Explicit Auditing

Wilmer Ricciotti and James Cheney

Laboratory for Foundations of Computer Science
University of Edinburgh
research@wilmer-ricciotti.net
jcheney@inf.ed.ac.uk

Abstract. The Calculus of Audited Units (CAU) is a typed lambda calculus resulting from a computational interpretation of Artemov's Justification Logic under the Curry-Howard isomorphism; it extends the simply typed lambda calculus by providing *audited types*, inhabited by expressions carrying a *trail* of their past computation history. Unlike most other auditing techniques, CAU allows the inspection of trails at runtime as a first-class operation, with applications in security, debugging, and transparency of scientific computation.
An efficient implementation of CAU is challenging: not only do the sizes of trails grow rapidly, but they also need to be normalized after every beta reduction. In this paper, we study how to reduce terms more efficiently in an untyped variant of CAU by means of explicit substitutions and explicit auditing operations.

1 Introduction

Transparency is an increasing concern in computer systems: for complex systems, whose desired behavior may be difficult to formally specify, auditing is an important complement to traditional techniques for verification and static analysis for security [2,4,10,26,18,15], program slicing [21,25], and provenance [20,23]. However, formal foundations of auditing as a programming language primitive are not yet well-established: most approaches view auditing as an extra-linguistic operation, rather than a first-class construct. Recently, however, Bavera and Bonelli [12] introduced calculus in which recording and analyzing audit trails are first-class operations. They proposed a λ-calculus based on a Curry-Howard correspondence with Justification Logic [6,7,5,8]. We refer to this system as the *calculus of audited units*, or **CAU**. In recent work, we developed a simplified form of **CAU** and proved strong normalization [24].

In **CAU**, audited units are handled using a type system based on modal logic, following Pfenning and Davies [22]. The type $[\![s]\!]\, A$ is the type of audited units, where s is "evidence", or the expression that was evaluated to produce the result of type A. Expressions of this type $!_q M$ contain a value of type A along with a "trail" q explaining how M was obtained by evaluating s. Trails are essentially (skeletons of) proofs of reduction of terms, which can be *inspected* by structural recursion using a special language construct.

To date, most work on foundations of auditing has focused on design, semantics, and correctness properties, and relatively little attention has been paid to efficient execution, while most work on auditing systems has neglected these foundational aspects. Some work on tracing and slicing has investigated the use of "lazy" tracing [21]; however, to the best of our knowledge there is no prior work on how to efficiently evaluate a language such as **CAU** in which auditing is a built-in operation. This is the problem studied in this paper.

A naïve approach to implementing the semantics of **CAU** as given by Bavera and Bonelli runs immediately into the following problem: a **CAU** reduction first performs a *principal contraction* (e.g. beta reduction), which typically introduces a local trail annotation describing the reduction, that can block further beta-reductions. The local trail annotations are then moved up to the nearest enclosing audited unit constructor using one or more *permutation reductions*. For example:

$$!_q\mathcal{F}[(\lambda x.M)\ N] \xrightarrow{\beta} !_q\mathcal{F}[\boldsymbol{\beta} \triangleright M\,\{N/x\}] \xrightarrow{\tau} !_{\mathbf{t}(q,\mathcal{Q}[\boldsymbol{\beta}])}\mathcal{F}[M\,\{N/x\}]$$

where $\mathcal{F}[]$ is a bang-free evaluation context and $\mathcal{Q}[\boldsymbol{\beta}]$ is a subtrail that indicates where in context $\mathcal{F}$ the $\boldsymbol{\beta}$-step was performed. As the size of the term being executed (and distance between an audited unit constructor and the redexes) grows, this evaluation strategy slows down quadratically in the worst case; eagerly materializing the traces likewise imposes additional storage cost.

While some computational overhead seems inevitable to accommodate auditing, both of these costs can in principle be mitigated. Trail permutations are computationally expensive and can often be delayed without any impact on the final outcome. Pushing trails to the closest outer bang does not serve any real purpose: it would be more efficient to keep the trail where it was created and perform normalization only if and when the trail must be inspected (and this operation does not even actually require an actual pushout of trails, because we can reuse term structure to compute the trail structure on-the-fly).

This situation has a well-studied analogue: in the λ-calculus, it is not necessarily efficient to eagerly perform all substitutions as soon as a β-reduction happens. Instead, calculi of *explicit substitutions* such as Abadi et al.'s $\lambda\sigma$ [1] have been developed in which substitutions are explicitly tracked and rewritten. Explicit substitution calculi have been studied extensively as a bridge between the declarative rewriting rules of λ-calculi and efficient implementations. Inspired by this work, we hypothesize that calculi with auditing can be implemented more efficiently by delaying the operations of trail extraction and erasure, using explicit symbolic representations for these operations instead of performing them eagerly. We envision this as a first step towards efficient abstract machines for calculi with tracing and trail inspection.

We do however need to make sure that the trails we produce still correctly describe the sequence of operations that were actually performed (e.g. respecting call-by-name or call-by-value reduction): when we perform a principal contraction, pre-existing trail annotations must be recorded as history that happened *before* the contraction, and not after. In the original eager reduction style, this is trivial because we never contract terms containing trails; however, to accommo-

date lazy treatment of trails, we need to be more careful. Accordingly, we will introduce explicit terms for *delayed trail erasure* $\lfloor M \rfloor$ and *delayed trail extraction* $\lceil M \rceil$. We can use these features to decrease the cost of normalization: for instance, the β-reduction above can be replaced by a rule with delayed treatment of substitution and trails, denoted by Beta:

$$!_q\mathcal{F}[(\lambda.M)\ N] \xrightarrow{\text{Beta}} !_q\mathcal{F}[\mathbf{t}(\mathbf{app}(\mathbf{lam}(\lceil M \rceil), \lceil N \rceil), \boldsymbol{\beta}) \triangleright \lfloor M \rfloor\ [\lfloor N \rfloor]]$$

Here, we use de Bruijn notation [13] (as in $\lambda\sigma$, and anticipating Sections 3 and 4), and write $M[N]$ for the explicit substitution of N for the outermost bound variable of $\lambda.M$. The trail constructor $\mathbf{t}$ stands for transitive composition of trails, while $\mathbf{app}$ and $\mathbf{lam}$ are congruence rules on trails, so the trail $\mathbf{t}(\mathbf{app}(\mathbf{lam}(\lceil M \rceil), \lceil N \rceil), \boldsymbol{\beta})$ says that the redex's trail is constructed by extracting the latent trail information from M and N, combining it appropriately, and then performing a $\boldsymbol{\beta}$ step. The usual contractum itself is obtained by substituting the erased argument $\lfloor N \rfloor$ into the erased function body $\lfloor M \rfloor$. Although this may look a bit more verbose than the earlier beta-reduction, the additional work done to create the trail $\mathbf{t}(\mathbf{app}(\mathbf{lam}(\lceil M \rceil), \lceil N \rceil), \boldsymbol{\beta})$ is all work that would have been done anyway using the eager system, while the use of lazy trail-extraction and trail-erasure operations gives us many more ways to do the remaining work efficiently — for example, if the trail is never subsequently used, we can just discard it without doing any more work.

Contributions. We study an extension of Abadi et al.'s calculus $\lambda\sigma$ [1] with explicit auditing operations. We consider a simplified, untyped variant $\mathbf{CAU}^-$ of the Calculus of Audited Units (Section 2); this simplifies our presentation because type information is not needed during execution. We revisit $\lambda\sigma$ in Section 3, extend it to include auditing and trail inspection features, and discuss problems with this initial, naïve approach. We also (Section 4) introduce explicit versions of the "trail extraction" and "trail erasure" operations, extend the rewriting theory to account for them: as our main result, we show that the resulting system $\mathbf{CAU}^-_\sigma$ correctly refines the untyped $\mathbf{CAU}$ (subject to an obvious translation). We evaluate the calculus (Section 5) illustrating how the resulting system can be asymptotically more efficient than a naïve implementation of the $\mathbf{CAU}$ rules.

2 The Untyped Calculus of Audited Units

The language $\mathbf{CAU}^-$ presented here is an untyped version of the calculi λ^h [12] and Ricciotti and Cheney λ^{hc} [24] obtained by erasing all typing information and a few other related technicalities: this will allow us to address all the interesting issues related to the reduction of $\mathbf{CAU}$ terms, but with a much less pedantic syntax. To help us explain the details of the calculus, we adapt some examples from our previous paper [24]; other examples are described by Bavera and Bonelli [12].

Unlike the typed variant of the calculus, we only need one sort of variables, denoted by the letters $x, y, z \ldots$. The syntax of $\mathbf{CAU}^-$ is as follows:

$$\textbf{Terms } M, N ::= x \mid \lambda x.M \mid M\ N \mid \mathrm{let}_!(x := M, N) \mid !_q M \mid q \triangleright M \mid \iota(\vartheta)$$
$$\textbf{Trails } q, q' ::= \mathbf{r} \mid \mathbf{t}(q, q') \mid \boldsymbol{\beta} \mid \boldsymbol{\beta}_! \mid \mathbf{ti} \mid \mathbf{lam}(q) \mid \mathbf{app}(q, q') \mid \mathbf{let}_!(q, q') \mid \mathbf{tb}(\zeta)$$

CAU$^-$ extends the pure lambda calculus with *audited units* $!_q M$ (colloquially, "bang M"), whose purpose is to decorate the term M with a log q of its computation history, callezd *trail* in our terminology: when M evolves as a result of computation, q will be updated by adding information about the reduction rules that have been applied. The form $!_q M$ is in general not intended for use in source programs: instead, we will write $!\ M$ for $!_{\mathbf{r}} M$, where $\mathbf{r}$ represents the empty execution history (*reflexivity* trail).

Audited units can then be employed in larger terms by means of the " let-bang" operator, which unpacks an audited unit and thus allows to access its contents. The variable declared by a $\mathrm{let}_!$ is bound in its second argument: in essence $\mathrm{let}_!(x := !_q M, N)$ will reduce to N, where free occurrences of x have been replaced by M; the trail q will not be discarded, but will be used to produce a new trail explaining this reduction.

The expression form $q \triangleright M$ is an auxiliary, intermediate annotation of M with partial history information which is produced during execution and will eventually stored in the closest surrounding bang.

Example 1. In **CAU**$^-$ we can express history-carrying terms explicitly: for instance, if we use $\bar{n}$ to express the Church encoding of a natural number n, and *plus* or *fact* for lambda terms computing addition and factorial on said representation, we can write audited units like

$$!_q \bar{2} \qquad !_{q'} \bar{6}$$

where q is a trail representing the history of $\bar{2}$ i.e., for instance, a witness for the computation that produced $\bar{2}$ by reducing *plus* $\bar{1}$ $\bar{1}$; likewise, q' might describe how computing *fact* $\bar{3}$ produced $\bar{6}$.

Supposing we wish to add these two numbers together, at the same time retaining their history, we will use the $\mathrm{let}_!$ construct to look inside them:

$$\mathrm{let}_!(x := !_q \bar{2}, \mathrm{let}_!(y := !'_q \bar{6}, \mathit{plus}\ x\ y)) \longrightarrow\!\!\!\!\rightarrow q'' \triangleright \bar{8}$$

where the final trail q'' is produced by composing q and q'; if this reduction happens inside an external bang, q'' will eventually be captured by it.

Trails, representing sequences of reduction steps, encode the (possibly partial) computation history of a given subterm. The main building blocks of trails are $\boldsymbol{\beta}$ (representing standard beta reduction), $\boldsymbol{\beta}_!$ (contraction of a let-bang redex) and $\mathbf{ti}$ (denoting the execution of a trail inspection). For every class of terms we have a corresponding congruence trail ($\mathbf{lam}, \mathbf{app}, \mathbf{let}_!, \mathbf{tb}$, the last of which is associated with trail inspections), with the only exception of bangs, which do not need a congruence rule because they capture all the computation happening inside them. The syntax of trails is completed by reflexivity $\mathbf{r}$ (representing a null computation history, i.e. a term that has not reduced yet) and transitivity

t (i.e. sequential composition of execution steps). As discussed by our earlier paper [24], we omit Bavera and Bonelli's symmetry trail form.

The last term form $\iota(\vartheta)$, called *trail inspection*, will perform primitive recursion on the computation history of the current audited unit. The metavariables ϑ and ζ associated with trail inspections are *trail replacements*, i.e. maps associating to each possible trail constructor, respectively, a term or a trail:

$$\vartheta ::= \{M_1/\mathbf{r}, M_2/\mathbf{t}, M_3/\boldsymbol{\beta}, M_4/\boldsymbol{\beta}_!, M_5/\mathbf{ti}, M_6/\mathbf{lam}, M_7/\mathbf{app}, M_8/\mathbf{let}_!, M_9/\mathbf{tb}\}$$
$$\zeta ::= \{q_1/\mathbf{r}, q_2/\mathbf{t}, q_3/\boldsymbol{\beta}, q_4/\boldsymbol{\beta}_!, q_5/\mathbf{ti}, q_6/\mathbf{lam}, q_7/\mathbf{app}, q_8/\mathbf{let}_!, q_9/\mathbf{tb}\}$$

When the trail constructors are irrelevant for a certain ϑ or ζ, we will omit them, using the notations $\{\overrightarrow{M}\}$ or $\{\overrightarrow{q}\}$. These constructs represent (or describe) the nine cases of a structural recursion operator over trails, which we write as $q\vartheta$.

Definition 1. *The operation $q\vartheta$, which produces a term by structural recursion on q applying the inspection branches ϑ, is defined as follows:*

$$\begin{array}{rl}
\mathbf{r}\vartheta \triangleq \vartheta(\mathbf{r}) & \mathbf{t}(q,q')\vartheta \triangleq \vartheta(\mathbf{t})\ (q\vartheta)\ (q'\vartheta) \\
\boldsymbol{\beta}\vartheta \triangleq \vartheta(\boldsymbol{\beta}) & \boldsymbol{\beta}_!\vartheta \triangleq \vartheta(\boldsymbol{\beta}_!) \\
\mathbf{ti}\vartheta \triangleq \vartheta(\mathbf{ti}) & \mathbf{lam}(q)\vartheta \triangleq \vartheta(\mathbf{lam})\ (q\vartheta) \\
\mathbf{app}(q,q')\vartheta \triangleq \vartheta(\mathbf{app})\ (q\vartheta)\ (q'\vartheta) & \mathbf{let}_!(q,q')\vartheta \triangleq \vartheta(\mathbf{let}_!)\ (q\vartheta)\ (q'\vartheta) \\
\mathbf{tb}(\{\overrightarrow{q}\})\vartheta \triangleq \vartheta(\mathbf{tb})\ \overrightarrow{(q\vartheta)} &
\end{array}$$

where the sequence $\overrightarrow{(q\vartheta)}$ is obtained from $\overrightarrow{q}$ by pointwise recursion.

Example 2. Trail inspection can be used to count all of the contraction steps in the history of an audited unit, by means of the following trail replacement:

$$\vartheta_+ ::= \{\bar{0}/\mathbf{r}, \mathit{plus}/\mathbf{t}, \bar{1}/\boldsymbol{\beta}, \bar{1}/\boldsymbol{\beta}_!, \bar{1}/\mathbf{ti}, \lambda x.x/\mathbf{lam}, \mathit{plus}/\mathbf{app}, \mathit{plus}/\mathbf{let}_!, \mathit{sum}/\mathbf{tb}\}$$

where *sum* is a variant of *plus* taking nine arguments, as required by the arity of **tb**. For example, we can count the contractions in $q = \mathbf{t}(\mathbf{let}_!(\boldsymbol{\beta}, \mathbf{r}), \boldsymbol{\beta}_!)$ as:

$$q\vartheta_+ = \mathit{plus}\ (\mathit{plus}\ \bar{1}\ \bar{0})\ \bar{1}$$

2.1 Reduction

Reduction in $\mathbf{CAU}^-$ includes rules to contract the usual beta redexes contracting an applied lambda abstraction, "beta-bang" redexes, which unpack the bang term appearing as the definiens of a $\mathrm{let}_!$, and trail inspections. These rules, which we call *principal contractions*, are defined as follows:

$$(\lambda x.M)\ N \xrightarrow{\beta} \boldsymbol{\beta} \triangleright M\{N/x\} \qquad \mathrm{let}_!(x := !_q M, N) \xrightarrow{\beta} \boldsymbol{\beta}_! \triangleright N\{q \triangleright M/x\}$$
$$!_q\mathcal{F}[\iota(\vartheta)] \xrightarrow{\beta} !_q\mathcal{F}[\mathbf{ti} \triangleright q\vartheta]$$

Substitution $M\{N/x\}$ is defined in the traditional way, avoiding variable capture. The first contraction is familiar, except for the fact that the reduct $M\{N/x\}$

has been annotated with a $\boldsymbol{\beta}$ trail. The second one deals with unpacking a bang: from $!_q M$ we obtain $q \triangleright M$, which is then substituted for x in the target term N; the resulting term is annotated with a $\boldsymbol{\beta_!}$ trail. The third contraction defines the result of a trail inspection $\iota(\vartheta)$. Trail inspection will be contracted by capturing the current history, as stored in the nearest enclosing bang, and performing structural recursion on it according to the branches defined by ϑ. The concept of "nearest enclosing bang" is made formal by contexts $\mathcal{F}$ in which the hole cannot appear inside a bang (or *bang-free* contexts, for short):

$$\mathcal{F} ::= \blacksquare \mid \lambda x.\mathcal{F} \mid \mathcal{F}\ M \mid M\ \mathcal{F} \mid \mathrm{let}_!(\mathcal{F}, M) \mid \mathrm{let}_!(M, \mathcal{F}) \mid q \triangleright \mathcal{F} \mid \iota(\{\overrightarrow{M}, \mathcal{F}, \overrightarrow{N}\})$$

The definition of the principal contractions is completed, as usual, by a contextual closure rule stating that they can appear in any context $\mathcal{E}$:

$$\frac{M \xrightarrow{\beta} N}{\mathcal{E}[M] \xrightarrow{\beta} \mathcal{E}[N]} \qquad \begin{aligned} \mathcal{E} ::= \blacksquare &\mid \lambda x.\mathcal{E} \mid \mathcal{E}\ M \mid M\ \mathcal{E} \mid \mathrm{let}_!(\mathcal{E}, M) \\ &\mid \mathrm{let}_!(M, \mathcal{E}) \mid !_q \mathcal{E} \mid q \triangleright \mathcal{E} \mid \iota(\{\overrightarrow{M}, \mathcal{E}, \overrightarrow{N}\}) \end{aligned}$$

The principal contractions introduce local trail subterms $q' \triangleright M$, which can block other reductions. Furthermore, the rule for trail inspection assumes that the q annotating the enclosing bang really is a complete log of the history of the audited unit; but at the same time, it violates this invariant, because the **ti** trail created after the contraction is not merged with the original history q.

For these reasons, we only want to perform principal contractions on terms not containing local trails: after each principal contraction, we apply the following rewrite rules, called *permutation reductions*, to ensure that the local trail is moved to the nearest enclosing bang:

$$\begin{aligned} \mathbf{r} \triangleright M &\xrightarrow{\tau} M & q \triangleright (q' \triangleright M) &\xrightarrow{\tau} \mathbf{t}(q, q') \triangleright M \\ !_q(q' \triangleright M) &\xrightarrow{\tau} !_{\mathbf{t}(q,q')} M & \lambda x.(q \triangleright M) &\xrightarrow{\tau} \mathbf{lam}(q) \triangleright \lambda x.M \\ (q \triangleright M)\ N &\xrightarrow{\tau} \mathbf{app}(q, \mathbf{r}) \triangleright M\ N & M\ (q \triangleright N) &\xrightarrow{\tau} \mathbf{app}(\mathbf{r}, q) \triangleright M\ N \\ \mathrm{let}_!(x := q \triangleright M, N) &\xrightarrow{\tau} \mathbf{let}_!(q, \mathbf{r}) \triangleright \mathrm{let}_!(x := M, N) \\ \mathrm{let}_!(x := M, q \triangleright N) &\xrightarrow{\tau} \mathbf{let}_!(\mathbf{r}, q) \triangleright \mathrm{let}_!(x := M, N) \\ \iota(\{M_1, \ldots, q \triangleright M_i, \ldots, M_9\}) &\xrightarrow{\tau} \mathbf{tb}(\{\mathbf{r}, \ldots, q, \ldots, \mathbf{r}\}) \triangleright \iota(\{\overrightarrow{M_9}\}) \end{aligned}$$

Moreover, the following rules are added to the $\xrightarrow{\tau}$ relation to ensure confluence:

$$\begin{aligned} \mathbf{lam}(\mathbf{r}) &\xrightarrow{\tau} \mathbf{r} & \mathbf{app}(\mathbf{r}, \mathbf{r}) &\xrightarrow{\tau} \mathbf{r} & \mathbf{t}(\mathbf{r}, q) &\xrightarrow{\tau} q \\ \mathbf{let}_!(\mathbf{r}, \mathbf{r}) &\xrightarrow{\tau} \mathbf{r} & \mathbf{tb}(\{\overrightarrow{\mathbf{r}}\}) &\xrightarrow{\tau} \mathbf{r} & \mathbf{t}(q, \mathbf{r}) &\xrightarrow{\tau} q \\ \mathbf{t}(\mathbf{t}(q_1, q_2), q_3) &\xrightarrow{\tau} \mathbf{t}(q_1, \mathbf{t}(q_2, q_3)) \\ \mathbf{t}(\mathbf{lam}(q), \mathbf{lam}(q')) &\xrightarrow{\tau} \mathbf{lam}(\mathbf{t}(q, q')) \\ \mathbf{t}(\mathbf{lam}(q_1), \mathbf{t}(\mathbf{lam}(q_1'), q)) &\xrightarrow{\tau} \mathbf{t}(\mathbf{lam}(\mathbf{t}(q_1, q_1')), q) \\ \mathbf{t}(\mathbf{app}(q_1, q_2), \mathbf{app}(q_1', q_2')) &\xrightarrow{\tau} \mathbf{app}(\mathbf{t}(q_1, q_1'), \mathbf{t}(q_2, q_2')) \\ \mathbf{t}(\mathbf{app}(q_1, q_2), \mathbf{t}(\mathbf{app}(q_1', q_2')), q) &\xrightarrow{\tau} \mathbf{t}(\mathbf{app}(\mathbf{t}(q_1, q_1'), \mathbf{t}(q_2, q_2')), q) \\ \mathbf{t}(\mathbf{let}_!(q_1, q_2), \mathbf{let}_!(q_1', q_2')) &\xrightarrow{\tau} \mathbf{let}_!(\mathbf{t}(q_1, q_1'), \mathbf{t}(q_2, q_2')) \\ \mathbf{t}(\mathbf{let}_!(q_1, q_2), \mathbf{t}(\mathbf{let}_!(q_1', q_2')), q) &\xrightarrow{\tau} \mathbf{t}(\mathbf{let}_!(\mathbf{t}(q_1, q_1'), \mathbf{t}(q_2, q_2')), q) \\ \mathbf{t}(\mathbf{tb}(\overrightarrow{q_1}), \mathbf{tb}(\overrightarrow{q_2})) &\xrightarrow{\tau} \mathbf{tb}(\overrightarrow{\mathbf{t}(q_1, q_2)}) \\ \mathbf{t}(\mathbf{tb}(\overrightarrow{q_1}), \mathbf{t}(\mathbf{tb}(\overrightarrow{q_2}), q)) &\xrightarrow{\tau} \mathbf{t}(\mathbf{tb}(\overrightarrow{\mathbf{t}(q_1, q_2)}), q) \end{aligned}$$

As usual, $\xrightarrow{\tau}$ is completed by a contextual closure rule. We prove

Lemma 1 ([12]). *$\xrightarrow{\tau}$ is terminating and confluent.*

When a binary relation $\xrightarrow{\mathcal{R}}$ on terms is terminating and confluent, we will write $\mathcal{R}(M)$ for the unique $\mathcal{R}$-normal form of M. Since principal contractions must be performed on τ-normal terms, it is convenient to merge contraction and τ-normalization in a single operation, which we will denote by $\xrightarrow{\mathbf{CAU}^-}$:

$$\frac{M \xrightarrow{\beta} N}{M \xrightarrow{\mathbf{CAU}^-} \tau(N)}$$

3 Naïve explicit substitutions

We seek to adapt the existing abstract machines for the efficient normalization of lambda terms to $\mathbf{CAU}^-$. Generally speaking, most abstract machines act on nameless terms, using de Bruijn's indices [13], thus avoiding the need to perform renaming to avoid variable capture when substituting a term into another.

Moreover, since a substitution $M\{N/x\}$ requires to scan the whole term M and is thus *not* a constant time operation, it is usually not executed immediately in an eager way. The abstract machine actually manipulates *closures*, or pairs of a term M and an environment s declaring lazy substitutions for each of the free variables in M: this allows s to be applied in an incremental way, while scanning the term M in search for a redex. In the $\lambda\sigma$-calculus of Abadi et al. [1], lazy substitutions and closures are manipulated explicitly, providing an elegant bridge between the classical λ-calculus and its concrete implementation in abstract machines. Their calculus expresses beta reduction as the rule

$$(\lambda.M)\ N \longrightarrow M[N]$$

where $\lambda.M$ is a nameless abstraction *à la de Bruijn*, and $[N]$ is a (suspended) *explicit substitution* mapping the variable corresponding to the first dangling index in M to N, and all the other variables to themselves. Terms in the form $M[s]$, representing closures, are syntactically part of $\lambda\sigma$, as opposed to substitutions $M\{N/x\}$, which are meta-operations that *compute* a term.

In this section we formulate a first attempt at adding explicit substitutions to $\mathbf{CAU}^-$. We will not prove any formal result for the moment, as our purpose is to elicit the difficulties of such a task. An immediate adaptation of $\lambda\sigma$-like explicit substitutions yields the following syntax:

Terms	M, N	$::= 1 \mid \lambda.M \mid M\ N \mid \mathrm{let}_!(M, N) \mid !_q M \mid q \triangleright M \mid \iota(\vartheta) \mid M[s]$
Substitutions	s, t	$::= \langle\rangle \mid \uparrow \mid s \circ t \mid M \cdot s$

where 1 is the first de Bruijn index, the nameless λ binds the first free index of its argument, and similarly the nameless $\mathrm{let}_!$ binds the first free index of its second argument. Substitutions include the identity (or empty) substitution $\langle\rangle$, lift $\uparrow$

(morally replacing all free indices n with their successor $n+1$), the composition $s \circ t$ (equivalent to the sequencing of s and t) and finally $M \cdot s$ (indicating a substitution that will replace the first free index with M, and other indices n with their predecessor $n-1$ under substitution s). Trails are unchanged.

We write $M[N_1 \cdots N_k]$ as syntactic sugar for $M[N_1 \cdots N_k \cdot \langle\rangle]$. Then, $\mathbf{CAU}^-$ reductions can be expressed as follows:

$$(\lambda.M)\ N \xrightarrow{\beta} \boldsymbol{\beta} \triangleright M[N] \qquad \mathrm{let}_!(!_q M, N) \xrightarrow{\beta} \boldsymbol{\beta_!} \triangleright N[q \triangleright M]$$
$$!_q \mathcal{F}[\iota(\vartheta)] \xrightarrow{\beta} !_q \mathcal{F}[\mathbf{ti} \triangleright q\vartheta]$$

(trail inspection, which does not use substitutions, is unchanged). The idea is that explicit substitutions make reduction more efficient because their evaluation does not need to be performed all at once, but can be delayed, partially or completely; delayed explicit substitutions applied to the same term can be merged, so that the term does not need to be scanned twice. The evaluation of explicit substitution can be defined by the following σ-rules:

$$\begin{array}{rlrl}
1[\langle\rangle] & \xrightarrow{\sigma} 1 & \langle\rangle \circ s & \xrightarrow{\sigma} s \\
1[M \cdot s] & \xrightarrow{\sigma} M & \uparrow \circ \langle\rangle & \xrightarrow{\sigma} \uparrow \\
(\lambda M)[s] & \xrightarrow{\sigma} \lambda(M[1 \cdot (s \circ \uparrow)]) & \uparrow \circ (M \cdot s) & \xrightarrow{\sigma} s \\
(M\ N)[s] & \xrightarrow{\sigma} M[s]\ N[s] & (M \cdot s) \circ t & \xrightarrow{\sigma} M[t] \cdot (s \circ t) \\
(!_q M)[s] & \xrightarrow{\sigma} !_q(M[s]) & (s_1 \circ s_2) \circ s_3 & \xrightarrow{\sigma} s_1 \circ (s_2 \circ s_3) \\
\mathrm{let}_!(M, N)[s] & \xrightarrow{\sigma} \mathrm{let}_!(M, N[1 \cdot (s \circ \uparrow)]) & (q \triangleright M)[s] & \xrightarrow{\sigma} q \triangleright (M[s]) \\
\iota(\{\overrightarrow{M}\})[s] & \xrightarrow{\sigma} \iota(\{\overrightarrow{M[s]}\}) & M[s][t] & \xrightarrow{\sigma} M[s \circ t]
\end{array}$$

These rules are a relatively minor adaptation from those of $\lambda\sigma$: as in that language, σ-normal forms do not contain explicit substitutions, save for the case of the index 1, which may be lifted multiple times, e.g.:

$$1[\uparrow^n] = 1[\underbrace{\uparrow \circ \cdots \circ \uparrow}_{n \text{ times}}]$$

If we take $1[\uparrow^n]$ to represent the de Bruijn index $n+1$, as in $\lambda\sigma$, σ-normal terms coincide with a nameless representation of $\mathbf{CAU}^-$.

The σ-rules are deferrable, in that we can perform β-reductions even if a term is not in σ-normal form. We would like to treat the τ-rules in the same way, perhaps performing τ-normalization only before trail inspection; however, we can see that changing the order of τ-rules destroys confluence even when β-redexes are triggered in the same order. Consider for example the reductions in Figure 1: performing a τ-step before the beta-reduction, as in the right branch, yields the expected result. If instead we delay the τ-step, the trail q decorating N is duplicated by beta reduction; furthermore, the order of q and $\boldsymbol{\beta}$ gets mixed up: even though q records computation that happened (once) *before* $\boldsymbol{\beta}$, the final trail asserts that q happened (twice) *after* $\boldsymbol{\beta}$.[1] As expected, the two trails (and consequently the terms they decorate) are not joinable.

[1] Although the right branch describes an unfaithful account of history, it is still a coherent one: we will explain this in more detail in the conclusions.

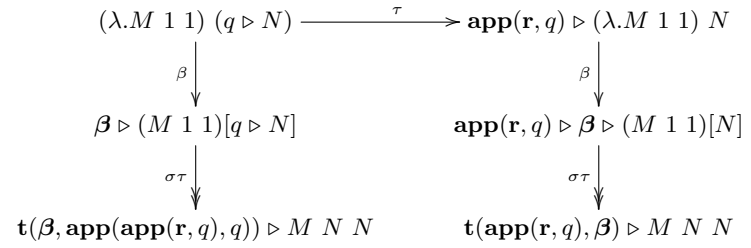

Fig. 1. Non-joinable reduction in $\mathbf{CAU}^-$ with naïve explicit substitutions

The example shows that β-reduction on terms whose trails have not been normalized is *anachronistic*. If we separated the trails stored in a term from the underlying, trail-less term, we might be able to define a *catachronistic*, or time-honoring version of β-reduction. For instance, if we write $\lfloor M \rfloor$ for trail-erasure and $\lceil M \rceil$ for the trail-extraction of a term M, catachronistic beta reduction could be written as follows:

$$
\begin{aligned}
(\lambda.M)\ N &\xrightarrow{\beta} \mathbf{t}(\lceil (\lambda.M)\ N \rceil, \boldsymbol{\beta}) \triangleright \lfloor M \rfloor\, [\lfloor N \rfloor] \\
\mathrm{let}_!(!_q M, N) &\xrightarrow{\beta} \mathbf{t}(\lceil \mathrm{let}_!(!_q M, N) \rceil, \boldsymbol{\beta_!}) \triangleright \lfloor N \rfloor\, [q \triangleright M] \\
!_q \mathcal{F}[\iota(\vartheta)] &\xrightarrow{\beta} !_q \mathcal{F}[\mathbf{ti} \triangleright q'\vartheta] \qquad \text{where } q' = \tau(\mathbf{t}(q, \lceil \mathcal{F}[\iota(\vartheta)] \rceil))
\end{aligned}
$$

We could easily define trail erasure and extraction as operations on pure $\mathbf{CAU}^-$ terms (without explicit substitutions), but the cost of eagerly computing their result would be proportional to the size of the input term; furthermore, the extension to explicit substitutions would not be straightforward. Instead, in the next section, we will describe an extended language to manipulate trail projections explicitly.

4 The calculus $\mathbf{CAU}^-_{\boldsymbol{\sigma}}$

We define the untyped Calculus of Audited Units with explicit substitutions, or $\mathbf{CAU}^-_\sigma$, as the following extension of the syntax of $\mathbf{CAU}^-$ presented in Section 2:

Terms	M, N	$::= 1 \mid \lambda.M \mid M\ N \mid \mathrm{let}_!(M, N) \mid !_q M \mid q \triangleright M \mid \iota(\vartheta)$
		$\mid\ M[s] \mid \lfloor M \rfloor$
Trails	q, q'	$::= \mathbf{r} \mid \mathbf{t}(q, q') \mid \boldsymbol{\beta} \mid \boldsymbol{\beta_!} \mid \mathbf{ti} \mid \mathbf{lam}(q) \mid \mathbf{app}(q, q')$
		$\mid\ \mathbf{let}_!(q, q') \mid \mathbf{tb}(\zeta) \mid \lceil M \rceil$
Substitutions	s, t	$::= \langle\rangle \mid \uparrow \mid M \cdot s \mid s \circ t$

$\mathbf{CAU}^-_\sigma$ builds on the observations about explicit substitutions we made in the previous section: in addition to closures $M[s]$, it provides syntactic trail erasures denoted by $\lfloor M \rfloor$; dually, the syntax of trails is extended with the explicit trail-extraction of a term, written $\lceil M \rceil$.

In the naïve presentation, we gave a satisfactory set of σ-rules defining the semantics of explicit substitutions, which we keep as part of $\mathbf{CAU}^-_\sigma$. In addition,

we need to provide new rules to define the semantics of explicit projections: we can adapt the informal definition of $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$, keeping in mind that the result of explicit projections, just like that of explicit substitutions, does not need to be computed immediately:

$$\begin{array}{rcl|rcl}
\lfloor 1\rfloor & \xrightarrow{\sigma} & 1 & \lceil 1\rceil & \xrightarrow{\sigma} & \mathbf{r} \\
\lfloor 1[\uparrow^n]\rfloor & \xrightarrow{\sigma} & 1[\uparrow^n] & \lceil 1[\uparrow^n]\rceil & \xrightarrow{\sigma} & \mathbf{rlam}(\lceil M\rceil) \\
\lfloor \lambda.M\rfloor & \xrightarrow{\sigma} & \lambda.\lfloor M\rfloor & \lceil \lambda.M\rceil & \xrightarrow{\sigma} & \\
\lfloor M\ N\rfloor & \xrightarrow{\sigma} & \lfloor M\rfloor\ \lfloor N\rfloor & \lceil M\ N\rceil & \xrightarrow{\sigma} & \mathbf{app}(\lceil M\rceil, \lceil N\rceil) \\
\lfloor !_q M\rfloor & \xrightarrow{\sigma} & !_q M & \lceil !_q M\rceil & \xrightarrow{\sigma} & \mathbf{r} \\
\lfloor \mathrm{let}_!(M,N)\rfloor & \xrightarrow{\sigma} & \mathrm{let}_!(\lfloor M\rfloor, \lfloor N\rfloor) & \lceil \mathrm{let}_!(M,N)\rceil & \xrightarrow{\sigma} & \mathbf{let}_!(\lceil M\rceil, \lceil N\rceil) \\
\lfloor q \triangleright M\rfloor & \xrightarrow{\sigma} & \lfloor M\rfloor & \lceil q \triangleright M\rceil & \xrightarrow{\sigma} & \mathbf{t}(q, \lceil M\rceil) \\
\left\lfloor \iota(\{\overrightarrow{M}\})\right\rfloor & \xrightarrow{\sigma} & \iota(\{\overrightarrow{\lfloor M\rfloor}\}) & \left\lceil \iota(\{\overrightarrow{M}\})\right\rceil & \xrightarrow{\sigma} & \mathbf{tb}(\{\overrightarrow{\lceil M\rceil}\})
\end{array}$$

These rules are completed by congruence rules asserting that they can be used in any subterm or subtrail of a given term or trail. The τ rules from Section 2 are added to $\mathbf{CAU}^-_\sigma$ with the obvious adaptations. We prove that σ and τ, together, yield a terminating and confluent rewriting system.

Theorem 1. $(\xrightarrow{\sigma} \cup \xrightarrow{\tau})$ *is terminating and confluent.*

Proof. Tools like AProVE [16] are able to prove termination automatically. Local confluence can be proved easily by considering all possible pairs of rules: full confluence follows as a corollary of these two results. □

We can prove that σ-normal terms coincide with the nameless variant of $\mathbf{CAU}^-$ outlined in Section 3. This is expressed by the following lemma:

Lemma 2. *The σ-normal terms, trails and substitutions of $\boldsymbol{CAU}^-_\sigma$ are expressed by the following grammar:*

$$\begin{array}{rcl}
M, N & ::= & 1 \mid 1[\uparrow^n] \mid \lambda.M \mid M\ N \mid \mathrm{let}_!(M,N) \mid !_q M \mid q \triangleright M \mid \iota(\vartheta) \\
q, q' & ::= & \mathbf{r} \mid \mathbf{t}(q,q') \mid \boldsymbol{\beta} \mid \boldsymbol{\beta_!} \mid \mathbf{ti} \mid \mathbf{lam}(q) \mid \mathbf{app}(q,q') \mid \mathbf{let}_!(q,q') \mid \mathbf{tb}(\zeta) \\
s, t & ::= & \langle\rangle \mid \uparrow^n \mid M \cdot s
\end{array}$$

From this result, we extract a definition of σ-normal contexts $\mathcal{E}$ and $\mathcal{S}$:

$$\begin{array}{rcl}
\mathcal{E} & ::= & \blacksquare \mid \lambda.\mathcal{E} \mid (\mathcal{E}\ N) \mid (M\ \mathcal{E}) \mid \mathrm{let}_!(\mathcal{E}, N) \mid \mathrm{let}_!(M, \mathcal{E}) \mid !_q \mathcal{E} \mid q \triangleright \mathcal{E} \mid \iota(\{\overrightarrow{M}, \mathcal{E}, \overrightarrow{N}\}) \\
\mathcal{S} & ::= & \mathcal{E} \cdot s \mid M \cdot \mathcal{S}
\end{array}$$

where all the terms M, N, trails q and substitutions s appearing in these definitions are σ-normal.

4.1 Beta reduction

We replace the definition of β-reduction by the following lazy rules that use trail-extraction and trail-erasure to ensure that the correct trails are eventually

produced:

$$\begin{aligned}
(\lambda.M)\ N &\xrightarrow{\text{Beta}} \mathbf{t}(\mathbf{app}(\mathbf{lam}(\lceil M\rceil),\lceil N\rceil),\boldsymbol{\beta}) \triangleright \lfloor M\rfloor\,[\lfloor N\rfloor] \\
\mathrm{let}_!(!_q M, N) &\xrightarrow{\text{Beta}} \mathbf{t}(\mathbf{let}_!(\mathbf{r},\lceil N\rceil),\boldsymbol{\beta}_!) \triangleright \lfloor N\rfloor\,[q \triangleright M] \\
!_q\mathcal{F}[\iota(\vartheta)] &\xrightarrow{\text{Beta}} !_q\mathcal{F}[\mathbf{ti} \triangleright q'\vartheta] \qquad \text{where } q' = \tau(\mathbf{t}(q,\lceil\mathcal{F}[\iota(\vartheta)]\rceil))
\end{aligned}$$

where $\mathcal{F}$ specifies that the reduction cannot take place within a bang, a substitution, or a trail erasure:

$$\mathcal{F} ::= \blacksquare \mid \lambda.\mathcal{F} \mid (\mathcal{F}\ N) \mid (M\ \mathcal{F}) \mid \mathrm{let}_!(\mathcal{F}, N) \mid \mathrm{let}_!(M, \mathcal{F}) \mid q \triangleright \mathcal{F} \mid \iota(\overrightarrow{M}, \mathcal{F}, \overrightarrow{N}) \mid \mathcal{F}[s]$$

As usual, the relation is extended to inner subterms by means of congruence rules. However, we need to be careful: we cannot reduce within a trail-erasure, because if we did, the newly created trail would be erroneously erased:

$$\begin{aligned}
\text{wrong: } \lfloor(\lambda.M)\ N\rfloor &\xrightarrow{\text{Beta}} \lfloor\mathbf{t}(\mathbf{app}(\mathbf{lam}(\lceil M\rceil),\lceil N\rceil),\boldsymbol{\beta}) \triangleright \lfloor M\rfloor\,[\lfloor N\rfloor]\rfloor \\
&\xrightarrow{\sigma} \lfloor\lfloor M\rfloor\,[\lfloor N\rfloor]\rfloor \\
\text{correct: } \lfloor(\lambda.M)\ N\rfloor &\xrightarrow{\sigma}\!\!\!\!\rightarrow (\lambda.\lfloor M\rfloor)\ \lfloor N\rfloor \\
&\xrightarrow{\text{Beta}} \mathbf{t}(\mathbf{app}(\mathbf{lam}(\lceil\lfloor M\rfloor\rceil),\lceil\lfloor N\rfloor\rceil),\boldsymbol{\beta}) \triangleright \lfloor M\rfloor\,[\lfloor N\rfloor]
\end{aligned}$$

This is why we express the congruence rule by means of contexts $\mathcal{E}_\sigma$ such that holes cannot appear within erasures (the definition also employs substitution contexts $\mathcal{S}_\sigma$ to allow reduction within substitutions):

$$\frac{M \xrightarrow{\text{Beta}} N}{\mathcal{E}_\sigma[M] \xrightarrow{\text{Beta}} \mathcal{E}_\sigma[N]}$$

$$\begin{aligned}
\mathcal{E}_\sigma ::= {} & \blacksquare \mid \lambda.\mathcal{E}_\sigma \mid (\mathcal{E}_\sigma\ N) \mid (M\ \mathcal{E}_\sigma) \mid \mathrm{let}_!(\mathcal{E}_\sigma, N) \\
& \mid \mathrm{let}_!(M, \mathcal{E}_\sigma) \mid !_q\mathcal{E}_\sigma \mid q \triangleright \mathcal{E}_\sigma \mid \iota(\{\overrightarrow{M}, \mathcal{E}_\sigma, \overrightarrow{N}\}) \\
& \mid \mathcal{E}_\sigma[s] \mid M[\mathcal{S}_\sigma] \\
\mathcal{S}_\sigma ::= {} & \mathcal{S}_\sigma \circ t \mid s \circ \mathcal{S}_\sigma \mid \mathcal{E}_\sigma \cdot s \mid M \cdot \mathcal{S}_\sigma
\end{aligned}$$

We denote $\sigma\tau$-equivalence (the reflexive, symmetric, and transitive closure of $\xrightarrow{\sigma\tau}$) by means of $\xleftrightarrow{\sigma\tau}$. As we will prove, $\sigma\tau$-equivalent $\mathbf{CAU}^-_\sigma$ terms can be interpreted as the same $\mathbf{CAU}^-$ term: for this reason, we define reduction in $\mathbf{CAU}^-_\sigma$ as the union of $\xrightarrow{\text{Beta}}$ and $\xleftrightarrow{\sigma\tau}$:

$$\xrightarrow{\mathbf{CAU}^-_\sigma} := \xrightarrow{\text{Beta}} \cup \xleftrightarrow{\sigma\tau}$$

4.2 Properties of the rewriting system

The main results we prove concern the relationship between $\mathbf{CAU}^-$ and $\mathbf{CAU}^-_\sigma$: firstly, every $\mathbf{CAU}^-$ reduction must still be a legal reduction within $\mathbf{CAU}^-_\sigma$; in addition, it should be possible to interpret every $\mathbf{CAU}^-_\sigma$ reduction as a $\mathbf{CAU}^-$ reduction over suitable $\sigma\tau$-normal terms.

Theorem 2. *If* $M \xrightarrow{\boldsymbol{CAU}^-}\!\!\!\!\rightarrow N$, *then* $M \xrightarrow{\boldsymbol{CAU}^-_\sigma}\!\!\!\!\rightarrow N$.

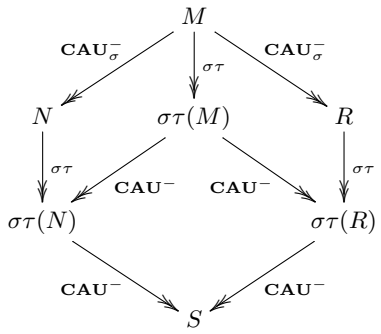

Fig. 2. Relativized confluence for $\mathbf{CAU}^-_\sigma$.

Theorem 3. *If* $M \xrightarrow{\boldsymbol{CAU}^-_\sigma}\!\!\!\!\!\rightarrow N$, *then* $\sigma\tau(M) \xrightarrow{\boldsymbol{CAU}^-}\!\!\!\!\!\rightarrow \sigma\tau(N)$.

Although $\mathbf{CAU}^-_\sigma$, just like $\mathbf{CAU}^-$, is *not* confluent (different reduction strategies produce different trails, and trail inspection can be used to compute on them, yielding different terms as well), the previous results allow us to prove a *relativized* confluence theorem:

Theorem 4. *If* $M \xrightarrow{\boldsymbol{CAU}^-_\sigma}\!\!\!\!\!\rightarrow N$ *and* $M \xrightarrow{\boldsymbol{CAU}^-_\sigma}\!\!\!\!\!\rightarrow R$, *and furthermore* $\sigma\tau(N)$ *and* $\sigma\tau(R)$ *are joinable in* $\boldsymbol{CAU}^-$, *then* N *and* R *are joinable in* $\boldsymbol{CAU}^-_\sigma$.

Proof. By Theorem 3 we know $\sigma\tau(M) \xrightarrow{\mathbf{CAU}^-}\!\!\!\!\!\rightarrow \sigma\tau(N)$ and $\sigma\tau(M) \xrightarrow{\mathbf{CAU}^-}\!\!\!\!\!\rightarrow \sigma\tau(R)$. By hypothesis, there exists a $\mathbf{CAU}^-$ term S such that $\sigma\tau(N) \xrightarrow{\mathbf{CAU}^-}\!\!\!\!\!\rightarrow S$ and $\sigma\tau(R) \xrightarrow{\mathbf{CAU}^-}\!\!\!\!\!\rightarrow S$. By Theorem 2 these two last reductions are also allowed in $\mathbf{CAU}^-_\sigma$. Figure 2 illustrates the proof. □

While the proof of Theorem 2 is not overly different from the similar proof for the $\lambda\sigma$-calculus, Theorem 3 is more interesting. The main challenge is to prove that whenever $M \xrightarrow{\text{Beta}} N$, then $\sigma\tau(M) \xrightarrow{\mathbf{CAU}^-}\!\!\!\!\!\rightarrow \sigma\tau(N)$. However, when proceeding by induction on $M \xrightarrow{\text{Beta}} N$, the terms $\sigma\tau(M)$ and $\sigma\tau(N)$ are too normalized to provide us with a good enough induction hypothesis: in particular, we would want them to be in the form $q \triangleright R$ even when q is reflexivity. We call terms in this quasi-normal form *focused*, and prove the theorem by reasoning on them. The appendix contains the details of the proof.

5 Evaluation

To evaluate how $\mathbf{CAU}^-_\sigma$ can be beneficial in achieving more efficient audited computation, let us consider the following term:

$$! \ (\lambda^{k+2}.\mathit{minus}\ 1\ (k+2))\ \iota(\vartheta_+)\ M_1 \cdots M_k\ \iota(\vartheta_+)$$

The first term in the application receives $k+2$ arguments as input, then subtracts the first one from the last (under the assuption that these two arguments represent integer numbers), ignoring the k intermediate arguments. We apply this function to a list of arguments, the first and last of which are inspections using the ϑ_+ defined in Example 2; the terms $M_1 \cdots M_k$ can be chosen freely.

The purpose of the term is to provide a rudimentary but effective *profiling* facility: under a call-by-value strategy, the term will compute the cost of evaluating the $\overrightarrow{M_k}$: under a call-by-value strategy, we start by reducing the first inspection, which will count the number of contractions performed before the start of the profiling process (if any); then we reduce the M_i, from left to right; finally, we count the contraction steps at the end of the process, by means of another trail inspection: by subtracting the two values, we obtain an approximation of the cost of evaluating the M_i.

We now examine this reduction in detail: for this purpose, we fix the M_i to be identity terms $I = \lambda.1$: this expression is already in normal form, therefore such a computation will only count the "overhead" contractions of the profiling process. In $\mathbf{CAU}^-$, the reduction proceeds by means of β-contractions followed by τ-normalization steps:

$$\begin{array}{l} !\ (\lambda^{k+2}.\mathit{minus}\ 1\ (k+2))\ \iota(\vartheta_+)\ \overrightarrow{I^k}\ \iota(\vartheta_+) \\ \xrightarrow{\beta} \quad !\ (\lambda^{k+2}.\mathit{minus}\ 1\ (k+2))\ (\mathbf{ti} \triangleright \bar{0})\ \overrightarrow{I^k}\ \iota(\vartheta_+) \\ \xrightarrow{\tau}{}^{(c_1)} !_{\mathbf{app}(\cdots\mathbf{app}(\mathbf{ti}),\mathbf{r})\cdots,\mathbf{r})}\ (\lambda^{k+2}.\mathit{minus}\ 1\ (k+2))\ \bar{0}\ \overrightarrow{I^k}\ \iota(\vartheta_+) \end{array}$$

We assume that no computation has been performed on the initial term, so the first trail inspection returns $\vartheta(\mathbf{r}) = \bar{0}$; the trail normalization step includes $k+2$ reductions to push **ti** to the outer bang; the usage of congruence trails **app** makes the trail grow to a size which is proportional to the number $k+2$ of arguments of the application; to complete the normalization, we need to examine this trail: the time required by this process is proportional to the size of the trail. Thus, the cost c_1 of this normalization grows linearly with the size of the application.

$$\begin{array}{l} \xrightarrow{\beta} \quad !_{\mathbf{app}(\cdots\mathbf{app}(\mathbf{ti}),\mathbf{r})\cdots,\mathbf{r})}\ (\boldsymbol{\beta} \triangleright \lambda^{k+1}.\mathit{minus}\ 1\ \bar{0})\ \overrightarrow{I^k}\ \iota(\vartheta_+) \\ \xrightarrow{\tau}{}^{(c_2)} !_{\mathbf{app}(\cdots\mathbf{t}(\mathbf{app}(\mathbf{ti}),\mathbf{r}),\boldsymbol{\beta})\cdots,\mathbf{r})}\ (\lambda^{k+1}.\mathit{minus}\ 1\ \bar{0})\ \overrightarrow{I^k}\ \iota(\vartheta_+) \end{array}$$

The next step is to reduce the first argument of the application. The cost c_2 of the subsequent trail normalization can be obtained similarly to the previous case.

$$\begin{array}{l} \xrightarrow{\beta} \quad !_{\mathbf{app}(\cdots\mathbf{app}(\mathbf{ti}),\mathbf{r})\cdots,\mathbf{r})}\ (\boldsymbol{\beta} \triangleright \lambda^{k}.\mathit{minus}\ 1\ \bar{0})\ \overrightarrow{I^{k-1}}\ \iota(\vartheta_+) \\ \xrightarrow{\tau}{}^{(c_e)} !_{\mathbf{app}(\cdots\mathbf{t}(\mathbf{app}(\mathbf{t}(\mathbf{app}(\mathbf{ti}),\mathbf{r}),\boldsymbol{\beta}),\mathbf{r}),\boldsymbol{\beta})\cdots,\mathbf{r})}\ (\lambda^{k}.\mathit{minus}\ 1\ \bar{0})\ \overrightarrow{I^{k-1}}\ \iota(\vartheta_+) \\ \cdots \\ \xrightarrow{\tau}{}^{c_{k+2}} !_{q_k}\ (\lambda.\mathit{minus}\ 1\ \bar{0})\ \iota(\vartheta_+) \\ \xrightarrow{\beta}{}^{c_\beta} !_{q_k}\ (\lambda.\mathit{minus}\ 1\ \bar{0})\ (\mathbf{ti} \triangleright q_k\vartheta_+) \end{array}$$

We reduce the application one argument after another: notice that, even though the size of the term decreases, the trail can only grow, although its size is always bounded by a linear function of k: thus, each trail normalization requires $c_i \in \Theta(k)$ steps. The cost of the final β-step is also $\Theta(k)$, because of the time needed to construct the term $q_k\vartheta_+$. The total cost of this reduction is thus $\left(c^\beta + \sum_{i=1}^{k+2} c_i\right) \in \Theta(k^2)$

We now consider the same term in $\mathbf{CAU}^-_\sigma$. The following result will help us compute with explicit operations in an efficient way.

Lemma 3. *The following rewritings are admissible in* $\boldsymbol{CAU}^-_\sigma$*:*

$(a.1)$ $M \longrightarrow \lceil M \rceil \triangleright \lfloor M \rfloor$
$(a.2)$ $\lceil \lfloor M \rfloor \rceil \longrightarrow \mathbf{r}$
$(a.3)$ $\lfloor \lambda.M \rfloor \ (q_N \triangleright \lfloor N \rfloor) \longrightarrow \mathbf{t}(\mathbf{app}(\mathbf{r}, q_N), \boldsymbol{\beta}) \triangleright \lfloor M[\lfloor N \rfloor]\rfloor$
$(a.4)$ $(q_M \triangleright \lfloor (\lambda.M)[s] \rfloor) \ (q_N \triangleright \lfloor N \rfloor) \longrightarrow \mathbf{t}(\mathbf{app}(q_M, q_N), \boldsymbol{\beta}) \triangleright \lfloor M[\lfloor N \rfloor \cdot s]\rfloor$

In $\mathbf{CAU}^-_\sigma$, we expect to evaluate terms $!\ N$ with no previous computation history: thus, we may start the reduction from $!\ \lfloor N \rfloor$ instead (this is not a limitation, since in the general case we may use rule $(a.1)$ to rewrite N to $\lceil N \rceil \triangleright \lfloor N \rfloor$, and then evaluate the erased subterm as normal). In our example, we proceed as follows:

$$
\begin{aligned}
& !\ \left\lfloor (\lambda^{k+2}.\mathit{minus}\ 1\ (k+2))\ \iota(\vartheta_+)\ \overrightarrow{I^k}\ \iota(\vartheta_+) \right\rfloor \\
\xrightarrow{\sigma\twoheadrightarrow} \quad & !\ \left\lfloor \lambda^{k+2}.\mathit{minus}\ 1\ (k+2) \right\rfloor\ \iota(\lfloor \vartheta_+ \rfloor)\ \overrightarrow{\lfloor I \rfloor^k}\ \lfloor \iota(\vartheta_+) \rfloor \\
\xrightarrow{\text{Beta}^{(\star)}} \quad & !\ \left\lfloor \lambda^{k+2}.\mathit{minus}\ 1\ (k+2) \right\rfloor\ (\mathbf{ti} \triangleright \lfloor \bar{0} \rfloor)\ \overrightarrow{\lfloor I \rfloor^k}\ \lfloor \iota(\vartheta_+) \rfloor
\end{aligned}
$$

The first σ reductions, which push trail erasure inside, require $\Theta(k)$ steps[2]. The Beta-reduction flagged with $(\star)$ requires us to scan the whole body of the bang to compute a normalized trail: although we can optimize it by not looking for trails within erasures (rule $(a.2)$), the body of the bang contains $k+2$ erasures, so the cost of this step is $\Theta(k)$.

$$
\begin{aligned}
& \xrightarrow{(a.3)} !\ (\mathbf{t}(\mathbf{app}(\mathbf{r}, \mathbf{ti}), \boldsymbol{\beta}) \triangleright \left\lfloor (\lambda^{k+1}.\mathit{minus}\ 1\ (k+2))[\lfloor \bar{0} \rfloor] \right\rfloor)\ \overrightarrow{\lfloor I \rfloor^k}\ \lfloor \iota(\vartheta_+) \rfloor \\
& \xrightarrow{(a.4)} !\ (\mathbf{t}(\mathbf{app}(\mathbf{t}(\mathbf{app}(\mathbf{r}, \mathbf{ti}), \boldsymbol{\beta}), \mathbf{r}), \boldsymbol{\beta}) \triangleright \\
& \qquad \left\lfloor (\lambda^{k}.\mathit{minus}\ 1\ (k+2))[\lfloor I \rfloor \cdot \lfloor \bar{0} \rfloor] \right\rfloor)\ \overrightarrow{\lfloor I \rfloor^{k-1}}\ \lfloor \iota(\vartheta_+) \rfloor
\end{aligned}
$$

In the following steps, we can enjoy the benefits of our calculus: not only we do not need to normalize trails after each contraction step, but certain frequent term forms that are *almost* redexes, can be optimized by means the admissible rules $(a.3)$ and $(a.4)$ (the former being a special case of the latter); each of these

[2] Actually, in an abstract machine implementation, these σ-steps would be executed as part of the redex search, so their contribution to the computational cost is negligible.

reductions only requires constant time.

$$
\begin{array}{l}
\cdots \\
\xrightarrow{(a_4)} \quad ! \ (q'_k \triangleright \left\lfloor (\lambda.\mathit{minus}\ 1\ (k+2))[\overrightarrow{\lfloor I \rfloor}^{k} \cdot \lfloor \bar{0} \rfloor] \right\rfloor) \ \iota(\lfloor \vartheta_+ \rfloor) \\
\xrightarrow{\text{Beta}}{}^{(\dagger)} \quad ! \ (q'_k \triangleright \left\lfloor (\lambda.\mathit{minus}\ 1\ (k+2))[\lfloor I \rfloor^{k} \cdot \lfloor \bar{0} \rfloor] \right\rfloor) \ (\mathbf{ti} \triangleright q_k \vartheta_+)
\end{array}
$$

The Beta-reduction flagged with (†) requires $\Theta(k)$ steps, because we need to obtain the normal form q_k of q'_k, and the size of this last trail grows linearly with k. The total cost of the reduction, given by three linear time and k constant time operations, is thus in $\Theta(k)$.

6 Conclusions and Future Directions

The calculus $\mathbf{CAU}^{-}_{\sigma}$ which we introduced in this paper provides a finer-grained view over the reduction of history-carrying terms: although it does not enforce a reduction strategy, but provides better grounds on which to implement reduction efficiently. In future work, we plan to adapt abstract machines such as the SECD machine [17] or the CAM machine [14] to the calculus of audited units, using $\mathbf{CAU}^{-}_{\sigma}$ as their formal justification.

In our discussion, we showed that the original definition of beta-reduction, when applied to terms that are not in trail-normal form, creates temporally unsound trails. We might wonder whether these anachronistic trails carry any meaning: let us take, as an example, the reduction on the left branch of Figure 1:

$$(\lambda.M\ 1\ 1)(q \triangleright N) \longrightarrow\!\!\!\!\!\rightarrow \mathbf{t}(\boldsymbol{\beta}, \mathbf{app}(\mathbf{app}(\mathbf{r}, q), q)) \triangleright M\ N\ N$$

We know that q is the trace left behind by the reduction that led to N from the original term, say R:

$$R \longrightarrow q \triangleright N$$

We can see that the anachronistic trail is actually consistent with the reduction of $(\lambda.M\ 1\ 1)\ R$ under a leftmost-outermost strategy:

$$
\begin{array}{l}
(\lambda.M\ 1\ 1)\ R \longrightarrow \boldsymbol{\beta} \triangleright M\ R\ R \longrightarrow\!\!\!\!\!\rightarrow \boldsymbol{\beta} \triangleright M\ (q \triangleright N)\ (q \triangleright N) \\
\qquad \longrightarrow\!\!\!\!\!\rightarrow \mathbf{t}(\boldsymbol{\beta}, \mathbf{app}(\mathbf{app}(\mathbf{r}, q), q)) \triangleright M\ N\ N
\end{array}
$$

Under the anachronistic reduction, q acts as the ghost of an original inner redex. Through substitution within M, we get evidence that the contraction of an inner redex can be swapped with a subsequent head reduction: this is a key result in the proof of standardization that is usually obtained using the notion of *residual* ([11], Lemma 11.4.5). Based on this remark, we conjecture that trails might be used to provide a more insightful proof: it would thus be interesting to see how trails relate to recent advancements in standardization ([3,9,27,19]).

References

1. Abadi, M., Cardelli, L., Curien, P.L., Lévy, J.J.: Explicit substitutions. Journal of Functional Programming 1(4), 375–416 (Oct 1991), https://www.cambridge.org/core/article/explicit-substitutions/C1B1AFAE8F34C953C1B2DF3C2D4C2125
2. Abadi, M., Fournet, C.: Access control based on execution history. In: NDSS (2003)
3. Accattoli, B., Bonelli, E., Kesner, D., Lombardi, C.: A nonstandard standardization theorem. In: Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 659–670. POPL '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2535838.2535886
4. Amir-Mohammadian, S., Chong, S., Skalka, C.: Correct audit logging: Theory and practice. In: POST. pp. 139–162 (2016)
5. Artemov, S.: The logic of justification. Review of Symbolic Logic 1(4), 477–513 (2008)
6. Artëmov, S.N.: Explicit provability and constructive semantics. Bulletin of Symbolic Logic 7(1), 1–36 (2001)
7. Artëmov, S.N., Bonelli, E.: The intensional lambda calculus. In: Logical Foundations of Computer Science. pp. 12–25 (2007)
8. Artemov, S.: Justification logic. In: Hölldobler, S., Lutz, C., Wansing, H. (eds.) Logics in Artificial Intelligence: 11th European Conference, JELIA 2008, Dresden, Germany, September 28-October 1, 2008. Proceedings. pp. 1–4. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-87803-2_1
9. Asperti, A., Levy, J.J.: The cost of usage in the λ-calculus. In: Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science. pp. 293–300. LICS '13, IEEE Computer Society, Washington, DC, USA (2013), http://dx.doi.org/10.1109/LICS.2013.35
10. Banerjee, A., Naumann, D.A.: History-based access control and secure information flow. In: Barthe, G., Burdy, L., Huisman, M., Lanet, J.L., Muntean, T. (eds.) Construction and Analysis of Safe, Secure, and Interoperable Smart Devices: International Workshop, CASSIS 2004, Marseille, France, March 10-14, 2004, Revised Selected Papers. pp. 27–48. Springer Berlin Heidelberg, Berlin, Heidelberg (2005), http://dx.doi.org/10.1007/978-3-540-30569-9_2
11. Barendregt, H.P.: The Lambda Calculus : its Syntax and Semantics. Studies in logic and the foundations of mathematics, North-Holland, Amsterdam, New-York, Oxford (1981), http://opac.inria.fr/record=b1078721
12. Bavera, F., Bonelli, E.: Justification logic and audited computation. Journal of Logic and Computation (2015), published online, June 19, 2015
13. de Bruijn, N.: Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. Indagationes Mathematicae 34(5), 381–392 (1972)
14. Cousineau, G., Curien, P.L., Mauny, M.: The categorical abstract machine. Science of Computer Programming 8(2), 173 – 202 (1987), http://www.sciencedirect.com/science/article/pii/0167642387900207
15. Garg, D., Jia, L., Datta, A.: Policy auditing over incomplete logs: theory, implementation and applications. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011. pp. 151–162 (2011), http://doi.acm.org/10.1145/2046707.2046726
16. Giesl, J., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Proving termination

of programs automatically with AProVE. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Automated Reasoning: 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings, pp. 184–191. Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-08587-6_13
17. Henderson, P.: Functional Programming: Application and Implementation. Prentice-Hall, Englewood Cliffs, NJ (1980)
18. Jia, L., Vaughan, J.A., Mazurak, K., Zhao, J., Zarko, L., Schorr, J., Zdancewic, S.: Aura: a programming language for authorization and audit. In: ICFP. pp. 27–38 (2008)
19. Kashima, R.: A proof of the standardization theorem in lambda-calculus. Tech. Rep. Research Reports on Mathematical and Computing Science, Tokyo Institute of Technology (2000)
20. Moreau, L.: The foundations for provenance on the web. Foundations and Trends in Web Science 2(2–3) (2010)
21. Perera, R., Acar, U.A., Cheney, J., Levy, P.B.: Functional programs that explain their work. In: ICFP. pp. 365–376. ACM (2012)
22. Pfenning, F., Davies, R.: A judgmental reconstruction of modal logic. Mathematical Structures in Computer Science 11(4), 511–540 (2001)
23. Ricciotti, W.: A core calculus for provenance inspection. In: Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming. pp. 187–198. PPDP '17, ACM, New York, NY, USA (2017), http://doi.acm.org/10.1145/3131851.3131871
24. Ricciotti, W., Cheney, J.: Strongly Normalizing Audited Computation. In: Goranko, V., Dam, M. (eds.) 26th EACSL Annual Conference on Computer Science Logic (CSL 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 82, pp. 36:1–36:21. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2017), http://drops.dagstuhl.de/opus/volltexte/2017/7681
25. Ricciotti, W., Stolarek, J., Perera, R., Cheney, J.: Imperative functional programs that explain their work. Proc. ACM Program. Lang. 1(ICFP), 14:1–14:28 (Aug 2017), http://doi.acm.org/10.1145/3110258
26. Vaughan, J.A., Jia, L., Mazurak, K., Zdancewic, S.: Evidence-based audit. In: CSF. pp. 177–191 (2008)
27. Xi, H.: Upper bounds for standardizations and an application. J. Symbolic Logic 64(1), 291–303 (03 1999), http://projecteuclid.org/euclid.jsl/1183745706

A Proofs

In this appendix, we provide more detail about the proofs mentioned in Section 4.2. We start by defining meta-level projections and focused forms, and then we prove some theorems about them.

Definition 2. *The meta-level projections* $\lfloor\!\lfloor M \rfloor\!\rfloor$ *and* $\lceil\!\lceil M \rceil\!\rceil$ *are defined as follows:*

$$\lfloor\!\lfloor M \rfloor\!\rfloor = \begin{cases} M' & \text{if } \sigma\tau(M) = q \triangleright M' \\ \sigma\tau(M) & \text{else} \end{cases} \qquad \lceil\!\lceil M \rceil\!\rceil = \begin{cases} q & \text{if } \sigma\tau(M) = q \triangleright M' \\ \mathbf{r} & \text{else} \end{cases}$$

Definition 3. *The focused form of a term* M*, denoted by* $\|M\|$*, is defined by:*

$$\|M\| = \lceil\!\lceil M \rceil\!\rceil \triangleright \lfloor\!\lfloor M \rfloor\!\rfloor$$

The focused form of a σ*-normal substitution* $s = \overrightarrow{N} \cdot \uparrow^p$ *is defined by:*

$$\|s\| = \overrightarrow{\|N\|} \cdot \uparrow^p$$

This definition is extended to all substitutions by taking $\|s\| = \|\sigma(s)\|$*.*

Lemma 4. *For all* M *and* s*, we have* $\sigma\tau(M) = \sigma\tau(\|M\|)$ *and* $\sigma\tau(s) = \sigma\tau(\|s\|)$*.*

Lemma 5. *For all* M*,* $\lfloor M \rfloor \xrightarrow{\sigma\tau}\!\!\!\!\rightarrow \lfloor\!\lfloor M \rfloor\!\rfloor$ *and* $\lceil M \rceil \xrightarrow{\sigma\tau}\!\!\!\!\rightarrow \lceil\!\lceil M \rceil\!\rceil$*.*

Proof. After unfolding the definitions, the proof is by induction on $\sigma\tau(M)$. □

We also define a meta-level operation corresponding to explicit substitutions, and prove the correspondence:

Definition 4. *For* σ*-normal* M *and* $\overrightarrow{N} = N_1, \ldots, N_k$*, the meta-level substitution* $M\left\{\overrightarrow{N}\right\}_p$ *is defined by recursion on* σ*-normal terms as follows:*

$$\begin{aligned}
m\left\{\overrightarrow{N}\right\}_p &= N_m & n\left\{\overrightarrow{N}\right\}_p &= n + p \\
(\lambda.M)\left\{\overrightarrow{N}\right\}_p &= \lambda.(M\left\{1, \overrightarrow{\mathrm{lift}(N)}\right\}_{p+1}) & (R\ S)\left\{\overrightarrow{N}\right\}_p &= R\left\{\overrightarrow{N}\right\}_p\ S\left\{\overrightarrow{N}\right\}_p \\
\mathrm{let}_!(R,S)\left\{\overrightarrow{N}\right\}_p &= \mathrm{let}_!\left(R\left\{\overrightarrow{N}\right\}_p,\right. & (!_q M)\left\{\overrightarrow{N}\right\}_p &= !_q(M\left\{\overrightarrow{N}\right\}_p) \\
&\qquad \left. S\left\{1, \overrightarrow{\mathrm{lift}(N)}\right\}_{p+1}\right) & \iota(\{\overrightarrow{M}\})\left\{\overrightarrow{N}\right\}_p &= \iota(\{\overrightarrow{M\left\{\overrightarrow{N}\right\}_p}\}) \\
(q \triangleright M)\left\{\overrightarrow{N}\right\}_p &= q \triangleright (M\left\{\overrightarrow{N}\right\}_p)
\end{aligned}$$

where $m \leq k < p$ *and* $\overrightarrow{\mathrm{lift}(N)}$ *lifts by one all of the free indices in each* N_i*. We will write* $M\left\{\overrightarrow{N}\right\}$ *as syntactic sugar for* $M\left\{\overrightarrow{N}\right\}_0$*.*

Lemma 6. *For all $M, \overrightarrow{N}$ in $\sigma\tau$-normal form, $M[\overrightarrow{N} \cdot \uparrow^p] \xrightarrow{\sigma\tau}\!\!\!\!\rightarrow M\left\{\overrightarrow{N}\right\}_p$. In particular, $M[R] \xrightarrow{\sigma\tau}\!\!\!\!\rightarrow M\{R\}$.*

Proof. Routine induction on σ-normal forms M. □

We can use meta-substitution to define β-reduction for σ-normal terms in the same style as in $\mathbf{CAU}^-$, and lift it to eager $\bar{\beta}$-reduction acting on focused terms:

$$(\lambda.M)\ N \xrightarrow{\beta} \boldsymbol{\beta} \triangleright M\{N\} \qquad \mathrm{let}_!(!_q M, N) \xrightarrow{\beta} \boldsymbol{\beta_!} \triangleright N\{q \triangleright M\}$$

$$!_q\mathcal{F}[\iota(\vartheta)] \xrightarrow{\beta} !_q\mathcal{F}[\mathbf{ti} \triangleright q\vartheta]$$

$$\frac{M \xrightarrow{\beta} N}{\mathcal{E}[M] \xrightarrow{\beta} \mathcal{E}[N]} \qquad \frac{M \xrightarrow{\beta} N}{\mathcal{S}[M] \xrightarrow{\beta} \mathcal{S}[N]}$$

$$\frac{M \xrightarrow{\beta} N}{\|M\| \xrightarrow{\bar{\beta}} \|N\|} \qquad \frac{s \xrightarrow{\beta} t}{\|s\| \xrightarrow{\bar{\beta}} \|t\|}$$

The last two rules, defining $\bar{\beta}$-reduction in terms of β-reduction, are not sufficiently compositional for our proofs, but we can prove a version where the premise is also a $\bar{\beta}$-reduction:

Lemma 7. *If $M \xrightarrow{\bar{\beta}} N$, then $\|\mathcal{E}[M]\| \xrightarrow{\bar{\beta}} \|\mathcal{E}[N]\|$ and $\|\mathcal{S}[M]\| \xrightarrow{\bar{\beta}} \|\mathcal{S}[N]\|$.*

Proof. From the hypothesis $M \xrightarrow{\bar{\beta}} N$ we obtain M', N' such that $M = \|M'\|$, $N = \|N'\|$ and $M' \xrightarrow{\beta} N'$. We proceed by induction on this reduction: in the congruence case we obtain a context $\mathcal{E}'$ that we need to merge with $\mathcal{E}$ to obtain the thesis. The substitution case follows similarly. □

Then we prove that β-reduction can be simulated in $\mathbf{CAU}^-_\sigma$:

Lemma 8. *If $M \xrightarrow{\beta} N$, then $M \xrightarrow{Beta} \xrightarrow{\sigma\tau}\!\!\!\!\rightarrow N$*

Proof. By induction on the hypothesis, using Lemma 6. □

Proof of Theorem 2. *If $M \xrightarrow{\boldsymbol{CAU}^-}\!\!\!\!\rightarrow N$, then $M \xrightarrow{\boldsymbol{CAU}^-_\sigma}\!\!\!\!\rightarrow N$.*
$M \xrightarrow{\mathbf{CAU}^-} N$ implies $M \xrightarrow{\beta} R \xrightarrow{\tau}\!\!\!\!\rightarrow \tau(R) = N$ for some R; then $M \xrightarrow{\mathbf{CAU}^-_\sigma}\!\!\!\!\rightarrow N$ is an immediate consequence of Lemma 8. □

We are also interested in proving the dual statement: for every $\mathbf{CAU}^-_\sigma$ Beta-reduction there should be a $\mathbf{CAU}^-$ reduction on the corresponding $\sigma\tau$-normal forms. To prove this theorem, we need a suitable version of the usual substitutivity property of β-reduction, which is proved in the standard way:

Lemma 9.
If $M \xrightarrow{\beta} M'$, *then* $M \left\{\overrightarrow{N}\right\}_k \xrightarrow{\beta} M' \left\{\overrightarrow{N}\right\}_k$.
If $N_i \xrightarrow{\beta} N_i'$, *then* $M \{N_1 \cdots N_{i-1}, N_i, N_{i+1} \cdots N_p\}_k \xrightarrow{\beta} \!\!\!\!\rightarrow$
$M \{N_1 \cdots N_{i-1}, N_i', N_{i+1} \cdots N_p\}_k$.

We can now prove that Beta-reductions can be mimicked by $\bar{\beta}$-reductions on the corresponding focused forms:

Lemma 10. *If* $M \xrightarrow{Beta} N$ *then* $\|M\| \xrightarrow{\bar{\beta}} \!\!\!\!\rightarrow \|N\|$.
If $s \xrightarrow{Beta} t$ *then* $\|s\| \xrightarrow{\bar{\beta}} \!\!\!\!\rightarrow \|t\|$.

Proof. By mutual induction on the derivations of $M \xrightarrow{\text{Beta}} N$ and $s \xrightarrow{\text{Beta}} t$. We consider two base cases, the applied explicit substitution case, and one of the other inductive cases (reduction in the first subterm of an application); the remaining cases can be proved similarly.

The first base case consists of lambda-application redexes:

$$(\lambda M)\ N \xrightarrow{\text{Beta}} \mathbf{t}(\mathbf{app}(\mathbf{lam}(\lceil M \rceil), \lceil N \rceil), \boldsymbol{\beta}) \triangleright \lfloor M \rfloor\, [\lfloor N \rfloor]$$

We need to prove that:

$$\|(\lambda M)\ N\| \xrightarrow{\bar{\beta}} \|\mathbf{t}(\mathbf{app}(\mathbf{lam}(\lceil M \rceil), \lceil N \rceil), \boldsymbol{\beta})) \triangleright \lfloor M \rfloor\, [\lfloor N \rfloor]\|$$

By confluence of $\sigma\tau$, we rewrite the left-hand side of the thesis with a $\sigma\tau$-equivalent focused term:

$$\|(\lambda M)\ N)\| = \|\mathbf{app}(\mathbf{lam}(\llceil M \rrceil), \llceil N \rrceil) \triangleright (\lambda \llfloor M \rrfloor)\ \llfloor N \rrfloor\|$$

Now we perform a $\xrightarrow{\bar{\beta}}$ step to obtain:

$$\begin{aligned} &\|\mathbf{app}(\mathbf{lam}(\llceil M \rrceil), \llceil N \rrceil) \triangleright \boldsymbol{\beta} \triangleright \llfloor M \rrfloor\, \{\llfloor N \rrfloor\}\| \\ =\ &\|\mathbf{t}(\mathbf{app}(\mathbf{lam}(\llceil M \rrceil), \llceil N \rrceil), \boldsymbol{\beta}) \triangleright \llfloor M \rrfloor\, \{\llfloor N \rrfloor\}\| \end{aligned}$$

We prove that the rhs of the thesis equals the result of the $\bar{\beta}$-reduction:

$$\begin{aligned} &\|\mathbf{t}(\mathbf{app}(\mathbf{lam}(\lceil M \rceil), \lceil N \rceil), \boldsymbol{\beta}) \triangleright \lfloor M \rfloor\, [\lfloor N \rfloor]\| \\ =\ &\|\mathbf{t}(\mathbf{app}(\mathbf{lam}(\llceil M \rrceil), \llceil N \rrceil), \boldsymbol{\beta}) \triangleright \llfloor M \rrfloor [\llfloor N \rrfloor]\| \\ =\ &\|\mathbf{t}(\mathbf{app}(\mathbf{lam}(\llceil M \rrceil), \llceil N \rrceil), \boldsymbol{\beta}) \triangleright \llfloor M \rrfloor\, \{\llfloor N \rrfloor\}\| \end{aligned}$$

The case of let-box redexes is similar. In $\mathbf{CAU}^{-}_{\sigma}$, we have:

$$\text{let}_!(!_q M, N) \xrightarrow{\text{Beta}} \mathbf{t}(\mathbf{let}_!(\mathbf{r}, \lceil N \rceil), \boldsymbol{\beta_!}) \triangleright \lfloor N \rfloor\, [q \triangleright M]$$

We need to prove that:

$$\|\text{let}_!(!_q M, N)\| \xrightarrow{\beta} \|\mathbf{t}(\mathbf{let}_!(\mathbf{r}, \lceil N \rceil), \boldsymbol{\beta_!}) \triangleright \lfloor N \rfloor\, [q \triangleright M]\|$$

We rewrite the lhs of the thesis:

$$\|\mathrm{let}_!(!_q M, N)\| = \|\mathbf{let}_!(\mathbf{r}, \lceil\!\lceil N \rceil\!\rceil) \triangleright \mathrm{let}_!(!_q M, \lfloor\!\lfloor N \rfloor\!\rfloor)\|$$

then perform a $\bar{\beta}$-step to obtain:

$$\begin{aligned}
&\|\mathbf{let}_!(\mathbf{r}, \lceil\!\lceil N \rceil\!\rceil) \triangleright \boldsymbol{\beta_!} \triangleright \lfloor\!\lfloor N \rfloor\!\rfloor \{q \triangleright M\}\| \\
=&\|\mathbf{t}(\mathbf{let}_!(\mathbf{r}, \lceil\!\lceil N \rceil\!\rceil), \boldsymbol{\beta_!}) \triangleright \lfloor\!\lfloor N \rfloor\!\rfloor \{q \triangleright M\}\| \\
=&\|\mathbf{t}(\mathbf{t}(\mathbf{let}_!(\mathbf{r}, \lceil\!\lceil N \rceil\!\rceil), \boldsymbol{\beta_!}), \lceil\!\lceil \lfloor\!\lfloor N \rfloor\!\rfloor \{q \triangleright M\} \rceil\!\rceil) \triangleright \lfloor\!\lfloor \lfloor\!\lfloor N \rfloor\!\rfloor \{q \triangleright M\} \rfloor\!\rfloor\|
\end{aligned}$$

We prove that the rhs of the thesis equals the result of the $\bar{\beta}$-reduction:

$$\begin{aligned}
&\|\mathbf{t}(\mathbf{let}_!(\mathbf{r}, \lceil N \rceil), \boldsymbol{\beta_!}) \triangleright \lfloor N \rfloor [q \triangleright M]\| \\
=&\|\mathbf{t}(\mathbf{let}_!(\mathbf{r}, \lceil\!\lceil N \rceil\!\rceil), \boldsymbol{\beta_!}) \triangleright \lfloor\!\lfloor N \rfloor\!\rfloor [q \triangleright M])\| \\
=&\|\mathbf{t}(\mathbf{let}_!(\mathbf{r}, \lceil\!\lceil N \rceil\!\rceil), \boldsymbol{\beta_!}) \triangleright \lfloor\!\lfloor N \rfloor\!\rfloor \{q \triangleright M\}\| \\
=&\|\mathbf{t}(\mathbf{t}(\mathbf{let}_!(\mathbf{r}, \lceil\!\lceil N \rceil\!\rceil), \boldsymbol{\beta_!}), \lceil\!\lceil \lfloor\!\lfloor N \rfloor\!\rfloor \{q \triangleright M\} \rceil\!\rceil) \triangleright \lfloor\!\lfloor \lfloor\!\lfloor N \rfloor\!\rfloor \{q \triangleright M\} \rfloor\!\rfloor\|
\end{aligned}$$

In the application case, we need to prove $\|M\ N\| \xrightarrow{\bar{\beta}} \|M'\ N\|$ under the induction hypothesis that $\|M\| \xrightarrow{\bar{\beta}} \|M'\|$. By Lemma 7 we prove

$$\|\|M\|\ \ N\| \xrightarrow{\bar{\beta}} \|\|M'\|\ \ N\|$$

which equals the thesis by substitution for $\sigma\tau$-equivalent subterms.

For applied explicit substitution, we have the following cases:

$$\|M\| \xrightarrow{\bar{\beta}} \|M'\| \Longrightarrow \|M[s]\| \xrightarrow{\bar{\beta}} \|M'[s]\|$$

$$\|s\| \xrightarrow{\bar{\beta}} \|s'\| \Longrightarrow \|M[s]\| \xrightarrow{\bar{\beta}} \|M[s']\|$$

They are both consequences of Lemma 9. □

In the following results, we write $\xrightarrow{\mathbf{CAU}^-}$ for the reduction relation in $\mathbf{CAU}^-$, i.e. a $\xrightarrow{\beta}$-step followed by τ-normalization, and $\xrightarrow{\mathbf{CAU}^-_\sigma}$ for the full rewriting system of $\mathbf{CAU}^-_\sigma$.

Lemma 11. *If* $M \xrightarrow{\bar{\beta}} N$ *then* $\sigma\tau(M) \xrightarrow{\boldsymbol{CAU}^-} \sigma\tau(N)$.

Proof. By the definition of $\bar{\beta}$-reduction, we obtain $M = \|M'\|, N = \|N'\|$ such that $M' \xrightarrow{\beta} N'$, then we prove $\sigma\tau(\|M'\|) = M'$ and $\sigma\tau(\|N'\|) = \tau(N')$. □

Lemma 12. *If* $M \xrightarrow{Beta} N$, *then* $\sigma\tau(M) \xrightarrow{\boldsymbol{CAU}^-} \sigma\tau(N)$

Proof. By Lemma 10 we get $\|M\| \xrightarrow{\bar{\beta}} \|N\|$. By Lemma 11 and Lemma 4, we prove the thesis. □

Lemma 13. *If* $M \xleftrightarrow{\sigma\tau} N$ *then* $\sigma\tau(M) = \sigma\tau(N)$

We can finally give the proof of the main theorem.

Proof of Theorem 3. *If* $M \xrightarrow{\boldsymbol{CAU}^{-}_{\sigma}} N$, *then* $\sigma\tau(M) \xrightarrow{\boldsymbol{CAU}^{-}} \sigma\tau(N)$.
We rewrite the hypothesis as

$$M \xrightarrow{\text{Beta}} \xleftarrow{\sigma\tau} \xrightarrow{\text{Beta}} \xleftarrow{\sigma\tau} \cdots N$$

Then the proof is a diagram chase based on Lemma 12 and Lemma 13. □