\mathsf{Eq}\,,$$

where

- $\theta'(c) \triangleq r$ and $\theta'(d) \triangleq \theta(d)$, for all $d \in \mathfrak{C}$ s.t. $d \neq c$; and
- $e_c \triangleq e$ and $e_d \triangleq \mathfrak{r}(\theta(d))$, for all $d \neq c$.

3.2.6 Case Eq - $\Box\mathsf{I}$

Finally, the conversion that fuses an occurrence of Eq just above the left hypothesis in an instance of $\Box\mathsf{I}$ with the trail of the corresponding unit is also coined, by abuse of language, permutation conversion. It replaces:

$$\dfrac{\dfrac{\Delta;\cdot;\Sigma \vdash A \mid r \quad \Delta;\cdot;\Sigma \vdash \mathsf{Eq}(A, r, s) \mid e_1}{\Delta;\cdot;\Sigma \vdash A \mid s}\,\mathsf{Eq} \qquad \Delta;\cdot;\Sigma \vdash \mathsf{Eq}(A, s, t) \mid e_2}{\Delta;\Gamma;\Sigma' \vdash [\![\, \Sigma.t \,]\!] A \mid \Sigma.t}\,\Box\mathsf{I}$$

with:

$$\dfrac{\Delta;\cdot;\Sigma \vdash A \mid r \qquad \dfrac{\Delta;\cdot;\Sigma \vdash \mathsf{Eq}(A, r, s) \mid e_1 \quad \Delta;\cdot;\Sigma \vdash \mathsf{Eq}(A, s, t) \mid e_2}{\Delta;\cdot;\Sigma \vdash \mathsf{Eq}(A, r, t) \mid \mathfrak{t}(e_1, e_2)}\,\mathsf{EqTrans}}{\Delta;\Gamma;\Sigma' \vdash [\![\, \Sigma.t \,]\!] A \mid \Sigma.t}\,\Box\mathsf{I}$$

4 Term assignment

Computation by normalization is non-confluent, as one might expect (audit trail inspection affects computation), hence a strategy is required. This section introduces the call-by-value $\lambda^{\mathfrak{h}}$-calculus. It

$$\frac{a : A \in \Gamma}{\Delta; \Gamma; \Sigma \vdash a : A \mid a}\ \mathsf{TVar} \qquad \frac{\Delta; \Gamma, a : A; \Sigma \vdash M : B \mid s}{\Delta; \Gamma; \Sigma \vdash \lambda a^A.M : A \supset B \mid \lambda a^A.s}\ \mathsf{TAbs}$$

$$\frac{\Delta; \Gamma_1; \Sigma_1 \vdash M : A \supset B \mid s \quad \Delta; \Gamma_2; \Sigma_2 \vdash N : A \mid t}{\Delta; \Gamma_{1,2}; \Sigma_{1,2} \vdash M\,N : B \mid s \cdot t}\ \mathsf{TApp} \qquad \frac{u : A[\Sigma] \in \Delta \quad \Sigma\sigma \subseteq \Sigma'}{\Delta; \cdot; \Sigma' \vdash \langle u; \sigma \rangle : A \mid \langle u; \sigma \rangle}\ \mathsf{TmVar}$$

$$\frac{\Delta; \cdot; \Sigma \vdash M : A \mid s \quad \Delta; \cdot; \Sigma \vdash \mathsf{Eq}(A, s, t) \mid e}{\Delta; \Gamma; \Sigma' \vdash !^{\Sigma}_{\overline{e}} M : [\![\Sigma.t]\!]A \mid \Sigma.t}\ \mathsf{TBox} \qquad \frac{\Delta; \Gamma_1; \Sigma_1 \vdash M : [\![\Sigma.r]\!]A \mid s \quad \Delta, u : A[\Sigma]; \Gamma_2; \Sigma_2 \vdash N : C \mid t}{\Delta; \Gamma_{1,2}; \Sigma_{1,2} \vdash \begin{array}{c} \mathrm{LET}\, u^{A[\Sigma]}\, \mathrm{BE}\, M\, \mathrm{IN}\, N : C^{u}_{\Sigma.r} \\ \mathrm{LET}(u^{A[\Sigma]}.t, s) \end{array} \mid}\ \mathsf{TLet}$$

$$\frac{\alpha : \mathsf{Eq}(A) \in \Sigma \quad \Delta; \cdot; \cdot \vdash \vartheta : \mathcal{T}^B \mid \theta}{\Delta; \Gamma; \Sigma \vdash \alpha\vartheta : B \mid \alpha\theta}\ \mathsf{TTI} \qquad \frac{\Delta; \Gamma; \Sigma \vdash M : A \mid s \quad \Delta; \Gamma; \Sigma \vdash \mathsf{Eq}(A, s, t) \mid e}{\Delta; \Gamma; \Sigma \vdash \overline{e \triangleright} M : A \mid t}\ \mathsf{TEq}$$

FIGURE 4. Typing schemes for $\lambda^{\mathfrak{h}}$.

is obtained via a term assignment for $\mathbf{JL}^{\mathfrak{h}}$. Recall from Section 2 that ϑ in TTI is a mapping from trail constructors to terms. The syntax of $\lambda^{\mathfrak{h}}$ terms is:

$M ::=$	a	term variable
	$\lambda a^A.M$	abstraction
	$M\,M$	application
	$\langle u; \sigma \rangle$	audited unit variables
	$!^{\Sigma}_{\overline{e}} M$	audited unit
	$\mathrm{LET}\, u^{A[\Sigma]}\, \mathrm{BE}\, M\, \mathrm{IN}\, M$	audited unit composition
	$\alpha\vartheta$	trail inspection
	$\overline{e \triangleright} M$	derived term.

The greyed expressions are part of the run-time syntax and hence not available to the programmer. We occasionally drop the type decoration, $A[\Sigma]$ above, in audited unit composition for readability. The term constructors for audited units and trail inspection have already been encountered in the introduction. The term $\langle u; \sigma \rangle$ is an *audited unit variable* which, when replaced by an audited unit, shall be subject to the substitution σ (which we recall is a trail variable renaming). The term $\mathrm{LET}\, u^{A[\Sigma]}\, \mathrm{BE}\, M\, \mathrm{IN}\, N$ is called *audited unit composition*; we refer to M as its *argument* and N its *body*. The expression $\overline{e \triangleright} M$ denotes a term M with suspended trail e; this trail will be permuted using permutative conversions to the innermost enclosing audited unit, as shall be made clear from the operational semantics.

4.1 *Typing schemes and operational semantics*

The typing judgement $\Delta; \Gamma; \Sigma \vdash M : A \mid s$ is defined by means of *typing schemes*. These are obtained by decorating the inference schemes of Figure 2 with terms. A term M is said to be *typable* if there exists $\Delta, \Gamma, \Sigma, A, s$ s.t. $\Delta; \Gamma; \Sigma \vdash M : A \mid s$ is derivable. The typing schemes are presented in Figure 4. We often resort to the instance of TBox in which $s = t$ and $e = \mathfrak{r}(s)$; in this case we omit e altogether and write $!^{\Sigma} M$. Also, note that the scheme TEq incorporates the trail e into the term assignment.

The operational semantics of $\lambda^{\mathfrak{h}}$ is specified by a binary relation of *typed* terms called *reduction*. This requires two substitution operations: one for substituting term (truth) variables and one for audited unit (validity) variables. Since terms may be decorated with trails, substitution (both for truth and validity hypothesis) replaces free occurrences of variables with both terms and evidence.

DEFINITION 4.1 (Substitution of truth and validity variables)
We write $M^a_{N,t}$ for substitution of truth variables and $M^u_{\Sigma.(N,t,e)}$ for substitution of validity variables (similar notions apply to substitution in propositions, proof code and trails). Note that '$\Sigma.$' in $\Sigma.(N,t,e)$ binds all free occurrences of trail variables from Σ which occur in N, t and e.

$$
\begin{array}{rcl}
a^a_{N,s} & \triangleq & N \\
b^a_{N,s} & \triangleq & b \\
(\lambda b^A.M)^a_{N,s} & \triangleq & \lambda b^A.M^a_{N,s} \\
(PQ)^a_{N,s} & \triangleq & P^a_{N,s}Q^a_{N,s} \\
\langle u;\sigma\rangle^a_{N,s} & \triangleq & \langle u;\sigma\rangle \\
(!^{\Sigma}_{e}M)^a_{N,s} & \triangleq & !^{\Sigma}_{e}M \\
(\mathrm{LET}\,u\,\mathrm{BE}\,M_1\,\mathrm{IN}\,M_2)^a_{N,s} & \triangleq & \mathrm{LET}\,u\,\mathrm{BE}\,M_1{}^a_{N,s}\,\mathrm{IN}\,M_2{}^a_{N,s} \\
(\alpha\theta)^a_{N,s} & \triangleq & \alpha\theta \\
(e\rhd M)^a_{N,s} & \triangleq & e^a_s\rhd M^a_{N,s}
\end{array}
\qquad
\begin{array}{rcl}
b^u_{\Sigma.(N,t,e)} & \triangleq & b \\
(\lambda b^A.M)^u_{\Sigma.(N,t,e)} & \triangleq & \lambda b^A.M^u_{\Sigma.(N,t,e)} \\
(PQ)^u_{\Sigma.(N,t,e)} & \triangleq & P^u_{\Sigma.(N,t,e)}Q^u_{\Sigma.(N,t,e)} \\
\langle u;\sigma\rangle^u_{\Sigma.(N,t,e)} & \triangleq & e\sigma\rhd N\sigma \\
\langle v;\sigma\rangle^u_{\Sigma.(N,t,e)} & \triangleq & \langle v;\sigma\rangle \\
(!^{\Sigma'}_{e'}M)^u_{\Sigma.(N,t,e)} & \triangleq & !^{\Sigma'}_{e'^u_{\Sigma.t}}M^u_{\Sigma.(N,t,e)} \\
(\mathrm{LET}\,v\,\mathrm{BE}\,P\ \mathrm{IN}\,Q)^u_{\Sigma.(N,t,e)} & \triangleq & \mathrm{LET}\,v\,\mathrm{BE}\,P^u_{\Sigma.(N,t,e)}\ \mathrm{IN}\,Q^u_{\Sigma.(N,t,e)} \\
(\alpha\vartheta)^u_{\Sigma.(N,t,e)} & \triangleq & \alpha\vartheta^u_{\Sigma.(N,t,e)} \\
(e'\rhd M)^u_{\Sigma.(N,t,e)} & \triangleq & e'^u_{\Sigma.t}\rhd M^u_{\Sigma.(N,t,e)}
\end{array}
.
$$

These definitions, in turn, rely on: (i) $s^u_{\Sigma.t}$ which traverses the structure of s replacing $\langle u;\sigma\rangle^u_{\Sigma.s}$ with $s\sigma$ and (ii) $e^u_{\Sigma.t}$ which traverses the structure of e until it reaches one of $\mathfrak{r}(r_1)$, $\mathfrak{ba}(a^A.r_1,r_2)$ or $\mathfrak{bb}(v^{A[\Sigma']}.r_1,\Sigma'.r_2)$ in which case it resorts to substitution over the r_is. Of particular interest is the fourth defining clause of the definition of $M^u_{\Sigma.(N,t,e)}$. Note how it substitutes $\langle u;\sigma\rangle$ with $e\sigma\rhd N\sigma$, thus (i) propagating the history of N and (ii) renaming the trail variables of N so that they make sense in the new host unit.

We say that a term M reduces to a term N if the *reduction* judgement $\Delta;\Gamma;\Sigma\vdash M\to N:A\,|\,s$ is derivable. In most cases we write $M\to N$ for the sake of readability. In order to define reduction we first introduce two intermediate notions, namely *principal reduction* ($\mapsto$) and *permutation reduction* ($\rightsquigarrow$). The former corresponds to principal contraction and the latter to permutation conversions of the normalization procedure. The set of *values* are as follows:

$$V ::= a\,|\,\langle u;\sigma\rangle\,|\,\lambda a^A.M\,|\,!^{\Sigma}_{e}V\,.$$

Note that $!^{\Sigma}_{e}V$ is a value only if the body is value. *Evaluation contexts* are represented with letters $\mathcal{E},\mathcal{E}'$, etc:

$$
\begin{array}{rcl}
\mathcal{E} & ::= & \Box\,|\,\mathcal{E}M\,|\,V\mathcal{E} \\
& | & \mathrm{LET}\,u^{A[\Sigma]}\,\mathrm{BE}\,\mathcal{E}\,\mathrm{IN}\,M\,|\,!^{\Sigma}_{e}\mathcal{E} \\
& | & \alpha\{c_1/V,\ldots,c_j/V,c_{j+1}/\mathcal{E},\ldots\}
\end{array}
\quad .
$$

DEFINITION 4.2 (Principal Reduction)
Principal reduction ($\mapsto$) is defined by the following reduction rule:

$$\frac{M\rightharpoonup N}{\mathcal{E}[M]\mapsto\mathcal{E}[N]}\,,$$

where $\rightharpoonup$ is the union of the following binary relations:

$$\frac{\Delta;\Gamma_1,a{:}A;\Sigma_1\vdash M:B\,|\,s \quad \Delta;\Gamma_2;\Sigma_2\vdash V:A\,|\,t}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash(\lambda a^A.M)\,V\rightharpoonup \mathfrak{ba}(a^A.s,t)\rhd M^a_{V,t}:B\,|\,(\lambda a^A.s)\cdot t}\,\beta^V$$

$$\frac{\Delta;\cdot;\Sigma\vdash V:A\,|\,r \quad \Delta;\cdot;\Sigma\vdash \mathsf{Eq}(A,r,s)\,|\,e_1 \quad \Delta,u{:}A[\Sigma];\Gamma_2;\Sigma_2\vdash N:C\,|\,t}{\Delta;\Gamma_{1,2};\Sigma_{1,2}\vdash O\rightharpoonup P:C^u_{\Sigma.s}\,|\,\mathrm{LET}(u^{A[\Sigma]}.t,\Sigma.s)}\,\beta^V_\Box\quad,$$

$$\frac{\alpha:\mathsf{Eq}(A)\in\Sigma \quad \Delta';\cdot;\cdot\vdash\vartheta:\mathcal{T}^B\,|\,\theta \quad \Delta\subseteq\Delta' \quad \Delta;\cdot;\Sigma\vdash\mathcal{F}[\alpha\vartheta]:A\,|\,r \quad \Delta;\cdot;\Sigma\vdash\mathsf{Eq}(A,r,s)\,|\,e}{\Delta;\Gamma;\Sigma'\vdash!^\Sigma_e\mathcal{F}[\alpha\vartheta]\rightharpoonup!^\Sigma_e\mathcal{F}[\,\mathfrak{ti}(\theta,\alpha)\rhd e\vartheta\,]:[\![\,\Sigma.s\,]\!]A\,|\,\Sigma.s}\,\mathcal{I}^V$$

where, in $\beta^V_\Box$, $O\triangleq\mathrm{LET}u^{A[\Sigma]}\mathrm{BE}!^\Sigma_e V\,\mathrm{IN}N$ and $P\triangleq\mathfrak{bb}(u^{A[\Sigma]}.t,\Sigma.s)\rhd N^u_{\Sigma.(V,s,e)}$. In the rule $\mathcal{I}^V$, the auxiliary contexts $\mathcal{F}$, described below, are defined as:

$$\begin{aligned}\mathcal{F}\ ::=\ &\Box\,|\,\mathcal{F}M\,|\,V\mathcal{F}\\ |\ &\mathrm{LET}u^{A[\Sigma]}\mathrm{BE}\mathcal{F}\,\mathrm{IN}M\\ |\ &\alpha\{c_1/V,\ldots,c_j/V,c_{j+1}/\mathcal{F},\ldots\}\end{aligned}\quad.$$

Moreover, the hole in $\mathcal{F}$ has type B. Finally, trail replacement ϑ in $\mathcal{I}^V$ is of the form $\{c_1/V,\ldots,c_{10}/V\}$, where c_i, $i\in 1..10$, stand for each of the trail constructors in $\mathfrak{C}$.

Comments on principal reduction follow. Note that reduction under the audited unit constructor is allowed. Contexts $\mathcal{F}$ differ from $\mathcal{E}$ by not allowing holes under the audited unit constructor. Each of the three principal reduction rules produce a trail of its execution. In accordance with our discussion in the introduction, $\beta^V_\Box$ replaces all occurrences of $\langle u;\sigma\rangle$ with $e\sigma\rhd V\sigma$, thereby correctly: (i) preserving trails and (ii) renaming trail variables so that they now refer to their host audited computation unit.

Regarding permutation reduction, the original rules obtained from the normalization procedure are the contextual closure of the first group of rules depicted in Figure 5. Type decorations in compatibility codes as well as the typing information of each of the subterms have been omitted for the sake of readability. Also, the trail replacement ϑ in the last rule is $\{c_1/M_1,\ldots,c_i/M_i,\ldots,c_{10}/M_{10}\}$. These rules are easily proven to be terminating. However, they are not confluent. As an example, consider the term $(e_1\rhd M)(e_2\rhd N)$. Application of the first rule yields $\mathfrak{app}(e_1,\mathfrak{r}(t_2))\rhd(M\ (e_2\rhd N))$ where t_2 is the proof code of the type derivation of $e_2\rhd N$ in $(e_1\rhd M)(e_2\rhd N)$. Likewise, application of the second yields $\mathfrak{app}(\mathfrak{r}(t_1),e_2)\rhd((e_1\rhd M)N)$ where t_1 is the proof code of the type derivation of $e_1\rhd M$ in $(e_1\rhd M)(e_2\rhd N)$. The reader may verify that these terms are not joinable. As a consequence we complete these rules with those in the second group depicted in Figure 5.

DEFINITION 4.3 (Permutation reduction)
Permutation reduction, $\rightsquigarrow$, is defined by the contextual closure of the reduction axioms of Figure 5. Contextual closure here means reduction is allowed inside any subexpression of a term, including trails.

PROPOSITION 4.1
$\rightsquigarrow$ is terminating and confluent.

Termination may be proved automatically by using AProVE [16]. Confluence follows by checking local confluence and resorting to Newman's Lemma.

$$
\begin{array}{rcl}
(e \rhd M)\ N & \rightsquigarrow & \mathtt{app}(e, \mathfrak{r}(t)) \rhd (M\ N) \\
M\ (e \rhd N) & \rightsquigarrow & \mathtt{app}(\mathfrak{r}(t), e) \rhd (M\ N) \\
\lambda a^A.(e \rhd M) & \rightsquigarrow & \mathtt{abs}(a.e) \rhd (\lambda a^A.M) \\
\text{LET } u \text{ BE } (e \rhd M) \text{ IN } N & \rightsquigarrow & \mathtt{let}(u.e, \mathfrak{r}(t)) \rhd (\text{LET } u \text{ BE } M \text{ IN } N) \\
\text{LET } u \text{ BE } M \text{ IN } (e \rhd N) & \rightsquigarrow & \mathtt{let}(u.\mathfrak{r}(s), e) \rhd (\text{LET } u \text{ BE } M \text{ IN } N) \\
!^{\Sigma}_{e_2}(e_1 \rhd M) & \rightsquigarrow & !^{\Sigma}_{\mathfrak{t}(e_1, e_2)} M \\
e_1 \rhd (e_2 \rhd M) & \rightsquigarrow & \mathfrak{t}(e_1, e_2) \rhd M \\
\alpha\{c_1/M_1, \ldots, c_i/e \rhd M_i, \ldots\} & \rightsquigarrow & \mathtt{trpl}(\mathfrak{r}(t_1), \ldots, \mathfrak{r}(t_{i-1}), e, \ldots, \mathfrak{r}(t_{10})) \rhd \alpha\vartheta
\end{array}
$$

$$
\begin{array}{rcl}
\mathfrak{t}(\mathtt{app}(e_1, e_2), \mathtt{app}(e_3, e_4)) & \rightsquigarrow & \mathtt{app}(\mathfrak{t}(e_1, e_3), \mathfrak{t}(e_2, e_4)) \\
\mathfrak{t}(\mathtt{abs}(a.e_1), \mathtt{abs}(a.e_2)) & \rightsquigarrow & \mathtt{abs}(a.\mathfrak{t}(e_1, e_2)) \\
\mathfrak{t}(\mathtt{let}(u.e_1, e_2), \mathtt{let}(u.e_3, e_4)) & \rightsquigarrow & \mathtt{let}(u.\mathfrak{t}(e_1, e_3), \mathfrak{t}(e_2, e_4)) \\
\mathfrak{t}(\mathtt{trpl}(\vec{e_1}), \mathtt{trpl}(\vec{e_2})) & \rightsquigarrow & \mathtt{trpl}(\overrightarrow{\mathfrak{t}(e_1, e_2)}) \\
\mathfrak{t}(\mathfrak{r}(s), e) & \rightsquigarrow & e \\
\mathfrak{t}(e, \mathfrak{r}(t)) & \rightsquigarrow & e \\
\mathfrak{t}(\mathfrak{t}(e_1, e_2), e_3) & \rightsquigarrow & \mathfrak{t}(e_1, \mathfrak{t}(e_2, e_3)) \\
\mathfrak{t}(\mathtt{app}(e_1, e_2), \mathfrak{t}(\mathtt{app}(e_3, e_4), e_5)) & \rightsquigarrow & \mathfrak{t}(\mathtt{app}(\mathfrak{t}(e_1, e_3), \mathfrak{t}(e_2, e_4)), e_5) \\
\mathfrak{t}(\mathtt{abs}(a.e_1), \mathfrak{t}(\mathtt{abs}(a.e_2), e_3)) & \rightsquigarrow & \mathfrak{t}(\mathtt{abs}(a.\mathfrak{t}(e_1, e_2)), e_3) \\
\mathfrak{t}(\mathtt{let}(u.e_1, e_2), \mathfrak{t}(\mathtt{let}(u.e_3, e_4), e_5)) & \rightsquigarrow & \mathfrak{t}(\mathtt{let}(u.\mathfrak{t}(e_1, e_3), \mathfrak{t}(e_2, e_4)), e_5) \\
\mathfrak{t}(\mathtt{trpl}(\vec{e_1}), \mathfrak{t}(\mathtt{trpl}(\vec{e_2}), e_3)) & \rightsquigarrow & \mathfrak{t}(\mathtt{trpl}(\overrightarrow{\mathfrak{t}(e_1, e_2)}), e_3)
\end{array}
$$

FIGURE 5. Permutation reduction rules.

REMARK 3
The fact that these reduction rules are defined over typed terms is crucial for confluence. Indeed, the system is not confluent over the set of untyped terms. For example, $\mathfrak{t}(\mathfrak{r}(s), \mathfrak{r}(t))$ reduces to both $\mathfrak{r}(s)$ and $\mathfrak{r}(t)$. However, in a typed setting $\mathfrak{t}(\mathfrak{r}(s), \mathfrak{r}(t))$ typable implies $s = t$.

DEFINITION 4.4 (Reduction)
Let $\rightarrowtail$ stand for permutation reduction to (the unique) normal form. Reduction ($\rightarrow$) is defined over terms in permutation-reduction normal form as $\mapsto \circ \rightarrowtail$, where $\circ$ denotes relation composition.

We summarize the relations introduced in this section:

$\mapsto$	principal reduction
$\rightsquigarrow$	permutation reduction
$\rightarrowtail$	permutation reduction to normal form
$\rightarrow$	reduction

4.2 Access-control example revisited

We now revisit our example, this time placing the focus on the following two salient features of $\lambda^{\mathfrak{h}}$: trail update and substitution of audited units.

Trail update. Consider the modification of the right-hand side of (2) (from the introduction) that reads $!^{\alpha}_{\mathfrak{r}((\lambda a^{Str}.s_1)\cdot s_2)} M$, where M is:

$$\big(\lambda a^{Str}.\mathsf{if}\, \mathit{FileIOPerm} \in \alpha\vartheta\ \mathsf{then}\ \mathbf{Win32Delete}\, a\ \mathsf{else}\ \mathbf{securityException}\big)\text{“..}\backslash \mathit{passwd}''\,.$$

It differs in two ways w.r.t. the right-hand side of (2). First the body M of the audited unit is an application rather than an abstraction. Second, rather than using the shorthand $!^{\alpha}M$ (see beginning of Section 4.1) the full notation reflecting the implicit trail has been displayed, namely: $\mathfrak{r}((\lambda a^{Str}.s_1)\cdot s_2)$. This trail asserts that $(\lambda a^{Str}.s_1)\cdot s_2$ is the code of a typing derivation for M and should be considered compatible with itself; *Str* is the type of strings. It reflects that no computation has taken place since it does not have occurrences of trails of β-steps, as we now describe. A β-step of computation from $!^{\alpha}_{\mathfrak{r}((\lambda a^{Str}.s_1)\cdot s_2)}M$ produces:

$$!^{\alpha}_{\mathfrak{t}(\mathfrak{ba}(a^{Str}.s_1,s_2),\mathfrak{r}((\lambda a^{Str}.s_1)\cdot s_2))}\ \mathsf{if}\ \mathit{FileIOPerm}\in\alpha\vartheta\ \mathsf{then}\ \mathbf{Win32Delete}\ \text{“..}\backslash\mathit{passwd}''\cdot\ \mathsf{else}\ \mathbf{securityException} \tag{3}$$

The trail of the audited unit has been updated to reflect the reduction step that took place. In $\mathfrak{t}(\mathfrak{ba}(a^{Str}.s_1,s_2),\mathfrak{r}((\lambda a^{Str}.s_1)\cdot s_2))$, the expressions $a^{Str}.s_1,s_2$ and $(\lambda a^{Str}.s_1)\cdot s_2$ are encodings of typing derivations for $(\lambda a^{Str}.\mathsf{if}\ \mathit{FileIOPerm}\in\alpha\vartheta\ \mathsf{then}\ \mathbf{Win32Delete}\ a\ \mathsf{else}\ \mathbf{securityException})$, ‘..*passwd*’ and M, resp. The β-step $\mathfrak{ba}(a^{Str}.s_1,s_2)$ states that the typing derivation s_1 where all free occurrences of a have been replaced by s_2 should be considered compatible with $(\lambda a^{Str}.s_1)\cdot s_2$; the trail constructor $\mathfrak{t}$ asserts transitivity of compatibility.

This account of the process of updating the trail while computing is slightly abridged; it is actually developed in two stages as discussed above: a *principal contraction*, which produces the trail, followed by a number of *permutative conversions*, which transports the trail to its innermost enclosing audited unit constructor.

Substitution for audited units. Let us return to our initial example **eraseFile** (2). How do we supply the name of the file to delete? Clearly we cannot simply apply **eraseFile** to a string since it is not an abstraction (indeed, it is an audited unit). We require some means of extracting the value computed by an audited unit. This is accomplished by audited unit composition $\textsc{let}\,u^{A[\Sigma]}\,\textsc{be}\,M\,\textsc{in}\,N$. As seen above, it evaluates as follows, determining a $\beta_{\Box}$-step. First evaluate the argument M until a value $!^{\alpha_1,\ldots,\alpha_n}_{e}V$ is obtained (note the V under the modal term constructor). Then replace the free occurrences of u in the body N with V. In *addition*, the following actions are performed, as dictated by the proof theoretical analysis developed in Section 3:

(1) e is copied so that trails are correctly persisted; and
(2) all free occurrences of $\alpha_1,\ldots,\alpha_n$ in V are renamed with the trail variables of the (innermost) audited unit in which u resides.[4] Hence, the reason why trail variables u are accompanied by a trail variable renaming.

Returning to our example, consider the following top-level declarations:

$$\begin{array}{lll}\mathbf{eraseFile} & \doteq & !^{\alpha}\lambda a^{Str}.\mathsf{if}\ \mathit{FileIOPerm}\in\alpha\vartheta\ \mathsf{then}\ \mathbf{Win32Delete}\ a\ \mathsf{else}\ \mathbf{securityException};\\ \mathbf{cleanup} & \doteq & !^{\beta}\lambda a^{Str}.\mathbf{eraseFile}\ \beta\ a;\\ \mathbf{bad} & \doteq & !^{\gamma}\,\mathbf{cleanup}\ \gamma\ \text{“..}\backslash\mathit{passwd}'';\end{array}$$

where, as in the last two clauses above, we write $\mathbf{f}\,\vec{\beta}\,\vec{N}$ to abbreviate $\mathsf{let}\,u=\mathbf{f}\,\mathsf{in}\,\langle u;\vec{\alpha}/\vec{\beta}\rangle\vec{N}$ assuming that $\mathbf{f}\doteq!^{\vec{\alpha}}_{e}\lambda^{\vec{a}:\vec{A}}.M$. Reduction from the term $!^{\delta}\mathbf{bad}\ \delta$ will eventually produce:

$$!^{\delta}_{e}\,\mathsf{if}\ \mathit{FileIOPerm}\in\delta\vartheta\ \mathsf{then}\ \mathbf{Win32Delete}\text{“..}\backslash\mathit{passwd}''\ \mathsf{else}\ \mathbf{securityException}. \tag{4}$$

[4] These two items illustrate how the reduction behaviour of $\textsc{let}\,u^{A[\Sigma]}\,\textsc{be}\,M\,\textsc{in}\,N$ differs from the standard computational interpretation of some modal logics [14, 15, 24].

Trail e will include three instances of a $\mathfrak{bb}$ trail constructor reflecting three $\beta_{\Box}$-steps and the $\mathfrak{Def}_{\mathbf{eraseFile}}$, $\mathfrak{Def}_{\mathbf{cleanup}}$ and $\mathfrak{Def}_{\mathbf{bad}}$ trail constructors reflecting three top-level function unfoldings. Evaluation of the condition $\mathit{FileIOPerm} \in \delta\vartheta$ will thefore return false, given that $\vartheta(\mathfrak{Def}_{\mathbf{bad}}) \triangleq \{\mathit{perms}(\mathbf{bad})\} = \emptyset$, and a security exception is raised.

REMARK 4
Finally, a remark on trail variables. They are *affine* in the sense that at different points in time during the execution process, they denote different terms given that the computation trail varies. Therefore, they should a occur at most once in a program (this is not a limitation since as many trail variables as necessary are available). From a logical point of view, failure to treat variables as affine renders the equational theory inconsistent. This is witnessed by the scheme EqTI, by taking two different proof codes e_1 and e_2 (representing two different trails).

4.3 Safety

This section addresses SR and Progress. Progress states that if a typable and closed term is not yet a value, then the term is the source of a reduction step. Together, these two properties determine *safety* of reduction w.r.t. the type system. SR follows from the fact that the reduction rules originate from proof normalization. The exception are the second group of rules of Figure 5 for which type preservation may be proved separately. We therefore have:

PROPOSITION 4.2 (Subject Reduction)
$\Delta; \Gamma; \Sigma \vdash M : A \mid s$ and $M \to N$ implies $\Delta; \Gamma; \Sigma \vdash N : A \mid s$.

Before addressing Progress we introduce some auxiliary notions. A term is *inspection-blocked* if it is of the form $\mathcal{F}[\alpha\vartheta]$. A term M is *tv-closed* if $\mathsf{fvT}(M) = \mathsf{fvV}(M) = \emptyset$, where recall from above that $\mathsf{fvT}(M)$ is the set of free truth variables of M and $\mathsf{fvV}(M)$ the set of free validity variables of M. It is *closed* if it is tv-closed and $\mathsf{fvTrl}(M) = \emptyset$, $\mathsf{fvTrl}(M)$ being the set of free trail variables of M.

LEMMA 4.3 (Canonical forms)
Assume $\cdot; \cdot; \Sigma \vdash V : A \mid s$. Then,

(1) If $A = A_1 \supset A_2$, then $V = \lambda a^{A_1}.M$ for some a, M.
(2) If $A = [\![\, \Sigma'.t \,]\!] A_1$, then $V = !_e^{\Sigma'} V'$ for some e, V'.

PROPOSITION 4.4
Suppose M is in permutation reduction-normal form, is typable and tv-closed. Then (1) M is a value or; (2) there exists N s.t. $M \mapsto N$ or; (3) M is inspection-blocked.

Since a closed term cannot be inspection-blocked:

COROLLARY 4.5 (Progress)
Suppose M is in permutation reduction normal form, is typable and closed. Then either M is a value or there exists N s.t. $M \to N$.

5 Strong Normalization

We address SN for reduction. In fact, we shall prove SN for a restriction of reduction. The restriction consists in requiring that M in the principal reduction axiom $\beta_{\Box}^{V}$ not have occurrences of the audited

computation unit constructor '!'[5]. In order to develop our proof, we introduce the notion of *full reduction* which lifts the value restriction in β^V and $\beta^V_\Box$ of Definition 4.2; and allows more general evaluation contexts for all three axioms. The evaluation contexts are extended as follows:

$$\begin{array}{rl}\mathcal{D} ::= & \Box \mid \lambda a^A.\mathcal{D} \mid \mathcal{D}M \mid M\mathcal{D} \\ \mid & \text{LET}\, u^{A[\Sigma]}\, \text{BE}\, \mathcal{D}\, \text{IN}\, M \mid \text{LET}\, u^{A[\Sigma]}\, \text{BE}\, M\, \text{IN}\, \mathcal{D} \mid !^{\Sigma}_{e}\mathcal{D} \\ \mid & e \rhd \mathcal{D} \\ \mid & \alpha\{c_1/M, \ldots, c_j/M, c_{j+1}/\mathcal{D}, \ldots\}\end{array}.$$

DEFINITION 5.1 (Full Principal Reduction)
Full principal reduction ($\overset{f}{\mapsto}$) is defined by the following reduction scheme:

$$\frac{M \overset{f}{\rightharpoonup} N}{\mathcal{D}[M] \overset{f}{\mapsto} \mathcal{D}[N]},$$

where $\overset{f}{\rightharpoonup}$ is the union of the following binary relations:

$$\frac{\Delta; \Gamma_1, a{:}A; \Sigma_1 \vdash M{:}B \mid s \quad \Delta; \Gamma_2; \Sigma_2 \vdash N{:}A \mid t}{\Delta; \Gamma_{1,2}; \Sigma_{1,2} \vdash (\lambda a^A.M)\, N \overset{f}{\rightharpoonup} \mathfrak{ba}(a^A.s, t) \rhd M^a_{N,t} : B \mid (\lambda a^A.s) \cdot t}\ \beta$$

$$\frac{\Delta; \cdot; \Sigma \vdash M{:}A \mid r \quad \Delta; \cdot; \Sigma \vdash \mathsf{Eq}(A, r, s) \mid e_1 \quad \Delta, u{:}A[\Sigma]; \Gamma_2; \Sigma_2 \vdash N{:}C \mid t}{\Delta; \Gamma_{1,2}; \Sigma_{1,2} \vdash O \overset{f}{\rightharpoonup} P : C^u_{\Sigma.s} \mid \text{LET}(u^{A[\Sigma]}.t, \Sigma.s)}\ \beta_\Box,$$

$$\frac{\alpha{:}\mathsf{Eq}(A) \in \Sigma \quad \Delta'; \cdot; \cdot \vdash \vartheta : \mathcal{T}^B \mid \theta \quad \Delta \subseteq \Delta' \quad \Delta; \cdot; \Sigma \vdash \mathcal{C}[\alpha\vartheta] : A \mid r \quad \Delta; \cdot; \Sigma \vdash \mathsf{Eq}(A, r, s) \mid e}{\Delta; \Gamma; \Sigma' \vdash !^{\Sigma}_{e}\mathcal{C}[\alpha\vartheta] \overset{f}{\rightharpoonup} !^{\Sigma}_{e}\mathcal{C}[\,\mathfrak{ti}(\theta, \alpha) \rhd e\vartheta\,] : [\![\, \Sigma.s \,]\!] A \mid \Sigma.s}\ \mathcal{I}$$

where, in $\beta_\Box$, $O \triangleq \text{LET}\, u^{A[\Sigma]}\, \text{BE}\, !^{\Sigma}_{e} M\, \text{IN}\, N$ and $P \triangleq \mathfrak{bb}(u^{A[\Sigma]}.t, \Sigma.s) \rhd N^u_{\Sigma.(M,s,e)}$. The auxiliary contexts $\mathcal{C}$ used in the rule $\mathcal{I}$ are also extended as follows:

$$\begin{array}{rl}\mathcal{C} ::= & \Box \mid \lambda a^A.\mathcal{C} \mid \mathcal{C}M \mid M\mathcal{C} \\ \mid & \text{LET}\, u^{A[\Sigma]}\, \text{BE}\, \mathcal{C}\, \text{IN}\, M \mid \text{LET}\, u^{A[\Sigma]}\, \text{BE}\, M\, \text{IN}\, \mathcal{C} \\ \mid & \alpha\{c_1/M, \ldots, c_j/M, c_{j+1}/\mathcal{C}, \ldots\} \\ \mid & e \rhd \mathcal{C}\end{array}.$$

As before, the type of the hole in $\mathcal{C}$ is B.

DEFINITION 5.2 (Full reduction)
Full reduction ($\overset{f}{\rightarrow}$) is defined as the union of *full principal reduction* ($\overset{f}{\mapsto}$) and permutation reduction ($\rightsquigarrow$).

In the sequel, we write $\overset{rf}{\mapsto}$ for the abovementioned restricted notion of reduction. That is, $\overset{f}{\mapsto}$, where $\beta_\Box$ has the additional assumption 'M is !-free'. The proof is by contradiction and is developed in two steps. The first shows that an infinite $\overset{f}{\mapsto} \cup \rightsquigarrow$ reduction sequence must include an infinite number of $\overset{f}{\mapsto}_{\mathcal{I}}$ steps. The second, that $\overset{f}{\mapsto}_{\mathcal{I}}$ is SN.

[5]We currently have no proof for unrestricted reduction; however, we believe the result should hold.

5.1 Step 1

We first note that $\overset{f}{\mapsto}_{\beta,\beta_\Box}$ is SN. This can be proved by defining a translation $\mathcal{S}(\bullet)$ on $\lambda^{\mathfrak{h}}$ types that 'forget' the modal connective and a similar translation from terms in $\lambda^{\mathfrak{h}}$ to terms of the simply typed lambda calculus with constants and products ($\lambda^{\supset,\times}$) such that: (i) it preserves typability (Lemma 5.1); (ii) it collapses terms related by permutation reduction (Lemma 5.2); and (iii) it maps full reduction to reduction in the simply typed lambda calculus (Lemma 5.4). Since we already know that $\rightsquigarrow$ is SN and that reduction in the simply typed lambda calculus is SN, our result shall follow. For a proposition A, $\mathcal{S}(A)$ is defined as follows; for contexts it behaves homomorphically producing multisets of labelled hypothesis:

$$\begin{array}{rcl@{\qquad}rcl} \mathcal{S}(P) & \triangleq & P & \mathcal{S}(\cdot) & \triangleq & \cdot \\ \mathcal{S}(A \supset B) & \triangleq & \mathcal{S}(A) \supset \mathcal{S}(B) & \mathcal{S}(\Gamma, a{:}A) & \triangleq & \mathcal{S}(\Gamma), a{:}\mathcal{S}(A) \\ \mathcal{S}([\![\, \Sigma.s \,]\!] A) & \triangleq & \mathcal{S}(A) & \mathcal{S}(\Delta, u{:}A[\Sigma]) & \triangleq & \mathcal{S}(\Delta), u{:}\mathcal{S}(A) \end{array} .$$

The *decoration* of a typed term is the expression obtained from replacing every occurrence of the trail lookup expression $\alpha\vartheta$ by $\alpha^B\vartheta$, where B is the result type of the lookup. We assume we have a constant c_B for each type B. For a term M, $\mathcal{S}(M)$ produces a term of the simply typed lambda calculus:

$$\begin{array}{rcl} \mathcal{S}(a) & \triangleq & a \\ \mathcal{S}(\lambda a^A.M) & \triangleq & \lambda a^{\mathcal{S}(A)}.\mathcal{S}(M) \\ \mathcal{S}(MN) & \triangleq & \mathcal{S}(M)\mathcal{S}(N) \\ \mathcal{S}(\langle u;\sigma\rangle) & \triangleq & u \\ \mathcal{S}(!^{\Sigma}_{e} M) & \triangleq & \mathcal{S}(M) \\ \mathcal{S}(\text{LET}\, u^{A[\Sigma]}\, \text{BE}\, M\, \text{IN}\, N) & \triangleq & (\lambda u^{\mathcal{S}(A)}.\mathcal{S}(N))\mathcal{S}(M) \\ \mathcal{S}(\alpha^B\vartheta) & \triangleq & (\lambda a^{\mathcal{S}(\mathcal{T}^B)}.c_B)\mathcal{S}(\vartheta) \\ \mathcal{S}(e \rhd M) & \triangleq & \mathcal{S}(M) \end{array} ,$$

where $\mathcal{S}(\mathcal{T}^B)$ abbreviates the product type $\langle \mathcal{S}(\mathcal{T}^B(c_1)), \ldots, \mathcal{S}(\mathcal{T}^B(c_{10}))\rangle$ and $\mathcal{S}(\vartheta)$ abbreviates the product term $\langle \mathcal{S}(\vartheta(c_1)), \ldots, \mathcal{S}(\vartheta(c_{10}))\rangle$. The translation of $\alpha^B\vartheta$ above guarantees that trail inspection maps to a β-step in $\lambda^{\supset,\times}$.

LEMMA 5.1
If $\Delta;\Gamma;\Sigma \vdash M : A \mid s$, then $\mathcal{S}(\Delta), \mathcal{S}(\Gamma) \vdash \mathcal{S}(M) : \mathcal{S}(A)$.

The following result is verified by noticing that $\mathcal{S}(\bullet)$ erases both evidence and the modal term constructor in terms.

LEMMA 5.2
If $M \rightsquigarrow N$, then $\mathcal{S}(M) = \mathcal{S}(N)$.

The translation function $\mathcal{S}(\bullet)$ permutes with both truth and validity substitution. Note that $\mathcal{S}(M)^a_{\mathcal{S}(N)}$ and $\mathcal{S}(M)^u_{\mathcal{S}(N)}$ below, is substitution in the simply typed lambda calculus. The proofs of these items is by induction on the structure of M; the third item resorts to the first one. Lemma 5.3 is required for Lemma 5.4.

LEMMA 5.3
(1) $\mathcal{S}(M\sigma) = \mathcal{S}(M)$;
(2) $\mathcal{S}(M^a_{N,t}) = \mathcal{S}(M)^a_{\mathcal{S}(N)}$; and
(3) $\mathcal{S}(M^u_{\Sigma.(N,t,e)}) = \mathcal{S}(M)^u_{\mathcal{S}(N)}$.

LEMMA 5.4
If $M \overset{f}{\mapsto}_{\beta,\beta_\Box} N$, then $\mathcal{S}(M) \overset{\lambda^{\supset,\times}}{\to} \mathcal{S}(N)$.

The following result is a consequence of Lemma 5.2, Lemma 5.4 and Lemma 5.3.

COROLLARY
If $M \overset{\beta,\beta_\Box}{\to} N$, then $\mathcal{S}(M) \overset{\lambda^{\supset,\times}}{\to} \mathcal{S}(N)$.

We may thus conclude with the following result.

PROPOSITION 5.5
$\overset{f}{\mapsto}_{\beta,\beta_\Box} \cup \rightsquigarrow$ is SN.

Therefore, an infinite $\overset{f}{\mapsto} \cup \rightsquigarrow$ reduction sequence must include an infinite number of $\overset{f}{\mapsto}_{\mathcal{I}}$ steps.

5.2 *Step 2*

Next we show that for $\overset{rf}{\mapsto}$ this is not possible. More precisely, we show that in an infinite $\overset{rf}{\mapsto} \cup \rightsquigarrow$ reduction sequence, there can only be a finite number of $\overset{f}{\mapsto}_{\mathcal{I}}$ steps. This entails:

PROPOSITION 5.6
$\overset{rf}{\mapsto} \cup \rightsquigarrow$ is SN. Hence $\lambda^{\mathfrak{h}}$, with the same restriction, is SN.

We now address the proof of the main lemma on which Proposition 5.6 relies (Lemma 5.8). We introduce a family of weight functions which strictly decrease by each application of a $\overset{f}{\mapsto}_{\mathcal{I}}$-step and which decreases with each application of a $\overset{rf}{\mapsto}_{\beta,\beta_\Box}$-step or $\rightsquigarrow$-step. A word on notation: $\langle\langle\ \rangle\rangle$ is the empty multiset; $\uplus$ is multiset union; and $n \uplus \mathcal{M}$ is the union of the multiset $\langle\langle n \rangle\rangle$ and $\mathcal{M}$, for $n \in \mathbb{N}$. We use the standard multiset extension $\prec$ of the well-founded ordering $<$ on natural numbers which is also well-founded. For each $n \in \mathbb{N}$ we define $\mathcal{W}_n(M)$ as the multiset given by the following inductive definition on M:

$$\begin{array}{rcl@{\qquad}rcl}
\mathcal{W}_n(a) & \triangleq & \langle\langle\ \rangle\rangle & \mathcal{W}_n(!^{\Sigma}_{e} M) & \triangleq & n * \mathcal{W}^t(M) \uplus \mathcal{W}_{n*\mathcal{W}^t(M)}(M) \\
\mathcal{W}_n(\lambda a^A.M) & \triangleq & \mathcal{W}_n(M) & \mathcal{W}_n(\text{LET } u \text{ BE } M \text{ IN } N) & \triangleq & \mathcal{W}_n(M) \uplus \mathcal{W}_n(N) \\
\mathcal{W}_n(MN) & \triangleq & \mathcal{W}_n(M) \uplus \mathcal{W}_n(N) & \mathcal{W}_n(\alpha\vartheta) & \triangleq & \biguplus_{i \in 1..10} \mathcal{W}_n(\vartheta(c_i)) \\
\mathcal{W}_n(\langle u; \sigma \rangle) & \triangleq & \langle\langle\ \rangle\rangle & \mathcal{W}_n(e \rhd M) & \triangleq & \mathcal{W}_n(M)
\end{array},$$

where $\mathcal{W}^t(M)$ is the number of free trail variables in M plus 1. Note that $\mathcal{W}^t(e \rhd M) \triangleq \mathcal{W}^t(M)$. The weight functions informally count the number of trail variables that are available for look-up in audited computation units. The principal reduction axiom β either erases the argument N or substitutes exactly one copy, given the affine nature of truth hypothesis. However, multiple copies of M can arise from $\beta_\Box$ reduction, possibly under '!' constructors (hence our restriction in item 2 below). Finally, we must take into account that although a trail variable is consumed by $\mathcal{I}$ it also copies the terms in ϑ (which may contain occurrences of the '!' constructor). In contrast to $\beta_\Box$, however, the consumed trail variable can be used to make the copies of '!' made by $e\vartheta$ weigh less than the outermost occurrence of '!' on the left-hand side of $\mathcal{I}$.

LEMMA 5.7
(1) $\mathcal{W}_n((\lambda a^A.M)N) \succeq \mathcal{W}_n(M^a_{N,t})$.

(2) If M has no occurrences of the modal term constructor, then $\mathcal{W}_n(\mathrm{LET}\, u^{A[\Sigma]}\, \mathrm{BE}\, !\overset{\Sigma}{e} M\, \mathrm{IN}\, N) \succ \mathcal{W}_n(\mathfrak{bb}(u^{A[\Sigma]}.t, \Sigma.s) \rhd N^{u}_{\Sigma.(M,s,e)})$.

(3) $\mathcal{W}_n(!\overset{\Sigma}{e}\mathcal{C}[\alpha\vartheta]) \succ \mathcal{W}_n(!\overset{\Sigma}{e}\mathcal{C}[\mathfrak{ti}(\theta,\alpha) \rhd e\vartheta])$.

From these results follow:

LEMMA 5.8

(1) $M \overset{rf}{\mapsto}_{\beta,\beta_\Box} N$ implies $\mathcal{W}_n(M) \succeq \mathcal{W}_n(N)$;

(2) $M \overset{f}{\mapsto}_{\mathcal{I}} N$ implies $\mathcal{W}_n(M) \succ \mathcal{W}_n(N)$; and

(3) $M \rightsquigarrow N$ implies $\mathcal{W}_n(M) = \mathcal{W}_n(N)$.

6 Conclusions

We have presented a proof theoretical analysis of a functional computation model that keeps track of its computation history. A Curry–Howard isomorphism of an affine fragment of Artemov's Justification Logic yields a lambda calculus $\lambda^{\mathfrak{h}}$ which models audited units of computation. Reduction in these units generates audit trails that are confined within the computation units. Moreover, addition of reification of trails yields a programming language where computation may inspect these local trails and make decisions based on them while at the same time conserving a sound logical foundation. We prove type safety for $\lambda^{\mathfrak{h}}$ and strong normalization for a restriction of it.

There are a number of possible research directions. The first is to study type-inference for the user syntax of $\lambda^{\mathfrak{h}}$. Another is to lift the restriction in the proof of strong normalization that M in the principal reduction axiom $\beta_\Box$ does not have occurrences of the audited computation unit constructor '!'. Second, consider adding features (polymorphism, recursion, etc.) in order to obtain a language closer to industrial scale ones. It would make sense to study audited computation in a classical setting where, based on audit trail look-up, the current continuation could be disposed of in favour of a more judicious computation. Finally, although examples from the security domain seem promising more (eg. [7]) is needed in order to better evaluate the applicability of these ideas.

Acknowledgements

The authors thank Sergei Artemov and Peter Thiemann for fruitful discussions.

Funding

This work was partially supported by Agencia Nacional de Promoción Científica y Tecnológica [PICT2012-2747] and by LIA INFINIS.

References

[1] M. Abadi and C. Fournet. Access control based on execution history. In *NDSS*. The Internet Society, 2003.

[2] J. Alt and S. Artemov. Reflective λ-calculus. In *Proceedings of the Dagstuhl-Seminar on Proof Theory in CS*, Vol. 2183 of *LNCS*, 2001.

[3] S. Artemov and E. Bonelli. The intensional lambda calculus. In *LFCS*, Vol. 4514 of *LNCS*, pp. 12–25. Springer, 2007.

[4] S. Artemov. Operational modal logic. *Technical Report MSI 95-29*, Cornell University, 1995.

[5] S. Artemov. Explicit provability and constructive semantics. *Bulletin of Symbolic Logic*, **7**, 1–36, 2001.
[6] S. Artemov. Justification logic. In *JELIA*, Vol. 5293 of *Lecture Notes in Computer Science*, S. Hölldobler, C. Lutz and H. Wansing, eds, pp. 1–4. Springer, 2008.
[7] A. Banerjee and D. Naumann. History-based access control and secure information flow. In *CASSIS*, Vol. 3362 of *Lecture Notes in Computer Science*, G. Barthe, L. Burdy, M. Huisman, J.-L. Lanet and T. Muntean, eds, pp. 27–48. Springer, 2004.
[8] H. Barendregt. *Handbook of Logic in Computer Science*, chapter Lambda Calculi with Types. Oxford University Press, 1992.
[9] F. Bavera and E. Bonelli. Justification logic and history based computation. In *ICTAC*, Vol. 6255 of *Lecture Notes in Computer Science*, A. Cavalcanti, D. Déharbe, M.-C. Gaudel and J. Woodcock, eds, pp. 337–351. Springer, 2010.
[10] E. Bonelli and F. Feller. The logic of proofs as a foundation for certifying mobile computation. In *LFCS*, Vol. 5407 of *LNCS*, pp. 76–91. Springer, 2009.
[11] E. Bonelli and G. Steren. Hypothetical logic of proofs. *Logica Universalis*, **8**, 103–140, 2014.
[12] V. Brezhnev. On the logic of proofs. In *Proceedings of the Sixth ESSLLI Student Session*, pp. 35–45, 2001.
[13] R. Davies and F. Pfenning. A modal analysis of staged computation. In *23rd POPL*, pp. 258–270. ACM Press, 1996.
[14] R. Davies and F. Pfenning. A judgmental reconstruction of modal logic. *Journal of MSCS*, **11**, 511–540, 2001.
[15] R. Davies and F. Pfenning. A modal analysis of staged computation. *Journal of the ACM*, **48**, 555–604, 2001.
[16] J. Giesl, R. Thiemann, P. Schneider-Kamp and S. Falke. Automated termination proofs with aprove. In *RTA*, Vol. 3091 of *LNCS*, pp. 210–220. Springer, 2004.
[17] L. Jia and D. Walker. Modal proofs as distributed programs (extended abstract). In *ESOP*, Vol. 2986 of *Lecture Notes in Computer Science*, D. A. Schmidt, ed., pp. 219–233. Springer, 2004.
[18] P. Martin-Löf. On the meaning of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic 1*, **1**, 11–60, 1996.
[19] J. Moody. Logical mobility and locality types. In *LOPSTR*, Vol. 3573 of *Lecture Notes in Computer Science*, S. Etalle, ed., pp. 69–84. Springer, 2004.
[20] T. Murphy VII, K. Crary, R. Harper and F. Pfenning. A symmetric modal lambda calculus for distributed computing. In *LICS*, pp. 286–295. IEEE Computer Society, 2004.
[21] T. Murphy VII, K. Crary and R. Harper. Distributed control flow with classical modal logic. In *CSL*, Vol. 3634 of *Lecture Notes in Computer Science*, C.-H. L. Ong, ed., pp. 51–69. Springer, 2005.
[22] T. Murphy VII, K Crary and R. Harper. Type-safe distributed programming with ml5. In *TGC*, Vol. 4912 of *Lecture Notes in Computer Science*, G. Barthe and C. Fournet, eds, pp. 108–123. Springer, 2007.
[23] A. Nanevski, F. Pfenning and B. Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, **9**, 2008.
[24] F. Pfenning and H-C. Wong. On a modal λ-calculus for S4. In *Proceedings of the Eleventh Conference on Mathematical Foundations of Programming Semantics (MFPS'95)*, 1995. Available as Electronic Notes in Theoretical Computer Science, Volume 1, Elsevier.
[25] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

Received 11 January 2012