Strongly Normalizing Audited Computation*†

Wilmer Ricciotti[1] and James Cheney[2]

1 **LFCS, University of Edinburgh, Edinburgh, UK**
`wricciot@inf.ed.ac.uk`
2 **LFCS, University of Edinburgh, Edinburgh, UK**
`jcheney@inf.ed.ac.uk`

Abstract

Auditing is an increasingly important operation for computer programming, for example in security (e.g. to enable history-based access control) and to enable reproducibility and accountability (e.g. provenance in scientific programming). Most proposed auditing techniques are ad hoc or treat auditing as a second-class, extralinguistic operation; logical or semantic foundations for auditing are not yet well-established. *Justification Logic* (JL) offers one such foundation; Bavera and Bonelli introduced a computational interpretation of JL called λ^h that supports auditing. However, λ^h is technically complex and strong normalization was only established for special cases. In addition, we show that the equational theory of λ^h is inconsistent. We introduce a new calculus λ^{hc} that is simpler than λ^h, consistent, and strongly normalizing. Our proof of strong normalization is formalized in Nominal Isabelle.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Lambda calculus, Justification Logic, Strong Normalization, Audited Computation

Digital Object Identifier 10.4230/LIPIcs.CSL.2017.36

1 Introduction

Auditing is of increasing importance in many areas, including in security (e.g. history-based access control [1], evidence-based audit [24, 15]) and to provide reproducibility or accountability for scientific computations (e.g. provenance tracking [3]). To date, most approaches to auditing propose to instrument programs in some *ad hoc* way, to provide a record of the computation, sometimes called *provenance*, *audit log*, *trace*, or *trail*, which is typically meant to be inspected after the computation finishes. Foundations of provenance or auditing have not been fully developed, and auditing typically remains a second-class citizen (or even an alien concept), in that trails are not accessible to the program itself during execution, but only to an external monitoring layer. We seek logical and semantic foundations for *self-auditing programs* that provide first-class recording and auditing primitives.

In this paper, we explore how ideas from *justification logic* [5] (a generalization of the *Logic of Proofs* [6]) can provide such a foundation. *Justification Logic* is the general name for a family of logics that refine the epistemic reading of modal necessity $\Box A$ ("A is known") by indexing such formulas with *justifications*, or (descriptions of) evidence that A holds. So, the formula $[\![a]\!]\, A$ can be read as "justification a is evidence (proof) establishing A". In elementary

* An extended version of this paper is available at `https://arxiv.org/abs/1706.03711`.
† Effort sponsored by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-13-1-3006. The U.S. Government and University of Edinburgh are authorised to reproduce and distribute reprints for their purposes notwithstanding any copyright notation thereon.

forms of justification logic, the justifications consist of uninterpreted constants that can be combined with algebraic operations (e.g. formal sum and product). A representative axiom of justification logic is the evidence-aware version of Modus Ponens:

$$[\![a]\!]\,(A \supset B) \supset [\![b]\!]\,A \supset [\![a \cdot b]\!]\,B$$

which can be read as "if a proves $A \supset B$ and b proves A then a together with b proves B". Another representative axiom is the reflection principle:

$$[\![a]\!]\,A \supset [\![!a]\!]\,[\![a]\!]\,A$$

which can be read as "if a proves A then $!a$ proves that a proves A", where ! is a *reflection* operator on justifications. Notice that in both cases, if we ignore the evidence annotations a, b, we get standard axioms "K" and "4" of modal logic respectively [16].

This paper explores using (constructive) justification logic as a foundation for programming with auditing, building on of previous work on calculi for modal logic [21] and justification logic [7, 9]. Artemov and Bonelli [7] introduced the *intensional lambda calculus* $\lambda^{\mathbf{I}}$, which builds upon Pfenning and Davies' type theory for modal logic [21] by recording *proof codes* in judgements and modalities; proof codes are a special case of proof terms. In $\lambda^{\mathbf{I}}$, "trails" that witness the equivalence of proof codes were introduced, but these trails could not be inspected by other constructs. More recently, Bavera and Bonelli [9] introduced λ^h, an extension of $\lambda^{\mathbf{I}}$ that allowed *trail inspection*, which can audit a computation's trail of evaluation from the currently-enclosing box constructor up to the point of inspection.

Trail inspection is a first-class construct of λ^h, and can be performed at arbitrary program points (including inside another trail inspection). Bavera and Bonelli outlined how this construct can be used to implement *history-based access control* [1] policies in which access control decisions are made based on the complete previous call history. Trail inspection could also be used for profiling, for example by counting the number of calls of each function in a subcomputation, or for more general auditing [4], for example by comparing the trails of multiple computations to look for differences or anomalies in their execution history. Furthermore, all of these applications can be supported in a single language, rather than requiring ad hoc changes to accommodate different monitoring semantics. (It is worth noting that these applications typically use data structures such as sets or maps that are not provided in λ^h; however, they are straightforward to add.)

Bavera and Bonelli proved basic metatheoretic properties of λ^h including subject reduction, and gave a partial proof of strong normalization. The latter proof considered only terms in which there was no nesting of box constructors. Although they conjectured that strong normalization holds for the full system, the general case was left as an open question. However, λ^h is a somewhat complex system, due to subtle aspects of the metatheory of trail inspection. Trails are viewed as proof terms witnessing equivalence of proof codes. Proof codes include trail inspections, whose results depend on the history of the term being executed. λ^h's trails are viewed as equality proofs and are thus subject to symmetry: operationally, the language provides a trail constructor $\mathbf{s}(q)$ that converts a trail q of type $s = t$ to one of type $t = s$. Since the reduct of an outer trail inspection depends on a non-local, volatile context (the trail declared in the nearest outer box constructor) symmetry makes it possible to equate any two codes of the same type: in other words, the theory of trails becomes inconsistent.

To avoid this problem, Bavera and Bonelli annotated trail inspections with affine trail variables α, α' representing the volatile context, and trail inspection is then denoted by $\alpha\theta$, where θ is a collection of functions specifying a structural recursion over the trail. Nevertheless, in λ^h we can prove any two codes s and t equivalent. A detailed example, along with the rules of λ^h, are presented in Appendix A; here, we sketch the derivation:

$$\dfrac{\dfrac{s = q_1\theta \qquad q_1\theta = \alpha\theta}{s = \alpha\theta} \qquad \dfrac{\dfrac{t = q_2\theta \qquad q_2\theta = \alpha\theta}{t = \alpha\theta}}{\alpha\theta = t}}{s = t}\ (*)$$

where q_1, q_2 and θ are chosen so that $s = q_1\theta$ and $t = q_2\theta$, which is possible whenever s and t have the same type. The problem seems to originate from the transitivity rule marked with (*), which constitutes an exception to the affineness of trail variables, since the same proof code must necessarily appear as the right-hand side of the first premise and as the left-hand side of the second one. This allows trail inspection on α to be triggered twice. It should be noted that this property does *not* contradict any proved results by Bavera and Bonelli; instead, it can be viewed as an (undesirable) form of proof-irrelevance: that is, $[\![s]\!]\, A \to [\![t]\!]\, A$ is inhabited for any well-formed proof codes s, t. Of course, if this is the case then the stated justification s for $[\![s]\!]\, A$ need have nothing with to do its actual proof, defeating the purpose of introducing the proof codes in the first place.

We present a new calculus, λ^{hc}, that overcomes these issues with λ^h, and provides a more satisfactory foundation for audited computation, including the following contributions:

- We dispense with λ^h's trail variables, instead adopting a trail inspection construct $\iota(\theta)$ that can be used anywhere in the program. We also adjust λ^h's treatment of trails by no longer viewing them as witnesses to proof code *equality* but rather to (directed) *reducibility*. At a technical level, this simply consists of dropping the symmetry axiom and trail form. Viewing trails as asymmetric reducibility proofs means that it is no longer inconsistent to relate a trail inspection redex with all of its possible contracta nondeterministically, and it also obviates the need for trail variables (or ordinary variables) to be affine; once we lift these restrictions, however, the rationale for trail variables evaporates. We present the improved system, and summarize its metatheory (including a proof that its equational theory is consistent) in Section 2.
- To show that λ^{hc} is well-behaved, we prove strong normalization (SN). Our proof of SN actually shows a stronger result: SN holds even if we conservatively assume that trail inspection operations can choose *any* well-formed trail; this is a useful insight because it means that we can extend the result to more realistic languages with more complex trails. We summarize this proof, which we fully formalized in Nominal Isabelle[1].

Section 4 discusses related work: in particular, we argue that our approach can be used to prove strong normalization for λ^h as well, even in the presence of the symmetry rule.

2 A history-aware calculus

2.1 Overview

In this section we describe λ^{hc}, an extension of the simply-typed lambda calculus allowing for several forms of auditing, thanks to an internalized notion of *computation history*. It is based in many respects on λ^h, but since our preferred notation differs in important details we give a self-contained presentation.

Operationally, we add to the simply typed λ-calculus a new term form $!_q M$, where M is an expression and q is a *trail* containing a symbolic description of the past history of M. History-aware expressions will be given their own type: if M has type A, and q is an

[1] The full formalization can be found at `http://www.wilmer-ricciotti.net/FORMAL/ccau.tgz`. A more detailed discussion can be found in the extended version of this paper.

appropriate trail showing how M has reduced, then the type of $!_q M$ will be indicated by $[\![s]\!]\, A$; the symbol '!' is called "bang". In our language, we will initially evaluate expressions in the form $!\ M$, where the absence of a subscript indicates that no computation has happened within M yet. As we reduce M, the bang will be annotated with a subscript, for example

$$!\ M \rightsquigarrow !_q N$$

where q must explain how N was obtained from the initial expression M.

As a slightly more involved example, we consider the following expression:

$$!\ \mathsf{if}\ (1 \stackrel{?}{=} 2)\ M\ N$$

Here, $\stackrel{?}{=}$ represents an equality test between integers, while M and N represent respectively the 'then' and 'else' branches of the if statement. To evaluate the expression, we first reduce the equality test to $\#F$ (boolean false), and the if to its 'else' branch. In λ^{hc}, this computation will be described as:

$$!\ \mathsf{if}\ (1 \stackrel{?}{=} 2)\ M\ N \rightsquigarrow !_{\mathcal{Q}[q]}\mathsf{if}\ \#F\ M\ N \rightsquigarrow !_{\mathbf{t}(\mathcal{Q}[q],q')} N$$

where q, $\mathcal{Q}$, and q' are defined in such a way that:

- q explains how $1 \stackrel{?}{=} 2$ reduces to $\#F$
- $\mathcal{Q}$ is a context indicating that q was applied to the guard of an if statement – in other words, it describes a congruence rule of reduction
- q' describes the reduction of $\mathsf{if}\ \#F \cdots$ to its 'else' branch
- $\mathbf{t}$ combines its two arguments by means of transitivity.

To make use of history-aware expressions, we will also provide constructs to access their contents or to inspect their history.

2.2 Syntax

We now give the formal definition of λ^{hc}. The language includes three syntactic classes, which we have already presented informally in the overview:

- *codes* s, t correspond to (extended) lambda expressions which have not been evaluated yet: in other words, they only contain bangs with no history
- *trails* q, q' encode the history of the computation transforming a certain code into another
- *terms* M, N are lambda expressions that have been partially evaluated: they are similar to codes, but their bangs are annotated with trails.

The *types* of the language (denoted by metavariables $A, B, C, \ldots$) include atomic types P, functions $A \supset B$, and *audited units* $[\![s]\!]\, A$. Notice that types contain codes, which can contain types in turn. Audited types correspond to terms that capture and record all the computation happening inside them, for subsequent retrieval.

The syntax of the language is presented in Figure 1. Codes and terms are defined in a similar way to Pfenning and Davies [21]. We use the symbol '!' ("bang") to denote box introduction in both codes and terms; the corresponding elimination form is given by the let operator, which unpacks an audited code or term allowing us to access its contents. The variable declared by a let is bound in its second argument: in essence, $\mathrm{let}(u := !_q M, N)$ will reduce to N, where u has been replaced by M; q will be used to produce a trail explaining this reduction.

Codes:

$$
\begin{array}{lcll}
s,t,\ldots & ::= & a & \\
& | & u & \\
& | & \lambda a^A.s & \\
& | & (s\ t) & \\
& | & !s & \\
& | & \mathrm{let}(u^A := s, t) & \\
& | & \iota(\theta) &
\end{array}
$$

Terms:

$$
\begin{array}{lcll}
M,N,\ldots & ::= & a & \text{simple variable} \\
& | & u & \text{audited variable} \\
& | & \lambda a^A.M & \text{abstraction} \\
& | & (M\ N) & \text{application} \\
& | & !_q M & \text{box introduction} \\
& | & \mathrm{let}(u^A := M, N) & \text{box elimination} \\
& | & \iota(\vartheta) & \text{trail inspection}
\end{array}
$$

Trails:

$$
\begin{array}{lcll}
q,q',\ldots & ::= & \mathbf{r}(s) & \text{reflexivity} \\
& | & \mathbf{t}(q,q') & \text{transitivity} \\
& | & \beta(a^A.s,t) & \text{beta reduction} \\
& | & \beta_{\square}(s,u^A.t) & \text{box beta reduction} \\
& | & \mathbf{ti}(q,\theta) & \text{trail inspection reduction} \\
& | & \mathbf{lam}(a^A.q) & \text{congruence wrt lambda} \\
& | & \mathbf{app}(q,q') & \text{congruence wrt application} \\
& | & \mathbf{let}(q,u^A.q') & \text{congruence wrt let} \\
& | & \mathbf{trpl}(\zeta) & \text{congruence wrt trail insp.}
\end{array}
$$

Trail inspection branches:

$$
\begin{array}{lcll}
\theta,\theta',\ldots & ::= & \{\overrightarrow{s/\psi}\} & \text{for codes} \\
\vartheta,\vartheta',\ldots & ::= & \{\overrightarrow{M/\psi}\} & \text{for terms} \\
\zeta,\zeta',\ldots & ::= & \{\overrightarrow{q/\psi}\} & \text{for trails}
\end{array}
$$

Trail inspection branch labels:

$$
\begin{array}{lcl}
\psi,\psi',\ldots & ::= & \mathbf{r} \mid \mathbf{t} \mid \beta \mid \beta_{\square} \mid \mathbf{ti} \\
& | & \mathbf{lam} \mid \mathbf{app} \mid \mathbf{let} \\
& | & \mathbf{trpl}_{[]} \mid \mathbf{trpl}_{::} \mid \mathbf{d}
\end{array}
$$

Figure 1 Syntax of λ^{hc}.

▶ **Example 1.** λ^{hc} allows us to manipulate history-carrying data explicitly. For instance, we could have the following history-carrying natural numbers:

$$!_q 2 : [\![1+1]\!]\,\mathbb{N} \qquad !_{q'} 6 : [\![\mathsf{fact}\ 3]\!]\,\mathbb{N}$$

The trail q represents the history of 2, i.e. a witness to the computation that produced 2 by adding $1+1$; similarly, q' describes how computing $\mathsf{fact}\ 3$ (for a suitable definition of the factorial function) yielded 6.

Now supposing we wish to add these two numbers together, at the same time retaining their history, we will use the let construct to look inside them:

$$!\ \mathrm{let}(u^{\mathbb{N}} := !_q 2, \mathrm{let}(v^{\mathbb{N}} := !_{q'} 6, u+v)) \overset{*}{\rightsquigarrow} !_{q''} 8 : \left[\!\!\left[\mathrm{let}(u^{\mathbb{N}} := !\ 1+1, \mathrm{let}(v^{\mathbb{N}} := !\ \mathsf{fact}\ 3, u+v))\right]\!\!\right]\ \mathbb{N}$$

where the type of the terms is preserved under reduction, and the final trail q'', produced by composing q and q', is expected to log the reductions of $1+1$, of $\mathsf{fact}\ 3$, and of the two lets.

Trail inspection codes and terms will perform computation by primitive recursion on a certain trail, which represents an audited unit's computation history. Trail inspection references the computation history of the current context ($\iota(\theta)$ and $\iota(\vartheta)$). Here, θ and ϑ contain the branches that define the recursion.

The only difference between codes and terms is that in terms, bang operators are annotated with a trail q: this follows the intuition that terms are codes considered with their computational evolution. Trails represent (multi-step) reductions relating two codes: thus, their structure matches the derivation tree of the reduction itself.

In this definition, when a trail contains an argument composed of a variable and another expression separated by a dot, the variable should be considered bound in that expression. The constructors $\beta, \beta_{\square}$ and **ti** define atomic reductions: β represents the regular β-reduction; $\beta_{\square}$ is similar, but reduces a let-bang combination; finally, **ti** reduces trail inspections. The constructors **lam**, **app**, **let**, and **trpl** provide congruence rules for all code constructors (with

the notable exception of bangs, since trails are not allowed to escape an audited unit): for example, if q is a trail from s to t, $\mathbf{lam}(a^A.q)$ constructs a trail from $\lambda a^A.s$ to $\lambda a^A.t$. Finally, the definition of trails is completed by reflexivity and transitivity (but excluding symmetry). Notice that reflexive trails take as a parameter the code whose identity is stated (for brevity, we will keep this parameter implicit in our examples).

▶ **Example 2.** We can build a pair of natural numbers using Church's encoding: λ^{hc} makes it possible to keep a log of this operation without adding specialized code:

$$\begin{aligned} & !\ ((\lambda x, y, p.p\ x\ y)\ 2)\ 6 \\ \rightsquigarrow & !_{\mathbf{t}(\mathbf{r},\mathbf{app}(\boldsymbol{\beta}(x.\lambda y,p.p\ x\ y,2),\mathbf{r}))}\ (\lambda y, p.p\ 2\ y)\ 6 \\ \rightsquigarrow & !_{\mathbf{t}(\mathbf{t}(\mathbf{r},\mathbf{app}(\boldsymbol{\beta}(x.\lambda y,p.p\ x\ y,2),\mathbf{r})),\boldsymbol{\beta}(y.\lambda p.p\ 2\ y,6))}\ \lambda p.p\ 2\ 6 \end{aligned}$$

The trail for the first computation step is obtained by transitivity (trail costructor $\mathbf{t}$) from the original trivial trail ($\mathbf{r}$, i.e. reflexivity) composed with $\boldsymbol{\beta}(x.\lambda y, p.p\ x\ y, 2)$, which describes the beta-reduction of $(\lambda x, y, p.p\ x\ y)\ 2$: this subtrail is wrapped in a congruence **app** because the reduction takes place deep inside the left-hand subterm of an application (the other argument of **app** is reflexivity, because no reduction takes place in the right-hand subterm).

The second beta-reduction happens at the top level and is thus not wrapped in a congruence. It is combined with the previous trail by means of transitivity.

Inspection branches θ, ϑ and ζ are maps from trail constructors to codes, terms, or trails. More precisely, since some of the trail constructors have a variable arity, in that case we need to distinguish a nil case ($[]$) and a cons case ($::$) to specify the corresponding branch of recursion; in addition, we do not require the map to be exhaustive, but allow a "default", catch-all branch specified using the label $\mathbf{d}$: this does not make the language more expressive, but will allow us to extend the language with additional constants without needing to modify the base syntax. We now give a formal definition of the result of trail inspection.

▶ **Definition 3.** The operation $q\theta$, which produces a code by structural recursion on q applying the inspection branches θ, is defined as follows:

$$\begin{aligned} \psi(\_)\theta &\triangleq \theta(\mathbf{d}) \quad (\text{if } \psi \notin \mathrm{dom}(\theta)) & \mathbf{lam}(a^A.q)\theta &\triangleq \theta(\mathbf{lam})\ (q\theta) \\ \mathbf{r}(\_)\theta &\triangleq \theta(\mathbf{r}) & \mathbf{app}(q,q')\theta &\triangleq \theta(\mathbf{app})\ (q\theta)\ (q'\theta) \\ \mathbf{t}(q,q')\theta &\triangleq \theta(\mathbf{t})\ (q\theta)\ (q'\theta) & \mathbf{let}(q,u^A.q')\theta &\triangleq \theta(\mathbf{let})\ (q\theta)\ (q'\theta) \\ \boldsymbol{\beta}(\_,\_)\theta &\triangleq \theta(\boldsymbol{\beta}) & \mathbf{trpl}(\{\})\theta &\triangleq \theta(\mathbf{trpl}_{[]}) \\ \boldsymbol{\beta}_{\square}(\_,\_)\theta &\triangleq \theta(\boldsymbol{\beta}_{\square}) & \mathbf{trpl}(\{q/\psi, \overrightarrow{q'/\psi'}\})\theta &\triangleq \theta(\mathbf{trpl}_{::})\ (q\theta)\ (\mathbf{trpl}(\{\overrightarrow{q'/\psi'}\})\theta) \\ \mathbf{ti}(\_,\_)\theta &\triangleq \theta(\mathbf{ti}) \end{aligned}$$

The twin operation $q\vartheta$, which produces a term by structural recursion on q, is defined similarly, by replacing each occurrence of θ with ϑ.

▶ **Example 4.** For profiling purposes, we might be interested in counting the number of computation steps that have happened thus far in an audited unit. This can be accomplished by means of a trail replacement ϑ_+ such that:

$$\vartheta_+(\psi) = \begin{cases} 1 & \text{if } \psi = \boldsymbol{\beta}, \boldsymbol{\beta}_{\square}, \mathbf{ti} \\ \lambda x.x & \text{if } \psi = \mathbf{lam} \\ \lambda x, y.x + y & \text{if } \psi = \mathbf{t}, \mathbf{app}, \mathbf{let}, \mathbf{trpl}_{::} \\ 0 & \text{else} \end{cases}$$

This trail replacement counts 1 for each reduction step ($\boldsymbol{\beta}, \boldsymbol{\beta}_{\square}, \mathbf{ti}$); the cases for transitivity and congruences add together the recursive results for subtrails; the last clause ignores other constructors, most notably $\mathbf{r}$ and $\mathbf{ti}_{[]}$, which are not reduction steps and do not have subtrails.

2.3 Substitutions

λ^{hc} employs two notions of substitution, one for each kind of variable supported by the language:

- Simple variable substitution, which is not history-aware, is applied to codes, types, or terms and maps a simple variable to, respectively, a code or a term; we denote it by $[t/a]$ or $[N/a]$, which can also be written γ for short.
- Audited variable substitution, which is history-aware, is applied to codes or terms and maps an audited variable to, respectively, a code t or a tuple (N, q, t); we denote it by $[t/u]$ or $[(N, q, t)/u]$, which can also be written δ for short.

We will also use the letter η to refer to either of the two flavors of substitution when more precision is not needed.

▶ **Definition 5** (substitution on codes). Substitution of simple and audited variables on a code r ($r[s/a]$ and $r[s/u]$) is defined as usual (avoiding variable capture). The full definition is shown in Appendix B.1.

Each of the two substitutions on codes is extended to trails and types in the obvious way, by traversing the trail or type until a subexpression containing a code s is found, then resorting to substitution on s. To define substitution on terms, we first need auxiliary definitions of the *source* and *target* of a trail:

▶ **Definition 6.** The source and target of a trail q ($\mathrm{src}(q)$ and $\mathrm{tgt}(q)$) are defined as follows:

$$\begin{aligned}
\mathrm{src}(\mathbf{r}(s)) &\triangleq s & \mathrm{tgt}(\mathbf{r}(s)) &\triangleq s \\
\mathrm{src}(\mathbf{t}(q_1, q_2)) &\triangleq \mathrm{src}(q_1) & \mathrm{tgt}(\mathbf{t}(q_1, q_2)) &\triangleq \mathrm{tgt}(q_2) \\
\mathrm{src}(\boldsymbol{\beta}(a^A.s_1, s_2)) &\triangleq (\lambda a^A.s_1)\ s_2 & \mathrm{tgt}(\boldsymbol{\beta}(a^A.s_1, s_2)) &\triangleq s_1\,[s_2/a] \\
\mathrm{src}(\boldsymbol{\beta}_{\square}(s_1, v^A.s_2)) &\triangleq \mathrm{let}(v^A := !s_1, s_2) & \mathrm{tgt}(\boldsymbol{\beta}_{\square}(s_1, v^A.s_2)) &\triangleq s_2\,[s_1/v] \\
\mathrm{src}(\mathbf{ti}(q', \theta)) &\triangleq \iota(\theta) & \mathrm{tgt}(\mathbf{ti}(q', \theta)) &\triangleq q'\theta
\end{aligned}$$

The omitted cases are routine and shown in Appendix B.2.

A valid trail q represents evidence that $\mathrm{src}(q)$ reduces to $\mathrm{tgt}(q)$ (we formally state this result in the next section).

We omit the definition of simple variable substitution on terms for brevity (it is similar to the corresponding substitution on codes). The definition of audited variable substitution on terms, however, deserves some more explanation: since this substitution arises from the unpacking of a bang term, which contains a trail we need to take care of, it takes the form $[(N, q, t)/u]$, where q is the computation history that led from the code t to the term N.

For related reasons, this substitution can be applied to a term to produce both a new term and a corresponding trail: the two operations are written $M \times \delta$ and $M \ltimes \delta$, respectively.

▶ **Definition 7** (audited variable substitution (terms)). The operations $M \times [(N, q, t)/u]$ and $M \ltimes [(N, q, t)/u]$ are defined as follows:

$$\begin{aligned}
u \ltimes [(N, q, t)/u] &\triangleq N & (!_{q'} R) \ltimes [(N, q, t)/u] &\triangleq !_{\mathbf{t}(q'[t/u], R \times [(N,q,t)/u])}(R \ltimes [(N, q, t)/u]) \\
u \times [(N, q, t)/u] &\triangleq q & (!_{q'} R) \times [(N, q, t)/u] &\triangleq \mathbf{r}(!(\mathrm{src}(q')\,[t/u]))
\end{aligned}$$

The omitted cases are routine and shown in Appendix B.3. The main feature of interest here is that when we $\ltimes$-substitute (N, q, t) for u in an audited unit $!_{q'}$, we need to augment the trail with a step showing how the replaced occurrence of u evaluated from t to N, as well as substituting for other occurrences of u; the corresponding case of $\times$-substitution returns the appropriate reflexivity trail, since no trail escapes a bang as a result of substitution.

▶ **Example 8.** We consider again the term from Example 1:

$$!\ \mathrm{let}(u^{\mathbb{N}} := !_q 2, \mathrm{let}(v^{\mathbb{N}} := !_{q'} 6, u + v))$$

Audited variable substitution is used to reduce let-! combinations; for example, to reduce the outer let in the example, we need to compute

$$\mathrm{let}(v^{\mathbb{N}} := !_{q'} 6, u + v) \ltimes [2, q, \mathrm{src}(q)/u] = \mathrm{let}(v^{\mathbb{N}} := !_{q'} 6, 2 + v)$$
$$\mathrm{let}(v^{\mathbb{N}} := !_{q'} 6, u + v) \times [2, q, \mathrm{src}(q)/u] = \mathbf{let}(\mathbf{r}, \mathbf{app}(\mathbf{app}(\mathbf{r}, q), \mathbf{r}))$$

(we assume that the infix notation for + is syntactic sugar for two nested applications).

2.4 Type system

We now introduce typing rules for codes, trails, and terms. These rules use three corresponding judgments with the following shape:

$\Delta; \Gamma \vdash s : A$	s has type A
$\Delta; \Gamma \vdash q : s \stackrel{\triangleright}{=}_A t$	q witnesses s reducing to t, with type A
$\Delta; \Gamma \vdash M : A \mid s$	M has type A and code s

The environments Δ and Γ are list of type declarations for audited and simple variables respectively, in the form $u :: A$ or $a : A$.

Figure 2 shows the typing rules of the language. In the rules for codes, if we forget for a moment about trail inspections, it is easy to recognize Pfenning and Davies's judgmental presentation of modal logic. The rules for terms are very similar, the biggest difference being found in rule T-Bang: to typecheck $!_q M$, we first obtain the type A and code t for M; then we typecheck the trail q to ensure that it has type A and target t; if both checks are successful, we can return $[\![s]\!]\, A$ as the type of the expression, and $!s$ as its code, where s is the source of q. This also shows that when we say that $!s$ is the code of $!_q M$, we mean that the term is the result of reducing the code, and the trail q records the computation history.

The rules for trail inspection, both as a code and as a term, rely on an auxiliary definition of $\mathcal{T}^B$, which is a function from inspection branch labels to the corresponding type; it depends on the superscript B, which refers to the output type of the inspection. The rule for inspection codes (TI) checks that the types of all the branches in θ match their labels. This requires θ to be defined on the default branch $\mathbf{d}$ to guarantee that the inspection is exhaustively defined. We define $\mathcal{T}^B$ as follows:

$$\mathcal{T}^B(\psi) \triangleq \begin{cases} B & (\psi = \mathbf{d}, \mathbf{r}, \beta, \beta_{\square}, \mathbf{ti}, \mathbf{iti}, \mathbf{trpl}_{[]}) \\ B \supset B & (\psi = \mathbf{lam}, \mathbf{itb}_{[]}) \\ B \supset B \supset B & (\psi = \mathbf{t}, \mathbf{app}, \mathbf{let}, \mathbf{itb}_{::}, \mathbf{trpl}_{::}) \end{cases}$$

Rather than typecheck a term to compute its code, we can define a function performing this operation directly.

▶ **Definition 9.** The code of a term M (notation: $\mathrm{cd}(M)$) is the code obtained by replacing all occurrences of $!_q N$ with $!\mathrm{src}(q)$. The full definition is shown in Appendix B.4.

$\boxed{\Delta;\Gamma \vdash s : A}$

$$\frac{a : A \in \Gamma}{\Delta;\Gamma \vdash a : A}\ \textsc{Var} \qquad \frac{\Delta;\Gamma, a : A \vdash s : B}{\Delta;\Gamma \vdash \lambda a^A.s : A \supset B}\ \supset I \qquad \frac{\Delta;\Gamma \vdash s : A \supset B \qquad \Delta;\Gamma \vdash t : A}{\Delta;\Gamma \vdash s\ t : B}\ \supset E$$

$$\frac{u :: A \in \Delta}{\Delta;\Gamma \vdash u : A}\ \textsc{mVar} \qquad \frac{\Delta;\cdot \vdash t : A}{\Delta;\Gamma \vdash !t : [\![t]\!]\ A}\ \Box I \qquad \frac{\Delta;\Gamma \vdash s : [\![r]\!]\ A \qquad \Delta, u :: A;\Gamma \vdash t : C}{\Delta;\Gamma \vdash \mathrm{let}(u^A := s, t) : C\,[r/u]}\ \Box E$$

$$\frac{\mathbf{d} \in \mathrm{dom}(\theta) \qquad \left[\Delta;\cdot \vdash \theta(\psi) : \mathcal{T}^B(\psi)\right]_{\psi \in \mathrm{dom}(\theta)}}{\Delta;\Gamma \vdash \iota(\theta) : B}\ \textsc{TI}$$

$\boxed{\Delta;\Gamma \vdash M : A \mid s}$

$$\frac{a : A \in \Gamma}{\Delta;\Gamma \vdash a : A \mid a}\ \textsc{T-Var} \qquad \frac{\Delta;\Gamma, a : A \vdash M : B \mid s}{\Delta;\Gamma \vdash \lambda a^A.M : A \supset B \mid \lambda a^A.s}\ \textsc{T-Abs}$$

$$\frac{\Delta;\Gamma \vdash M : A \supset B \mid s \qquad \Delta;\Gamma \vdash N : A \mid t}{\Delta;\Gamma \vdash M\ N : B \mid s\ t}\ \textsc{T-App}$$

$$\frac{u :: A \in \Delta}{\Delta;\Gamma \vdash u : A \mid u}\ \textsc{T-mVar} \qquad \frac{\Delta;\cdot \vdash M : A \mid t \qquad \Delta;\cdot \vdash q : s \stackrel{\triangleright}{=}_A t}{\Delta;\Gamma \vdash !_q M : [\![s]\!]\ A \mid !s}\ \textsc{T-Bang}$$

$$\frac{\Delta;\Gamma \vdash M : [\![r]\!]\ A \mid s \qquad \Delta, u :: A;\Gamma \vdash N : C \mid t}{\Delta;\Gamma \vdash \mathrm{let}(u^A := M, N) : C\,[r/u] \mid \mathrm{let}(u^A := s, t)}\ \textsc{T-Let}$$

$$\frac{\mathbf{d} \in \mathrm{dom}(\vartheta) \qquad \mathrm{dom}(\vartheta) = \mathrm{dom}(\theta) \qquad \left[\Delta;\cdot \vdash \vartheta(\psi) : \mathcal{T}^B(\psi) \mid \theta(\psi)\right]_{\psi \in \mathrm{dom}(\vartheta)}}{\Delta;\Gamma \vdash \iota(\vartheta) : B \mid \iota(\theta)}\ \textsc{T-TI}$$

$\boxed{\Delta;\Gamma \vdash q : s \stackrel{\triangleright}{=}_A t}$

$$\frac{\Delta;\Gamma \vdash s : A}{\Delta;\Gamma \vdash \mathbf{r}(s) : s \stackrel{\triangleright}{=}_A s}\ \textsc{Q-Refl} \qquad \frac{\Delta;\Gamma \vdash q_1 : r \stackrel{\triangleright}{=}_A s \qquad \Delta;\Gamma \vdash q_2 : s \stackrel{\triangleright}{=}_A t}{\Delta;\Gamma \vdash \mathbf{t}(q_1, q_2) : r \stackrel{\triangleright}{=}_A t}\ \textsc{Q-Trans}$$

$$\frac{\Delta;\Gamma, a : A \vdash s : B \qquad \Delta;\Gamma \vdash t : A}{\Delta;\Gamma \vdash \beta(a^A.s, t) : (\lambda a^A.s)\ t \stackrel{\triangleright}{=}_B s\,[t/a]}\ \textsc{Q-}\beta$$

$$\frac{\Delta;\cdot \vdash s : A \qquad \Delta, u :: A;\Gamma \vdash t : C}{\Delta;\Gamma \vdash \beta_{\Box}(s, u^A.t) : \mathrm{let}(u^A := !s, t) \stackrel{\triangleright}{=}_{C[s/u]} t\,[s/u]}\ \textsc{Q-}\beta_{\Box}$$

$$\frac{\mathbf{d} \in \mathrm{dom}(\vartheta) \qquad \Delta;\cdot \vdash q : s \stackrel{\triangleright}{=}_A t \qquad \left[\Delta;\cdot \vdash \theta(\psi) : \mathcal{T}^B(\psi)\right]_{\psi \in \mathrm{dom}(\theta)}}{\Delta;\Gamma \vdash \mathbf{ti}(q, \theta) : \iota(\theta) \stackrel{\triangleright}{=}_B q\theta}\ \textsc{Q-TI}$$

$$\frac{\Delta;\Gamma, a : A \vdash q : s \stackrel{\triangleright}{=}_B t}{\Delta;\Gamma \vdash \mathbf{lam}(a^A.q) : \lambda a^A.s \stackrel{\triangleright}{=}_{A \supset B} \lambda a^A.t}\ \textsc{Q-Abs}$$

$$\frac{\Delta;\Gamma \vdash q_1 : s_1 \stackrel{\triangleright}{=}_{A \supset B} t_1 \qquad \Delta;\Gamma \vdash q_2 : s_2 \stackrel{\triangleright}{=}_A t_2}{\Delta;\Gamma \vdash \mathbf{app}(q_1, q_2) : s_1\ s_2 \stackrel{\triangleright}{=}_B t_1\ t_2}\ \textsc{Q-App}$$

$$\frac{\Delta;\Gamma \vdash q_1 : s_1 \stackrel{\triangleright}{=}_{[\![r]\!]A} t_1 \qquad \Delta, u :: A;\Gamma \vdash q_2 : s_2 \stackrel{\triangleright}{=}_C t_2}{\Delta;\Gamma \vdash \mathbf{let}(q_1, u^A.q_2) : \mathrm{let}(u^A := s_1, s_2) \stackrel{\triangleright}{=}_{C[r/u]} \mathrm{let}(u^A := t_1, t_2)}\ \textsc{Q-Let}$$

$$\frac{\mathbf{d} \in \mathrm{dom}(\zeta) \qquad \mathrm{dom}(\zeta) = \mathrm{dom}(\theta) = \mathrm{dom}(\theta') \qquad \left[\Delta;\cdot \vdash \zeta(\psi) : \theta(\psi) \stackrel{\triangleright}{=}_{\mathcal{T}^B(\psi)} \theta'(\psi)\right]_{\psi \in \mathrm{dom}(\zeta)}}{\Delta;\Gamma \vdash \mathbf{trpl}(\zeta) : \iota(\theta) \stackrel{\triangleright}{=}_B \iota(\theta')}\ \textsc{Q-Trpl}$$

Figure 2 Typing rules for λ^{hc}.

The rule T-Bang, which typechecks audited terms, needs to compute the type of a trail: this is performed by the typechecking rules for trails. Each of these rules typechecks a different trail constructor: in particular, four rules define the contraction of redexes (Q-$\boldsymbol{\beta}$, Q-$\boldsymbol{\beta}_{\blacksquare}$, Q-TI), Q-Refl and Q-Trans ensure that trails induce a preorder on codes, and the remaining rules model congruence with respect to all code constructors but bangs. Let us remark that the trail q mentioned in trail inspections cannot be synthesized from the redex (since codes are not history-aware), but must be provided as an argument to **ti**.

We can prove that operations of Definitions 6 and 9 match the typing judgments:

▶ **Lemma 10.** *If* $\Delta;\Gamma \vdash q : s \stackrel{\triangleright}{=}_A t$ *then* $\mathrm{src}(q) = s$ *and* $\mathrm{tgt}(q) = t$.

▶ **Lemma 11.** *If* $\Delta;\Gamma \vdash M : A \mid s$ *then* $\mathrm{cd}(M) = s$.

2.5 Semantics

In this section, we define a reduction relation expressing how terms compute to values. Reduction differs from trails since it relates terms (rather than codes), and since unlike trails it does not appear as part of terms (however, it is defined in such a way that reduced terms will contain modified trails expressing a *log* of reduction).

Our definition of reduction makes use of contexts, i.e. terms with holes, represented by black boxes (■); similarly, we provide a notion of trail contexts, denoted by $\mathcal{Q}$. The notation $\mathcal{E}[M]$ indicates the term obtained by filling the hole in $\mathcal{E}$ with M; $\mathcal{Q}[q]$ is defined similarly.

$$\begin{array}{rcl} \mathcal{E} & ::= & \blacksquare \mid \lambda a^A.\mathcal{E} \mid (\mathcal{E}\ M) \mid (M\ \mathcal{E}) \mid !_q\mathcal{E} \mid \mathrm{let}(u^A := \mathcal{E}, M) \mid \mathrm{let}(u^A := M, \mathcal{E}) \mid \iota(\{\overrightarrow{M/\psi}, \mathcal{E}/\psi', \overrightarrow{N/\psi''}\}) \\ \mathcal{Q} & ::= & \blacksquare \mid \mathbf{t}(\mathcal{Q}, q) \mid \mathbf{t}(q, \mathcal{Q}) \mid \mathbf{app}(\mathcal{Q}, q) \mid \mathbf{app}(q, \mathcal{Q}) \mid \mathbf{lam}(a^A.\mathcal{Q}) \mid \mathbf{let}(\mathcal{Q}, u^A.q) \mid \mathbf{let}(q, u^A.\mathcal{Q}) \\ & \mid & \mathbf{trpl}(\{\overrightarrow{q_1/\psi}, \mathcal{Q}/\psi', \overrightarrow{q_2/\psi''}\}) \end{array}$$

Following [9] we use $\mathcal{F}$ to refer to contexts that do not allow holes inside bangs. Box-free contexts $\mathcal{F}$ and trail contexts $\mathcal{Q}$ are related by the following definition:

▶ **Definition 12** (canonical trail context)**.** For every $\mathcal{F}$, there exists a canonical trail context $\mathcal{Q}_{\mathcal{F}}$, defined as follows:

$$\begin{array}{rclrcl} \mathcal{Q}_{\blacksquare} & \triangleq & \blacksquare & \mathcal{Q}_{(\mathcal{F}\ M)} & \triangleq & \mathbf{app}(\mathcal{Q}_{\mathcal{F}}, \mathbf{r}(\mathrm{cd}(M))) \\ \mathcal{Q}_{(M\ \mathcal{F})} & \triangleq & \mathbf{app}(\mathbf{r}(\mathrm{cd}(M)), \mathcal{Q}_{\mathcal{F}}) & \mathcal{Q}_{\lambda a^A.\mathcal{F}} & \triangleq & \mathbf{lam}(a^A.\mathcal{Q}_{\mathcal{F}}) \\ \mathcal{Q}_{\mathrm{let}(\mathcal{F}, u^A.M)} & \triangleq & \mathbf{let}(\mathcal{Q}_{\mathcal{F}}, u.\mathbf{r}(\mathrm{cd}(M))) & \mathcal{Q}_{\mathrm{let}(M, u^A.\mathcal{F})} & \triangleq & \mathbf{let}(\mathbf{r}(\mathrm{cd}(M)), u^A.\mathcal{Q}_{\mathcal{F}}) \\ \multicolumn{6}{c}{\mathcal{Q}_{\iota(\{\overrightarrow{M/\psi}, \mathcal{F}/\psi', \overrightarrow{N/\psi''}\})} \triangleq \mathbf{trpl}(\{\overrightarrow{\mathbf{r}(\mathrm{cd}(M))/\psi}, \mathcal{Q}_{\mathcal{F}}/\psi'.\overrightarrow{\mathbf{r}(\mathrm{cd}(N))/\psi''}\})} \end{array}$$

Informally, $\mathcal{Q}_{\mathcal{F}}$ uses congruences to express a reflexive trail for $\mathcal{F}$, with a hole as the subtrail corresponding to the hole in $\mathcal{F}$. It is $\mathcal{Q}_{\mathcal{F}}$ that is responsible for producing the **app** congruence of Example 2.

Thanks to trail contexts and our definition of audited variable substitution, we can define reduction directly, without an auxiliary judgment pushing trails towards the closest outer bang. To avoid dealing with the unwanted situation of trail inspections not guarded by a bang, we only consider terms surrounded by an outer bang.

The reduction rules are defined in Figure 3. They operate by contracting a subterm appearing in a context $\mathcal{F}$, at the same time updating the trail of the enclosing bang by asserting that q is followed by a new contraction appearing in the trail context $\mathcal{Q}_{\mathcal{F}}$. Rule B-$\beta_{\square}$ is an exception, in that after performing the $\boldsymbol{\beta}_{\blacksquare}$ contraction, we still need to take into account the residuals of q', expressed by the substitution $N \times [(M, q', \mathrm{src}(q'))/u]$

We write $s \rightsquigarrow t$ when there exists a well-typed q such that $\mathrm{src}(q) = s$ and $\mathrm{tgt}(q) = t$.

$$\frac{}{!_q\mathcal{F}[(\lambda a^A.M)\ N)] \rightsquigarrow !_{\mathbf{t}(q,\mathcal{Q}_\mathcal{F}[\beta(a.\mathrm{cd}(M),\mathrm{cd}(N))])}\mathcal{F}[M\,[N/a]]}\ \text{B-}\beta \qquad \frac{M \rightsquigarrow N}{!_q\mathcal{F}[M] \rightsquigarrow !_q\mathcal{F}[N]}\ \text{B-BANG}$$

$$\frac{q_f = \mathbf{t}(q, \mathcal{Q}_\mathcal{F}[\mathbf{t}(\beta_\square(\mathrm{src}(q'), \mathrm{cd}(N)), N \times [(M, q', \mathrm{src}(q'))/u])])}{!_q\mathcal{F}[\mathrm{let}(!_{q'}M, u.N)] \rightsquigarrow !_{q_f}\mathcal{F}[N \ltimes [(M, q', \mathrm{src}(q'))/u]]}\ \text{B-}\beta_\square$$

$$\frac{}{!_q\mathcal{F}[\iota(\vartheta)] \rightsquigarrow !_{\mathbf{t}(q,\mathcal{Q}_\mathcal{F}[\mathbf{ti}(q,\mathrm{cd}(\vartheta))])}\mathcal{F}[q\vartheta]}\ \text{B-TI}$$

Figure 3 Term reduction rules for λ^{hc}.

▶ **Example 13.** We take again the term from Example 1 and reduce the outer let as follows:

$$!\ \mathrm{let}(u^{\mathbb{N}} := !_q 2, \mathrm{let}(v^{\mathbb{N}} := !_{q'} 6, u + v)) \rightsquigarrow !_{q_f}\ \ \mathrm{let}(v^{\mathbb{N}} := !_{q'} 6, 2 + v)$$

where we use the substitutions we computed in Example 8. Since the reduction happens immediately inside the bang, we use rule B-$\beta_\square$ with $\mathcal{Q}_\blacksquare = \blacksquare$, thus q_f is as follows:

$$\begin{aligned} q_f &= \mathbf{t}(\mathbf{r}, \mathbf{t}(\beta_\square(\mathrm{src}(q), u.\mathrm{cd}(\mathrm{let}(v := !_{q'} 6, u + v)), \mathbf{let}(\mathbf{r}, \mathbf{app}(\mathbf{app}(\mathbf{r}, q), \mathbf{r})))) \\ &= \mathbf{t}(\mathbf{r}, \mathbf{t}(\beta_\square(1 + 1, u.\,\mathrm{let}(v := !\ \mathsf{fact}\ 3, u + v)), \mathbf{let}(\mathbf{r}, \mathbf{app}(\mathbf{app}(\mathbf{r}, q), \mathbf{r})))) \end{aligned}$$

(where src and cd compute according to the hypotheses we made about q and q').

▶ **Example 14.** To show how to use the profiling inspection from Example 4, we need to assume a certain evaluation strategy: for example, call-by-value. We then write $(x \leftarrow M; N)$ as syntactic sugar for $(\lambda x.N)\ M$ and evaluate the following term:

$$\begin{aligned} &!\ (t_0 \leftarrow \iota(\vartheta_+); \\ &\quad \_ \leftarrow ((\lambda x, y, p.p\ x\ y)\ 2)\ 6; \\ &\quad t_1 \leftarrow \iota(\vartheta_+); \\ &\quad t_1 - t_0) \end{aligned}$$

In this term, t_0 evaluates to 0 because the bang starts with a reflexive trail; by the time we get to the third line, the outer trail contains a log of the inspection on the first line, the two beta reductions needed to evaluate the second line, and two more beta reductions to account for sequential compositions: thus t_1 evaluates to 5; finally, we evaluate $t_1 - t_0 = 5$.

▶ **Definition 15** (normal form)**.** A term M is in normal form iff for all trails q and all terms N, $!_q M \not\rightsquigarrow N$. Likewise, a code is in normal form iff there exists no $t \neq s$ such that $s \rightsquigarrow t$.

It should be noted that λ^{hc}, despite being strongly normalizing (as we will prove in the next section), is not confluent: the same term may reduce to different values under different reduction strategies. In particular, the trails appearing in bangs are sensitive to the reduction order: a call-by-value strategy and a call-by-name strategy will produce different trails. We could recover confluence by forcing a certain evaluation strategy and quotienting trails by means of a suitable equivalence relation: this is beyond the scope of the present paper.

The main properties of λ^{hc}, such as subject reduction, can be proved similarly to those for λ^h. As explained in Section 1, we also need to establish the consistency of trails as proofs of reducibility.

▶ **Theorem 16.** *For all codes s, t of λ^{hc} in normal form such that $s \neq t$, there exist no Δ, Γ, q, A such that $\Delta; \Gamma \vdash q : s \overset{\triangleright}{=}_A t$*

Proof. If such a judgment were provable, by structural induction on it we would be able to show that q must be a combination of reflexivity, transitivity, and congruence rules (since both s and t are in normal form, no contraction is possible in absence of symmetry). Then $s = t$, which falsifies the hypothesis. ◀

3 Strong Normalization

We now summarize our proof of strong normalization for λ^{hc}. The presence of a recursion operator on trails, whose occurrences can be arbitrarily nested, together with the fact that trails grow monotonically during execution, makes this result non-trivial.

Our proof employs the well-known notion of "candidates of reducibility" [13] (sets of strongly normalizing terms enjoying certain desirable properties). Candidates of reducibility are a powerful and flexible tool, which has been used to prove the strong normalization property of very expressive lambda calculi [12, 18, 25] and also other results such as the Church-Rosser property [11].

In the literature, it is possible to find several definitions of candidate of reducibility: along with Girard's definition, we can cite Tait's saturated sets [23] and Parigot's inductive definition [19]. Our proof employs Girard's candidates, and can be easily compared to the similar proof for the System F [14]. Some acquaintance with that proof will be assumed in the rest of this section. We will use $\mathcal{SN}$ to denote the set of all strongly normalizing terms. Reduction on strongly normalizing terms is a well-founded relation, which allows us to reason by well-founded induction on it.

3.1 A simplified calculus

As a technical means to investigate normalization of λ^{hc}, we define λ^{hs}, a simplified version of the calculus which forgets about trails associated with box introductions. Its types, terms, and contexts are defined by the following grammar:

$$\begin{array}{rcl} \tau, \sigma & ::= & P \mid \tau \supset \sigma \mid \Box\tau \\ s, t, \ldots & ::= & a \mid u \mid \lambda a^{\tau}.s \mid (s\ t) \mid !s \mid \mathrm{let}(u^{\tau} := s, t) \mid \iota(\theta) \\ \theta, \theta', \ldots & ::= & \{\overrightarrow{s/\psi}\} \\ \mathcal{E} & ::= & \blacksquare \mid \lambda a^{\tau}.\mathcal{E} \mid (\mathcal{E}\ s) \mid (s\ \mathcal{E}) \mid !\mathcal{E} \mid \mathrm{let}(u^{\tau} := \mathcal{E}, s) \mid \mathrm{let}(u^{\tau} := s, \mathcal{E}) \mid \iota(\{\overrightarrow{s/\psi}, \mathcal{E}/\psi', \overrightarrow{t/\psi''}\}) \end{array}$$

The terms of λ^{hs} correspond closely to the codes of λ^{hc}. Their semantics, however, is different: $\iota(\theta)$ does not perform inspection on the trail of the enclosing bang, but at the time of its evaluation will receive an arbitrary trail. We omit the definition of trails for brevity, but it can be obtained from the corresponding notion in λ^{hc}, by replacing codes with λ^{hs} terms.

We can also define simple and audited variable substitution on λ^{hs} terms, in a way that mimics the corresponding notion of λ^{hc} (Definition 5).

We provide an erasure map from λ^{hc} types and terms to λ^{hs} (its extension to contexts is immediate).

$$\begin{array}{ccccc} |P| = P & & |A \supset B| = |A| \supset |B| & & |[\![s]\!]\, A| = \Box|A| \\ |a| \triangleq a & |u| \triangleq u & |\lambda a^{A}.M| \triangleq \lambda a^{|A|}.|M| & |M\ N| \triangleq |M|\ |N| & \\ |!_q M| \triangleq !|M| & & |\,\mathrm{let}(u^{A} := M, N)| \triangleq \mathrm{let}(u^{A} := |M|, |N|) & |\iota(\vartheta)| \triangleq \iota(|\vartheta|) & \end{array}$$

The typing rules and reduction rules for λ^{hs} are given in Figures 4 and 5. They are similar to their counterparts in λ^{hc}, but greatly simplified: in particular, trail inspection is allowed to reduce by nondeterministically picking any trail.

Unsurprisingly, erasure preserves well-typedness and reduction:

$$\frac{a : \tau \in \Gamma}{\Delta;\Gamma \vdash a : \tau} \qquad \frac{u :: \tau \in \Delta}{\Delta;\Gamma \vdash u : \tau} \qquad \frac{\Delta;\Gamma, a : \tau \vdash M : \sigma}{\Delta;\Gamma \vdash \lambda a^{\tau}.M : \tau \supset \sigma} \qquad \frac{\Delta;\cdot \vdash M : \tau}{\Delta;\Gamma \vdash !M : \Box\tau}$$

$$\frac{\Delta;\Gamma \vdash M : \tau \supset \sigma \quad \Delta;\Gamma \vdash N : \tau}{\Delta;\Gamma \vdash M\ N : \sigma} \qquad \frac{\Delta;\Gamma \vdash M : \Box\tau \quad \Delta, u :: \tau;\Gamma \vdash N : \sigma}{\Delta;\Gamma \vdash \mathrm{let}(u^{\tau} := M, N) : \sigma} \qquad \frac{\mathbf{d} \in \mathrm{dom}(\theta) \quad [\Delta;\cdot \vdash \theta(\psi) : \mathcal{T}^{\sigma}(\psi)]_{\psi \in \mathrm{dom}(\theta)}}{\Delta;\Gamma \vdash \iota(\theta) : \sigma}$$

Figure 4 Typing rules for λ^{hs} terms.

$$((\lambda a.s)\ t) \rightsquigarrow s\,[t/a] \qquad \mathrm{let}(!s, u.t) \rightsquigarrow s\,[t/u] \qquad \iota(\theta) \rightsquigarrow q\theta \qquad \frac{s \rightsquigarrow t}{\mathcal{E}[s] \rightsquigarrow \mathcal{E}[t]}$$

Figure 5 Reduction rules for λ^{hs} terms.

▶ **Lemma 17.** *If* $\Delta;\Gamma \vdash_{\lambda^{hc}} M : A | s$*, then* $\Delta;\Gamma \vdash_{\lambda^{hs}} |M| : |A|$.

▶ **Lemma 18.** $M \rightsquigarrow_{\lambda^{hc}} N \Longrightarrow |M| \rightsquigarrow_{\lambda^{hs}} |N|$

By the combination of Lemma 17 and Lemma 18, we know that if λ^{hs} is strongly normalizing, then λ^{hc} must also, *a fortiori*, be strongly normalizing. We will thus proceed to prove SN in the simpler system, and extend it to λ^{hc} as a corollary.

3.2 Summary of the proof

We now give the definition of candidates of reducibility *à la Girard*.

▶ **Definition 19** (neutral term)**.** A term is neutral if it is not of the form $\lambda a^{A}.s$ or $!s$.

▶ **Definition 20** (candidates of reducibility)**.** A set $\mathcal{C}$ of terms is a *candidate of reducibility* iff it satisfies Girard's CR conditions:
CR1. $\mathcal{C} \subseteq \mathcal{SN}$
CR2. $s \in \mathcal{C} \wedge s \rightsquigarrow t \Longrightarrow t \in \mathcal{C}$
CR3. $s \in \mathcal{NT} \wedge (\forall t. s \rightsquigarrow t \Longrightarrow t \in \mathcal{C}) \Longrightarrow s \in \mathcal{C}$.
The set of candidates of reducibility will be denoted $\mathcal{CR}$.

We identify certain subsets of candidates that are stable wrt. validity substitution:

▶ **Definition 21** (substitutive sub-candidate)**.** For all candidates $\mathcal{C}$, validity variables u, and sets of terms $\mathcal{D}$, we define its substitutive subset $\mathcal{C}_u^{\mathcal{D}}$ as $\mathcal{C}_u^{\mathcal{D}} \triangleq \{s \in \mathcal{C} : \forall t \in \mathcal{D}, s\,[t/u] \in \mathcal{C}\}$.

Notice that sub-candidates are not in $\mathcal{CR}$; they do, however, satisfy CR1 and CR2, which is enough for us to use them in the definition of *reducible sets*.

▶ **Definition 22** (reducible set)**.** For all types τ, the set Red_{τ} of reducible terms of type τ is defined by recursion on τ as follows:

- $\mathsf{Red}_P = \mathcal{SN}$
- $\mathsf{Red}_{\tau\supset\sigma} = \{s : \forall t \in \mathsf{Red}_{\tau}, (s\ t) \in \mathsf{Red}_{\sigma}\}$
- $\mathsf{Red}_{\Box\tau} = \{s : \forall u, \forall \mathcal{C} \in \mathcal{CR}, \forall t \in \mathcal{C}_u^{\mathsf{Red}_{\tau}}, \mathrm{let}(u := s, t) \in \mathcal{C}\}$

In particular, $s \in \mathsf{Red}_{\Box\tau}$ if, and only if, for all reducibility candidates $\mathcal{C}$ and audited variables u, if we take a term $t \in \mathcal{C}_u^{\mathsf{Red}_{\tau}}$, then $\mathrm{let}(u := s, t) \in \mathcal{C}$. We are allowed to use Red_{τ} in $\mathcal{C}_u^{\mathsf{Red}_{\tau}}$ because τ is a subexpression of $\Box\tau$.

▶ **Lemma 23.** *For each type* τ*,* $\mathsf{Red}_{\tau} \in \mathcal{CR}$.

▶ **Theorem 24.** *Let* $\Delta, \Gamma, \overrightarrow{\eta}$ *s.t.* $\mathrm{dom}(\overrightarrow{\eta}) = \mathrm{dom}(\Delta) \cup \mathrm{dom}(\Gamma)$*, for all* $u \in \mathrm{dom}(\Delta)$, $\overrightarrow{\eta}(u) \in \mathsf{Red}_{\Delta(u)}$*, and for all* $a \in \mathrm{dom}(\Gamma)$, $\overrightarrow{\eta}(a) \in \mathsf{Red}_{\Gamma(a)}$*. Then,* $\Delta; \Gamma \vdash s : \tau$ *implies* $s\overrightarrow{\eta} \in \mathsf{Red}_\tau$*.*

Proof. We use the standard technique for System F [14], with additional subproofs for $s \in \mathsf{Red}_\tau \implies !s \in \mathsf{Red}_{\Box\tau}$ and $(\forall \psi \in \mathrm{dom}(\theta), \theta(\psi) \in \mathsf{Red}_{\mathcal{T}^\sigma(\psi)}) \implies \iota(\theta) \in \mathsf{Red}_\sigma$. SN follows as a corollary when $\overrightarrow{\eta}$ is the identity substitution. ◀

3.3 Formalization

We formalized λ^{hc}, together with our proof of strong normalization, in Nominal Isabelle. Nominal Isabelle mechanizes the management of variable binding, relieving the user from the burden of defining binding infrastructure (e.g. de Bruijn indices, lifting operations, etc.). On the other hand, many of the definitions used in a nominal formalization must be proved to be "well-behaved", beyond what would usually be made explicit in a pencil-and-paper proof. The main well-behavedness property required in Nominal Isabelle is *equivariance*, stating that a function f or a set $\mathcal{S}$ is stable under finite permutations of names π:

$$\forall x. f(\pi \cdot x) = \pi \cdot f(x) \qquad \forall x. x \in \mathcal{S} \iff \pi \cdot x \in \mathcal{S}$$

Most of the definitions used in the proof are equivariant, including typing and reduction rules, the set $\mathcal{SN}$ of strongly normalizing terms, the set $\mathcal{CR}$ of reducibility candidates, reducibility sets Red_A for all types A, and the operator $\mathcal{C}^{\mathcal{D}}_u$.

We do not, however, prove equivariance for individual reducibility candidates $\mathcal{C} \in \mathcal{CR}$: on the contrary, reducibility candidates do not need to be equivariant. To prove it, one can take a closure operator $[\cdot]$ mapping a set of strongly normalizing terms to the smallest reducibility candidate containing it (for its existence and definition see e.g. [22]): it is easy to see that, for all variables a, b, $!a \in [\{!b\}]$ if and only if $a = b$.

Our impredicative definition of $\mathsf{Red}_{\Box\tau}$, which quantifies over arbitrary candidates, is handled gracefully by Nominal Isabelle. Lindley and Stark [17] show a technique ($\top\top$-lifting) that can be adapted to provide a predicative definition of $\mathsf{Red}_{\Box A}$. As it happens, the lower logical complexity of $\top\top$-lifting relies on the additional definition of *continuations*, which would require some more effort to be formalized. On the other hand, our approach seems likely to extend to handle structural recursion (System T), following Aschieri and Zorzi [8], or impredicative polymorphism (System F), following Girard et al. [14].

4 Related work

The first Justification Logic-style system, known as the Logic of Proofs, was introduced by Artemov [6, 5]. Most work on justification logic systems (as for modal logic) is presented via Hilbert-style axiom systems extending classical propositional logic. Pfenning and Davies' judgemental reconstruction of modal logic [21] provides a natural deduction-style proof system for (intuitionistic) modal logic with necessity and possibility modalities. Artemov and Bonelli [7] introduced a system $\lambda^{\mathbf{I}}$ for justification logic, extending Pfenning and Davies' treatment of necessity ($\Box A$). They introduced equality proofs (trails) to recover subject reduction and proved strong normalization for $\lambda^{\mathbf{I}}$. Bavera and Bonelli later introduced (outer) trail inspection as part of an extended calculus called λ^h [9] from which we took inspiration.

Our system λ^{hc} seems (at least to us) an improvement over λ^h: it is simpler, and avoids the inconsistency arising from viewing trails as equivalence proofs. Nevertheless, the technique we used to prove strong normalization in λ^{hc} seems to suffice for λ^h as well. The only complication concerns the definition of reduction: while in λ^{hc} reduction acts non-locally by

updating the trail in the nearest enclosing bang, in λ^h reduction produces an intermediate term annotated with a new local trail, and the trail in the enclosing bang is updated after a sequence of *permutation reductions*:

$$!_q\mathcal{F}[M] \rightsquigarrow !_q\mathcal{F}[q' \triangleright M'] \overset{*}{\rightsquigarrow} !_{q''}\mathcal{F}[M']$$

It is thus necessary, though not overly complicated, to redefine reduction as a single-step operation including permutation reductions, and prove its equivalence to the original definition. This allows us to remove intermediate terms $q \triangleright M$ from the calculus altogether. In general, the (efficient) reduction theory of systems such as λ^h and λ^{hc} deserves further study.

Our work is also partly motivated by previous work on provenance and tracing for functional languages. Perera et al. [20] presented a pure, ML-like core language in which program execution yields both a value and a *trace*, corresponding roughly to the large-step operational derivation leading to the result. They gave trace slicing algorithms and techniques for extracting program slices and differential slices from traces; in subsequent work Acar et al. [3] explored security implications such as the disclosure and obfuscation properties of traces [10]. Indeed, trail inspection can be considered as a generic mechanism for defining *provenance views* as considered by these papers. Although Bavera and Bonelli [9] motivated λ^h as a basis for history-based access control (following Abadi and Fournet [1]), we are not aware of comparable work on provenance security based on justification logic. Our ongoing investigation suggests that λ^h-style trails contain enough information to extract many forms of provenance; however, to perform this extraction by means of trail inspection, we would usually need to reverse beta-reductions, and in particular to undo the substitution in beta-reduced terms. Since inspections treat beta and beta-box trails as black boxes, this cannot be achieved in the current version of the calculus. An extension of the language providing transparent beta trails and operations to undo substitutions is the subject of our current study.

Audit has been considered by a number of security researchers recently, for example Amir-Mohamedian et al. [4] consider correctness for audit logging, but auditing is again an extralinguistic operation (implemented using aspect-oriented programming). Vaughan et al. [24] introduced new formalisms for evidence-based audit. In Aura, an implementation of this approach [15], dependently-typed programs execute in the presence of a policy specified in authorization logic [2], and whenever a restricted resource is requested, a proof that access is authorized is constructed at run time and stored in an audit log for later inspection. The relationship between this approach and ours, and more generally between justification logic and authorization logic, remains to be investigated.

5 Conclusions

The motivation for this work is the need to provide better foundations for audited computation, as advocated in recent work on provenance and security and on type-theoretic forms of justification logic such as λ^h. However, as we have shown, λ^h is at the same time overly restrictive (in requiring affine variable and trail variable usage, i.e. forbidding copying of ordinary data) and overly permissive: despite these restrictions aimed at keeping the equational theory consistent, one can still prove any two compatibly-typed proof codes equivalent, using symmetry and the nondeterministic equational law for trail inspection.

We presented a new calculus λ^{hc} based on justification logic that includes audit operations such as trail inspection, but has fewer limitations and is simpler than λ^h. We show that λ^h avoids this problem and has a consistent reduction theory. We also prove strong normalization

for λ^{hc} via a simplified system λ^{hs}, and we have mechanically checked the proof. This approach also seems sufficient to prove SN for λ^{h}, though we have not formalized this result.

In future work, we intend to consider larger-scale programming languages based on the ideas of λ^{hc}, investigate connections to provenance tracking and slicing techniques, and clarify the security guarantees offered by justification logic-based audit.

References

1 M. Abadi and C. Fournet. Access control based on execution history. In *NDSS*, 2003.

2 Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst.*, 15(4), 1993. doi:10.1145/155183.155225.

3 U. A. Acar, A. Ahmed, J. Cheney, and R. Perera. A core calculus for provenance. *Journal of Computer Security*, 21:919–969, 2013. doi:10.1007/978-3-642-28641-4_22.

4 S. Amir-Mohammadian, S. Chong, and C. Skalka. Correct audit logging: Theory and practice. In *POST*, pages 139–162, 2016. doi:10.1007/978-3-662-49635-0_8.

5 S. Artemov. The logic of justification. *Review of Symbolic Logic*, 1(4):477–513, 2008. doi:10.1017/S1755020308090060.

6 S. N. Artëmov. Explicit provability and constructive semantics. *Bulletin of Symbolic Logic*, 7(1):1–36, 2001. doi:10.2307/2687821.

7 S. N. Artëmov and E. Bonelli. The intensional lambda calculus. In *Logical Foundations of Computer Science*, pages 12–25, 2007. doi:10.1007/978-3-540-72734-7_2.

8 Federico Aschieri and Margherita Zorzi. Non-determinism, non-termination and the strong normalization of System T. In *TLCA*, pages 31–47, 2013. doi:10.1007/978-3-642-38946-7_5.

9 F. Bavera and E. Bonelli. Justification logic and audited computation. *Journal of Logic and Computation*, 2015. Published online, June 19, 2015. doi:10.1093/logcom/exv037.

10 J. Cheney. A formal framework for provenance security. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium (CSF)*, pages 281–293. IEEE, 2011. doi:10.1109/CSF.2011.26.

11 J. H. Gallier. On Girard's "candidats de réductibilité". In *Logic and Computer Science*, pages 123–203. Academic Press, 1990.

12 H. Geuvers. A short and flexible proof of strong normalization for the calculus of constructions. In *Selected Papers from TYPES'94*, LNCS, pages 14–38. Springer-Verlag, 1995. doi:10.1007/3-540-60579-7_2.

13 J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In *Scandinavian Logic Symposium, 1978*, pages 63–92. North-Holland, 1971.

14 J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, New York, NY, USA, 1989.

15 L. Jia, J. A. Vaughan, K. Mazurak, J. Zhao, L. Zarko, J. Schorr, and S. Zdancewic. Aura: a programming language for authorization and audit. In *ICFP*, pages 27–38, 2008. doi:10.1145/1411204.1411212.

16 C. I. Lewis. *Symbolic Logic*. Dover Publications, 1959.

17 Sam Lindley and Ian Stark. Reducibility and $\top\top$-lifting for computation types. In *TLCA*, number 3461 in LNCS, pages 262–277. Springer-Verlag, 2005. doi:10.1007/11417170_20.

18 Z. Luo. *An Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1990.

19 M. Parigot. Strong normalization for the second order classical natural deduction. In *Proceedings of LICS*, 1994.

20 R. Perera, U. A. Acar, J. Cheney, and P. B. Levy. Functional programs that explain their work. In *ICFP*, pages 365–376. ACM, 2012. doi:10.1145/2398856.2364579.

21 F. Pfenning and R. Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001. doi:10.1017/S0960129501003322.

22 Colin Riba. On the stability by union of reducibility candidates. In *FOSSACS*, pages 317–331, Berlin, Heidelberg, 2007. Springer-Verlag. doi:10.1007/978-3-540-71389-0_23.

23 W. W. Tait. A realizability interpretation of the theory of species. In *Logic Colloquium, 1972–73*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer, 1975. doi:10.1007/BFb0064875.

24 J. A. Vaughan, L. Jia, K. Mazurak, and S. Zdancewic. Evidence-based audit. In *CSF*, pages 177–191, 2008. doi:10.1109/CSF.2008.24.

25 B. Werner. *Une Théorie des Constructions Inductives*. PhD thesis, Université Paris VII, 1994.

A The type system of λ^h

In Figures 6, 7, and 8, we provide the typing rules of Bavera and Bonelli's λ^h for reference, using syntax similar to that used in this paper.

A.1 Inconsistency of the equational theory of λ^h

In Section 1, we provided an informal derivation tree showing that the equational theory of λ^h allows to equate any two proof codes of the same type. We give below a more accurate (but isomorphic) derivation tree. Then, if we choose a θ such that $\theta(\mathbf{r}) = s$ and $\theta(\boldsymbol{\beta}) = t$, this effectively proves $s = t$, where s and t can be chosen freely:

$$\dfrac{\dfrac{\dfrac{\cdot;\cdot;\cdot \vdash \mathbf{r}(s') : s' =_B s'}{\cdot;\cdot;\cdot \vdash \theta : \mathcal{T}^A}\ \textsc{EqTI}}{\cdot;\cdot;\alpha : \mathsf{Eq}(A) \vdash \mathbf{ti}(\theta,\alpha) : \mathbf{r}(s')\theta =_A \alpha\theta}\ \textsc{EqTI} \qquad \dfrac{\dfrac{\cdot;\cdot;\cdot \vdash \boldsymbol{\beta}(b^{B'}.t', t'') : t'\left[t''/b\right] =_{B''} (\lambda b^{B'}.t')\ t''}{\cdot;\cdot;\cdot \vdash \theta : \mathcal{T}^A}\ \textsc{EqTI}}{\cdot;\cdot;\alpha : \mathsf{Eq}(A) \vdash \mathbf{ti}(\theta,\alpha) : \boldsymbol{\beta}(b^{B'}.t', t'')\theta =_A \alpha\theta}\ \textsc{EqSym}}{\cdot;\cdot;\alpha : \mathsf{Eq}(A) \vdash \mathbf{t}(\mathbf{ti}(\theta,\alpha), \mathbf{s}(\mathbf{ti}(\theta,\alpha))) : \mathbf{r}(s')\theta =_A \boldsymbol{\beta}(b^{B'}.t', t'')\theta}\ \textsc{EqTrans}$$

This derivation contains a lot of extraneous details, so we summarize the main features which contribute to the final result. In this derivation, we use transitivity on two subderivations. The right-hand subderivation shows that $s = \mathbf{r}(s')\theta$ is equal to $\alpha\theta$, using the rule EqTI. Here, s' is an arbitrary well-formed term of some type B. The EqTI rule allows us to conclude that any code of the form $q\theta$ is equal to $\alpha\theta$, and here we have used $q = \mathbf{r}(s')$.

In the second subderivation we use rule EqTI again to conclude that $t = \boldsymbol{\beta}(b^{B'}.t', t'')\theta$ is equivalent to $\alpha\theta$. Here, terms t' ad t'' are arbitrary well-formed terms of appropriate types to form a redex $\lambda b.t')\ t''$. We then use symmetry to show that $\alpha\theta$ is equivalent to $\boldsymbol{\beta}(b^{B'}.t', t'')\theta$. The desired conclusion follows. Since we can easily choose θ so that it returns one proof code s when handling a reflexivity trail and a different proof code t when handling a β-redex trail, the above derivation can be modified to prove that any two proof codes (of the same type A) are equivalent.

Finally, in applying the transitivity rule EqTrans, it is important that this rule does *not* require affine use of the trail variable α, so that it can be used in both subderivations. Changing this might fix the above problem, but would also have unknown consequences on the rest of the system. We have instead focused on an alternative system that does not require trail variables or affineness, by removing the symmetry trail.

$$\frac{a : A \in \Gamma}{\Delta;\Gamma;\Sigma \vdash a : A}\ \textsc{Var} \qquad \frac{u : A[\Sigma] \in \Delta \qquad \Sigma\sigma \subseteq \Sigma'}{\Delta;\Gamma;\Sigma' \vdash \langle u,\sigma\rangle : A}\ \textsc{mVar}$$

$$\frac{\Delta;\Gamma,a : A;\Sigma \vdash s : B}{\Delta;\Gamma \vdash \lambda a^A.s : A \supset B} \supset I \qquad \frac{\Delta;\Gamma_1;\Sigma_1 \vdash s : A \supset B \qquad \Delta;\Gamma_2;\Sigma_2 \vdash t : A}{\Delta;\Gamma_1,\Gamma_2;\Sigma_1,\Sigma_2 \vdash s\ t : B} \supset E$$

$$\frac{\Delta;\cdot;\Sigma \vdash s : A \qquad \Delta;\cdot;\Sigma \vdash q : s =_A t}{\Delta;\Gamma;\Sigma' \vdash \Sigma.t : [\![\Sigma.t]\!]\ A} \Box I \qquad \frac{\Delta;\Gamma_1;\Sigma_1 \vdash s : [\![\Sigma.r]\!]\ A \qquad \Delta,u : A[\Sigma];\Gamma_2;\Sigma_2 \vdash t : C}{\Delta;\Gamma_1,\Gamma_2;\Sigma_1,\Sigma_2 \vdash \mathrm{let}(u^{A[\Sigma]} := s,t) : C\,[\Sigma.r/u]} \Box E$$

$$\frac{\alpha : \mathsf{Eq}(A) \in \Sigma \qquad \Delta;\cdot;\cdot \vdash \theta : \mathcal{T}^B}{\Delta;\Gamma;\Sigma \vdash \alpha\theta : B}\ \textsc{TI} \qquad \frac{\Delta;\Gamma;\Sigma \vdash s : A \qquad \Delta;\Gamma;\Sigma \vdash q : s =_A t}{\Delta;\Gamma;\Sigma \vdash t : A}\ \textsc{Eq}$$

Figure 6 Typing rules for λ^h proof codes.

$$\frac{\Delta;\Gamma;\Sigma \vdash s : A}{\Delta;\Gamma;\Sigma \vdash \mathbf{r}(s) : s =_A s}\ \textsc{EqRefl} \qquad \frac{\Delta;\Gamma_1,a : A;\Sigma_1 \vdash s : A \supset B \qquad \Delta;\Gamma_2;\Sigma_2 \vdash t : A}{\Delta;\Gamma_1,\Gamma_2;\Sigma_1,\Sigma_2 \vdash \beta(a^A.s,t) : s\,[t/a] =_B (\lambda a^A.s)\ t}\ \textsc{Eq}\beta$$

$$\frac{\Delta;\cdot;\Sigma_1 \vdash A|r \qquad \Delta,u : A[\Sigma_1];\Gamma_2;\Sigma_2 \vdash C|t \qquad \Delta;\cdot;\Sigma_1 \vdash q : r =_A s \qquad \Gamma_2 \subseteq \Gamma_3 \quad \Sigma_2 \subseteq \Sigma_3}{\Delta;\Gamma_3;\Sigma_3 \vdash \beta_{\Box}(\Sigma_1.s,u^{A[\Sigma_1]}.t) : t\,[\Sigma_1.s/u] =_{C[\Sigma_1.s/u]} \mathrm{let}(u^{A[\Sigma_1]} := s,t)}\ \textsc{Eq}\beta_{\Box}$$

$$\frac{\Delta;\cdot;\Sigma_1 \vdash q : s =_A t \qquad \Delta;\cdot;\cdot \vdash \mathcal{T}^B|\theta \qquad \alpha : \mathsf{Eq}(A) \in \Sigma_2}{\Delta;\Gamma;\Sigma_2 \vdash \mathbf{ti}(\vartheta,\alpha) : q\theta =_B \alpha\theta}\ \textsc{EqTI}$$

$$\frac{\Delta;\Gamma;\Sigma \vdash q : s =_A t}{\Delta;\Gamma;\Sigma \vdash \mathbf{s}(q) : t =_A s}\ \textsc{EqSym} \qquad \frac{\Delta;\Gamma;\Sigma \vdash q_1 : r =_A s \qquad \Delta;\Gamma;\Sigma \vdash q_2 : s =_A t}{\Delta;\Gamma;\Sigma \vdash \mathbf{t}(q_1,q_2) : r =_A t}\ \textsc{EqTrans}$$

$$\frac{\Delta;\Gamma,a : A;\Sigma \vdash q : s =_B t}{\Delta;\Gamma;\Sigma \vdash \mathbf{lam}(a^A.q) : \lambda a^A.s =_{A\supset B} \lambda a^A.t}\ \textsc{EqAbs}$$

$$\frac{\Delta;\Gamma_1;\Sigma_1 \vdash q_1 : s_1 =_{A\supset B} t_1 \qquad \Delta;\Gamma_2;\Sigma_2 \vdash q_2 : s_2 =_A t_2}{\Delta;\Gamma_1,\Gamma_2;\Sigma_1,\Sigma_2 \vdash \mathbf{app}(q_1,q_2) : s_1\ s_2 =_B t_1\ t_2}\ \textsc{EqApp}$$

$$\frac{\Delta;\Gamma_1;\Sigma_1 \vdash q_1 : s_1 =_{[\![\Sigma_r]\!]A} t_1 \qquad \Delta,u : A[\Sigma];\Gamma_2;\Sigma_2 \vdash q_2 : s_2 =_C t_2}{\Delta;\Gamma_1,\Gamma_2;\Sigma_1,\Sigma_2 \vdash \mathbf{let}(q_1,u^{A[\Sigma]}.q_2) : \mathrm{let}(u^{A[\Sigma]} := s_1,s_2) =_{C[\Sigma.r/u]} \mathrm{let}(u^{A[\Sigma]} := t_1,t_2)}\ \textsc{EqLet}$$

$$\frac{\alpha : \mathsf{Eq}(A) \in \Sigma \qquad \Delta;\cdot;\cdot \vdash \overrightarrow{q} : \theta =_{\mathcal{T}^B} \theta'}{\Delta;\Gamma;\Sigma \vdash \mathbf{trpl}(\alpha,\overrightarrow{q}) : \alpha\theta =_B \alpha\theta'}\ \textsc{EqTrpl}$$

Figure 7 Typing rules for λ^h trails.

$$\frac{a : A \in \Gamma}{\Delta;\Gamma;\Sigma \vdash a : A \mid a}\ \textsc{TVar} \qquad \frac{u : A[\Sigma] \in \Delta \qquad \Sigma\sigma \subseteq \Sigma'}{\Delta;\cdot;\Sigma' \vdash \langle u,\sigma\rangle : A \mid \langle u,\sigma\rangle}\ \textsc{TmVar}$$

$$\frac{\Delta;\Gamma,a : A;\Sigma \vdash M : B \mid s}{\Delta;\Gamma;\Sigma \vdash \lambda a^A.M : A \supset B \mid \lambda a^A.s}\ \textsc{TAbs} \qquad \frac{\Delta;\Gamma_1;\Sigma_1 \vdash M : A \supset B \mid s \qquad \Delta;\Gamma_2;\Sigma_2 \vdash N : A \mid t}{\Delta;\Gamma_1,\Gamma_2;\Sigma_1,\Sigma_2 \vdash M\ N : B \mid s\ t}\ \textsc{TApp}$$

$$\frac{\Delta;\cdot;\Sigma \vdash M : A \mid s \qquad \Delta;\cdot;\Sigma \vdash q : s = t}{\Delta;\Gamma;\Sigma' \vdash !^{\Sigma}_{q} M : [\![\Sigma.t]\!]\ A \mid \Sigma.t}\ \textsc{TBox}$$

$$\frac{\Delta;\Gamma_1;\Sigma_1 \vdash M : [\![\Sigma.r]\!]\ A \mid s \qquad \Delta,u : A[\Sigma];\Gamma_2;\Sigma_2 \vdash N : C \mid t}{\Delta;\Gamma_1,\Gamma_2;\Sigma_1,\Sigma_2 \vdash \mathrm{let}(u^{A[\Sigma]} := M,N) : C\,[\Sigma.r/u] \mid \mathrm{let}(u^{A[\Sigma]} := s,t)}\ \textsc{TLet}$$

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

Figure 8 Typing rules for λ^h terms.

B Full definitions

B.1 Substitution on codes

For $\gamma = [t/a]$ and $\delta = [t/u]$, we define substitution of simple and audited variables on a code r (notation: $r\gamma$ and $r\delta$) as follows:

$$
\begin{aligned}
b\gamma &\triangleq \gamma(b) \\
v\gamma &\triangleq v \\
(\lambda b^A.s)\gamma &\triangleq \lambda b^A.s\gamma && (b\#a,t) \\
(s\ s')\gamma &\triangleq (s\gamma)\ (s'\gamma) \\
(!s)\gamma &\triangleq !s \\
\mathrm{let}(v^A := s, s')\gamma &\triangleq \mathrm{let}(v^A := s\gamma, s'\gamma) && (v\#t) \\
\iota(\theta)\gamma &\triangleq \iota(\theta\gamma)
\end{aligned}
$$

$$
\begin{aligned}
b\delta &\triangleq b \\
v\delta &\triangleq \delta(v) \\
(\lambda b^A.s)\delta &\triangleq \lambda b^A.s\delta && (b\#t) \\
(s\ s')\delta &\triangleq (s\delta)\ (s'\delta) \\
(!s)\delta &\triangleq !(s\delta) \\
\mathrm{let}(v^A := s, s')\delta &\triangleq \mathrm{let}(v^A := s\delta, s'\delta) && (v\#u,t) \\
\iota(\theta)\delta &\triangleq \iota(\theta\delta)
\end{aligned}
$$

where $\theta\gamma$ and $\theta\delta$ are defined pointwise and the notation $\gamma(b)$ is defined as t if $a = b$, and as b otherwise ($\delta(v)$ is defined similarly).

B.2 Source and target of a trail

The source and target of a trail q (notation: $\mathrm{src}(q)$ and $\mathrm{tgt}(q)$) are defined as follows:

$$
\begin{aligned}
\mathrm{src}(\mathbf{r}(s)) &\triangleq s \\
\mathrm{src}(\mathbf{t}(q_1,q_2)) &\triangleq \mathrm{src}(q_1) \\
\mathrm{src}(\boldsymbol{\beta}(a^A.s_1,s_2)) &\triangleq (\lambda a^A.s_1)\ s_2 \\
\mathrm{src}(\boldsymbol{\beta}_{\square}(s_1,v^A.s_2)) &\triangleq \mathrm{let}(v^A := !s_1, s_2) \\
\mathrm{src}(\mathbf{ti}(q',\theta)) &\triangleq \iota(\theta) \\
\mathrm{src}(\mathbf{lam}(a^A.q')) &\triangleq \lambda a^A.\mathrm{src}(q') \\
\mathrm{src}(\mathbf{app}(q',q'')) &\triangleq (\mathrm{src}(q')\ \mathrm{src}(q'')) \\
\mathrm{src}(\mathbf{let}(q',v^A.q'')) &\triangleq \mathrm{let}(v^A := \mathrm{src}(q'), \mathrm{src}(q'')) \\
\mathrm{src}(\mathbf{trpl}(\zeta)) &\triangleq \iota(\mathrm{src}(\zeta))
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{tgt}(\mathbf{r}(s)) &\triangleq s \\
\mathrm{tgt}(\mathbf{t}(q_1, q_2)) &\triangleq \mathrm{tgt}(q_2) \\
\mathrm{tgt}(\boldsymbol{\beta}(a^A.s_1, s_2)) &\triangleq s_1\,[s_2/a] \\
\mathrm{tgt}(\boldsymbol{\beta}_{\square}(s_1, v^A.s_2)) &\triangleq s_2\,[s_1/v] \\
\mathrm{tgt}(\mathbf{ti}(q', \theta)) &\triangleq q'\theta \\
\mathrm{tgt}(\mathbf{lam}(a^A.q')) &\triangleq \lambda a^A.\mathrm{tgt}(q') \\
\mathrm{tgt}(\mathbf{app}(q', q'')) &\triangleq (\mathrm{tgt}(q')\ \mathrm{tgt}(q'')) \\
\mathrm{tgt}(\mathbf{let}(q', v^A.q'')) &\triangleq \mathrm{let}(v^A := \mathrm{tgt}(q'), \mathrm{tgt}(q'')) \\
\mathrm{tgt}(\mathbf{trpl}(\zeta)) &\triangleq \iota(\mathrm{tgt}(\zeta))
\end{aligned}
$$

where $\mathrm{src}(\zeta)$ and $\mathrm{tgt}(\zeta)$ are defined pointwise.

B.3 Audited variable substitution

Given $\delta = [(N, q, t)/u]$, fix $\delta' = [t/u]$. Then, the operations $M \times \delta$ and $M \ltimes \delta$ are defined as follows:

$$
\begin{aligned}
a \times \delta &\triangleq \mathbf{r}(a) \\
v \times \delta &\triangleq \begin{cases} q & (u = v) \\ \mathbf{r}(v) & (u \neq v) \end{cases} \\
(\lambda a^A.R) \times \delta &\triangleq \mathbf{lam}(a^A.R \times \delta) && (\heartsuit) \\
(R\ S) \times \delta &\triangleq \mathbf{app}(R \times \delta, S \times \delta) \\
(!_{q'} R) \times \delta &\triangleq \mathbf{r}(!(\mathrm{src}(q')\delta')) \\
\mathrm{let}(v^A := R, S) \times \delta &\triangleq \mathbf{let}(R \times \delta, v^A.S \times \delta) && (\spadesuit) \\
\iota(\vartheta) \times \delta &\triangleq \mathbf{trpl}(\vartheta \times \delta)
\end{aligned}
$$

$$
\begin{aligned}
a \ltimes \delta &\triangleq a \\
v \ltimes \delta &\triangleq \begin{cases} N & (u = v) \\ v & (u \neq v) \end{cases} \\
(\lambda a^A.R) \ltimes \delta &\triangleq \lambda a^A.R \ltimes \delta && (\heartsuit) \\
(R\ S) \ltimes \delta &\triangleq (R \ltimes \delta)\ (S \ltimes \delta) \\
(!_{q'} R) \ltimes \delta &\triangleq !_{\mathbf{t}(q'\delta', R \times \delta)}(R \ltimes \delta) \\
\mathrm{let}(v^A := R, S) \ltimes \delta &\triangleq \mathrm{let}(v^A := R \ltimes \delta, u.S \ltimes \delta) && (\spadesuit) \\
\iota(\vartheta) \ltimes \delta &\triangleq \iota(\vartheta \ltimes \delta)
\end{aligned}
$$

$(\heartsuit)\quad a \# N, q, t \quad (\spadesuit)\quad v \# u, N, q, t$

where $\vartheta \times \delta$ and $\vartheta \ltimes \delta$ are defined pointwise on ϑ.

B.4 Codes

The code of a term M (notation: $\mathrm{cd}(M)$) is defined as follows:

$$\begin{aligned}
\mathrm{cd}(a) &\triangleq a \\
\mathrm{cd}(u) &\triangleq u \\
\mathrm{cd}(\lambda a^A.N) &\triangleq \lambda a^a.\mathrm{cd}(N) \\
\mathrm{cd}(N\ R) &\triangleq (\mathrm{cd}(N)\ \mathrm{cd}(R)) \\
\mathrm{cd}(!_q N) &\triangleq !\mathrm{src}(q) \\
\mathrm{cd}(\mathrm{let}(u^A := N, R)) &\triangleq \mathrm{let}(u^A := \mathrm{cd}(N), \mathrm{cd}(R)) \\
\mathrm{cd}(\iota(\vartheta)) &\triangleq \iota(\mathrm{cd}(\vartheta))
\end{aligned}$$

where $\mathrm{cd}(\vartheta)$ is defined pointwise.