Triangulating Context Lemmas

Craig McLaughlin
LFCS
University of Edinburgh
United Kingdom
Craig.McLaughlin@ed.ac.uk

James McKinna
LFCS
University of Edinburgh
United Kingdom
James.McKinna@ed.ac.uk

Ian Stark
LFCS
University of Edinburgh
United Kingdom
Ian.Stark@ed.ac.uk

Abstract

The idea of a *context lemma* spans a range of programming-language models: from Milner's original through the CIU theorem to 'CIU-like' results for multiple language features. Each shows that to prove observational equivalence between program terms it is enough to test only some restricted class of contexts: applicative, evaluation, reduction, *etc.*

We formally reconstruct a distinctive proof method for context lemmas based on cyclic inclusion of three program approximations: by triangulating between 'applicative' and 'logical' relations we prove that both match the observational notion, while being simpler to compute. Moreover, the observational component of the triangle condenses a series of approximations covering variation in the literature around what variable-capturing structure qualifies as a 'context'.

Although entirely concrete, our approach involves no term dissection or inspection of reduction sequences; instead we draw on previous context lemmas using operational logical relations and biorthogonality. We demonstrate the method for a fine-grained call-by-value presentation of the simply-typed lambda-calculus, and extend to a CIU result formulated with frame stacks.

All this is formalised and proved in Agda: building on work of Allais et al., we exploit dependent types to specify lambda-calculus terms as well-typed and well-scoped by construction. By doing so, we seek to dispel any lingering anxieties about the manipulation of concrete contexts when reasoning about bound variables, capturing substitution, and observational equivalences.

CCS Concepts • **Theory of computation → Operational semantics**; **Program reasoning**; *Automated reasoning*; *Type theory*;

Keywords Context lemma, CIU theorem, Agda, Dependent types, Observational equivalence, Logical relations

ACM Reference Format:
Craig McLaughlin, James McKinna, and Ian Stark. 2018. Triangulating Context Lemmas. In *Proceedings of 7th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP'18).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3167081

CPP'18, January 8–9, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5586-5/18/01…$15.00
https://doi.org/10.1145/3167081

1 Introduction

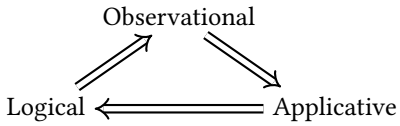
The problem of determining equivalence between programs or program fragments (terms) is well-studied and a variety of approaches have been developed [11–13, 15, 16, 19, 27–30, 32]. Typically, the equivalence of interest is *contextual* or *observational equivalence*. Roughly speaking, this relates two terms if they *behave equivalently* in any program context. The motivation for working with observational equivalence is that it supports powerful equational reasoning about code, such as rewriting via chains of equivalence proofs and the substitution of equivalent sub-terms ('equals for equals') in larger programs.

The exact meaning of 'program context' and 'behave equivalently' may vary: for example, program contexts often involve some kind of free-variable capture; while for equivalence it may be enough that two programs both terminate, or that they return equal values at ground type. The quantification over program contexts from the language itself turns out to make observational equivalence robust — independent of the class of available observations — and self-regulating, as a language with more elaborate features has a correspondingly larger collection of testing contexts.

Sadly, this quantification over all possible enclosing contexts can also make observational equivalence difficult to prove directly. However, if it can be captured by some restricted form of contexts then the proof burden is reduced without sacrificing reasoning power. Such correspondence results are known as *context lemmas*, following Milner's pioneering article [20], where he proved a context lemma for a typed combinatory logic with first-order function symbols.

Since then there have been many similar results published for a variety of calculi. The particular approach used to obtain the result varies, with many based on Howe's coinductive 'precongruence candidate' method [12, 13]. In contrast, following earlier proofs for the nu-calculus [32] and ReFS [30], we isolate a key part of their argument for independent study: proving a context lemma by showing that a certain triangle

of relations coincide. We propose this *triangulation* proof method as a general technique for proving context lemmas.



The triangle links three kinds of relation between terms:

Observational where terms are related by their behaviour in some class of contexts;

Applicative where function terms are related if they take equal arguments to related results;

Logical where function terms are related if they take related arguments to related results.

Rather than equivalences, we in fact work with inclusions between their associated *approximation* preorders; all results immediately carry over to equivalences.

While observational relations are defined by quantification over contexts, both applicative and logical relations are given by type structure. This structural character of these lower relations shapes the proofs along all three sides — in particular the implication from logical to observational arises from the fundamental theorem of logical relations.

The base of the triangle also shares out the distinctive properties of observational relations for powerful reasoning: applicative approximation is easily shown to be transitive, and logical approximation a congruence.

Observational relations require some chosen class of contexts, and in fact we see the triangle apex ramify into a chain of relations from behaviour in all contexts down through more restricted collections, with each relation naturally implying the next as the set of testing contexts becomes smaller. However, the triangulation result brackets all these observational relations between logical on the left and applicative on the right, proving that they collapse into each other. This collapse — that testing in all contexts is the same as testing over some restricted class — is the context lemma.

In this paper, we formally construct this triangulation proof method in a fine-grained call-by-value setting using the Agda interactive theorem prover. Leveraging state-of-the-art technology for dealing with well-typed and well-scoped terms [2], we show that, contrary to popularly held belief [15, 27], handling concrete contexts even in the formalised setting is straightforward, and moreover our method does not require inspection of reduction sequences [19, 32]. First, we apply the method to show the classical result for our simply-typed lambda calculus, namely that applicative contexts suffice (Section 3), using a big-step semantics. While we are not the first to undertake such a formalisation effort [3, 21], we believe our *concrete, first-order* representation to be more robust than previous attempts. To demonstrate the extensibility of the formalism, we generalise the triangle

Terms	M, N	::=	$V \mid \texttt{if}\ V\ \texttt{then}\ M\ \texttt{else}\ N \mid V\ V$
		$\mid$	$\texttt{let}\ x = M\ \texttt{in}\ N$
Values	V, U	::=	$x \mid \texttt{tt} \mid \texttt{ff} \mid n \mid () \mid \lambda x : A.M$
Types	A, B	::=	$\texttt{int} \mid \texttt{bool} \mid \texttt{unit} \mid A \to B$
Sugar	$M\ V$	::=	$\texttt{let}\ f = M\ \texttt{in}\ f\ V$ (where f is fresh)

Figure 1. $\lambda^{\to}_{FG}$ types and terms.

to the more powerful notion of CIU approximation (Section 4) by using frame stacks. Our intention is to extend our approach to more elaborate calculi involving effects, where such fine-grained stack-based approaches seem a better fit than big-step semantics. Demonstrating extensibility of our approach is moreover in the spirit of other proposals for mechanised metatheory developments [5]. In particular, we aim in future to scale the development to support core calculi with algebraic effects and handlers [6, 18]. The Agda source code is available online at:

https://www.github.com/cmcl/triangulating-context-lemmas

2 A Fine-Grained Call-By-Value Calculus

The work carried out here is performed with respect to a call-by-value calculus, $\lambda^{\to}_{FG}$, inspired by fine-grained call-by-value [17] (FGCBV) and a normal form for terms used by Pitts [29]. Figure 1 gives an informal context-free grammar for the calculus, with named λ and `let` bindings. The distinguishing features are the separation between the phrase classes for values and terms (we systematically elide the explicit lifting of values into terms) and the restriction to one term constructor, `let`, to express sequencing. For convenience, we introduce the syntactic sugar $M\ V$ for application of a term M to a value V, used in stating certain properties for $\lambda^{\to}_{FG}$. We consider a simple type discipline, where τ ranges over ground types ($\tau \in \{\texttt{bool}, \texttt{int}, \texttt{unit}\}$). For values, x ranges over an infinite set of variables, b ranges over $\{\texttt{tt}, \texttt{ff}\}$, and n over $\mathbb{Z}$. In the definitions and proofs that follow, $=$ and $\triangleq$ denote equalities that hold definitionally, and $\equiv$ denotes equalities which hold propositionally.

2.1 Formalising $\lambda^{\to}_{FG}$

Henceforth, reflecting our Agda [25] formalisation, we give $\lambda^{\to}_{FG}$ a de Bruijn presentation in a dependently-typed meta-language. In particular, we emphasise the following view on our representation and proofs about it: namely that it is a formalisation *in* a dependently typed metalanguage, and therefore *implementable in any implementation* of such a type theory. For this paper, we used the Agda theorem prover to check our constructions, but any system such as Coq [34] or Idris [7] would do just as well, modulo pragmatics. Part of our purpose in insisting on this point is to encourage readers to develop similar representations in their own tool of choice. In fact, at the cost of losing meta-language typing

guarantees, our development could be reprised in any other implemented theory supporting sufficiently rich inductive definitions, such as those of Isabelle/HOL [24]. However, our conviction is that a *dependently-typed* meta-language is a precision tool, acutely well-adapted to the demands of typed object-language representation. For the sake of exposition, we focus on the mathematics, rather than its rendering in Agda concrete syntax; readers interested in the latter should consult our source code. For their convenience, we provide a Rosetta Stone in Table 1 at the end of the paper, mediating between informal concepts, LaTeX rendering, and their Agda formalisation.

We follow the discipline advocated by Allais *et al.* [2], henceforth referred to as the ACMM framework, which allows us to represent the informal syntax of Figure 1 in terms of inductive families of types [10]. For each syntactic category of the object language — variables, values, terms — we define such a family indexed by object-language type A and context Γ. Thus we express $\lambda^{\rightarrow}_{FG}$ directly via its typing rules, as in Figure 2, so that values and terms are by construction well-typed and well-scoped. As a consequence, we sometimes need to introduce a renaming, in particular the special case of *weakening*, to ensure the type-and-scope correctness of terms. For example, in the syntactic sugar for application, the argument is applied in an environment extended with the binding for the function:

$$\frac{\dfrac{\vdots}{\Gamma \vdash M : A \to B} \qquad \dfrac{\Gamma \vdash V : A}{\Gamma, A \to B \vdash \mathit{weak}(V) : A}}{\Gamma \vdash M\ V : B}$$

where *weak* takes Γ-terms to Γ, A-terms for any type A by shifting de Bruijn indices. From now on, for readability, we elide any explicit weakening in terms. The principal ACMM features we exploit are as follows.

- ACMM uses first-order abstract syntax (FOAS) in its term representation (rather than weak higher-order abstract syntax (HOAS), as used, for example, by Despeyroux et al. in [9]), specifically a de Bruijn index [8] variable representation: so α-convertibility is syntactic equality; variables $\mathtt{var}_k$ are declared in terms of offsets k (generated by zero and successor) into an environment Γ consisting of a list of types; function values $\lambda_A\, M$ comprise a type label A and a well-scoped term body M; and similarly for `let`-expressions $\mathtt{let}_A\, M\, N$. In each case, we suppress object-language type labels where they are unambiguous in context.
- Whereas, for operations and proofs defined over terms, ACMM does use (weak) HOAS: thus renaming, capture-avoiding substitution, *etc.* are all instances of a generic Kripke-style traversal over the phrase classes of the language. Such traversals are specified by supplying type families $\mathcal{V}, \mathcal{MV}, \mathcal{MT}$ to interpret respectively each of variables, values, and terms, subject to suitable closure conditions. That for the λ constructor is (somewhat simplified) as follows:

$$\frac{\Box\,(\mathcal{V}\ A\ \Delta \longrightarrow \mathcal{MT}\ B\ \Delta)}{\mathcal{MV}\ (A \Rightarrow B)\ \Gamma}$$

where the modality $\Box$ quantifies over all context extensions $\Gamma \subseteq \Delta$ by *renamings*. A particular practical consequence is that, by design, only *value* substitutions are expressible in the formalism — which in fact accords with usual informal practice when presenting fine-grained call-by-value.

$$\begin{array}{lrcl} \text{Environments} & \Gamma, \Delta & ::= & \cdot \mid \Gamma, A \\ \text{Indices} & k & ::= & 0 \mid \mathsf{succ}\, k \end{array}$$

$$\textsc{zero}\ \frac{}{0 : (A \in \Gamma, A)} \qquad \textsc{succ}\ \frac{k : (B \in \Gamma)}{\mathsf{succ}\, k : (B \in \Gamma, A)} \qquad \textsc{var}\ \frac{k : (A \in \Gamma)}{\Gamma \vdash \mathtt{var}_k : A}$$

$$\textsc{boolean}\ \frac{}{\Gamma \vdash b : \mathtt{bool}}\ (b = \mathtt{tt}, \mathtt{ff}) \qquad \textsc{int}\ \frac{}{\Gamma \vdash n : \mathtt{int}}\ (n \in \mathbb{Z})$$

$$\textsc{unit}\ \frac{}{\Gamma \vdash () : \mathtt{unit}} \qquad \textsc{fun}\ \frac{\Gamma, A \vdash M : B}{\Gamma \vdash \lambda_A\, M : A \to B}$$

$$\textsc{ifThenElse}\ \frac{\Gamma \vdash V : \mathtt{bool} \qquad \Gamma \vdash N_{\mathtt{tt}} : A \qquad \Gamma \vdash N_{\mathtt{ff}} : A}{\Gamma \vdash \mathtt{if}\ V\ \mathtt{then}\ N_{\mathtt{tt}}\ \mathtt{else}\ N_{\mathtt{ff}} : A}$$

$$\textsc{app}\ \frac{\Gamma \vdash V : A \to B \qquad \Gamma \vdash U : A}{\Gamma \vdash V\ U : B}$$

$$\textsc{let}\ \frac{\Gamma \vdash M : A \qquad \Gamma, A \vdash N : B}{\Gamma \vdash \mathtt{let}_A\, M\, N : B}$$

Figure 2. $\lambda^{\rightarrow}_{FG}$ typing rules

Our choice of ACMM can be seen as contributing to a now decades-old debate on object-language representations, particularly with respect to binding structures. There is not space to do justice to all the issues here (for an interesting discussion, see for example [31]), nor is it the main focus of the paper. Nevertheless, some comments may be in order.

- *Adequacy* for a FOAS representation, by contrast with varieties of HOAS, is straightforward — even trivial — as part of grounds for belief in the formalisation. We view this as particularly effective in the case of formalising object-language *contexts* (in Section 2.2) in defining varieties of observational approximation,

$$\textsc{ifThenElse}\ \frac{}{\texttt{if}\ b\ \texttt{then}\ N_{\texttt{tt}}\ \texttt{else}\ N_{\texttt{ff}} \rightsquigarrow N_b} \qquad \textsc{appBeta}\ \frac{}{(\lambda_A\, M)\ V \rightsquigarrow M[V]}$$

Figure 3. $\lambda^{\rightarrow}_{FG}$ primitive reduction semantics

$$\textsc{primRed}\ \frac{M \rightsquigarrow M'}{M \Rightarrow M'} \qquad \textsc{letValue}\ \frac{}{\texttt{let}_A\ V\ M \Rightarrow M[V]}$$

$$\textsc{letRed}\ \frac{M \Rightarrow M'}{\texttt{let}_A\ M\ N \Rightarrow \texttt{let}_A\ M'\ N}$$

Figure 4. $\lambda^{\rightarrow}_{FG}$ small-step operational semantics

where the informal treatment of variables and binding is a potential minefield.

- There is no doubting the *convenience* of HOAS in terms of definability of operations such as substitution; nor, as the ACMM authors point out, in terms of flexibility of 'logical relations' arguments about the behaviour of such operations: we make heavy use of such results. Arguably, given our use of 'logical' definitions of relations such as in Section 3.2, we could perhaps have done more to leverage this aspect of ACMM. We leave this to future work.

Definition 2.1 (Simultaneous Substitution)**.** For typing environments Γ and Δ let $\theta : \Gamma \vDash \Delta$ denote a simultaneous substitution θ of variables in environment Γ by $\lambda^{\rightarrow}_{FG}$ values in environment Δ. Informally, θ takes Γ-terms to Δ-terms: given $\Gamma \vdash M : A$ we may apply the substitution to M and obtain $\Delta \vdash \theta(M) : A$. In the special case of $\theta : \Gamma, A \vDash \Gamma$ given at $\texttt{var}_0$ by value $\Gamma \vdash V : A$ and everywhere else the identity we write $\theta(M)$ as $M[V]$ in the usual way. For the special case of $\theta : \Gamma \vDash \cdot$ we say that θ is a Γ-*closing* substitution. If $\theta : \Gamma \vDash \Delta$ and $\theta' : \Delta \vDash \Theta$ then $\theta' \circ \theta : \Gamma \vDash \Theta$ denotes the composition of the simultaneous substitutions.

Definition 2.2. Given $\theta : \Gamma \vDash \Delta$, define θ_k to be the value $\Delta \vdash V : A$ specified for the variable $\Gamma \vdash \texttt{var}_k : A$. Then, θ_0 denotes the value specified for $\texttt{var}_0$. For $\theta : \Gamma, A \vDash \Delta$, define the simultaneous substitution $(\texttt{succ}\,\theta) : \Gamma \vDash \Delta$ by its behaviour on variables $\Gamma \vdash \texttt{var}_k : A$, for all indices k:

$$(\texttt{succ}\,\theta)_k = \theta_{(\texttt{succ}\,k)}\,.$$

Moreover, any $\theta : \Gamma, A \vDash \Delta$ is extensionally equal to that of the substitution obtained by extending $\texttt{succ}\,\theta$ by θ_0.

We define a small-step operational semantics for $\lambda^{\rightarrow}_{FG}$ via a primitive reduction on closed terms, in Figure 3, which is then contained within the general reductions of Figure 4. The separation of these two relations is inspired by Pitts [29] and

$$\textsc{value}\ \frac{\cdot \vdash V : A}{V \Downarrow_A V} \qquad \textsc{if}\ \frac{N_b \Downarrow_A V}{\texttt{if}\ b\ \texttt{then}\ N_{\texttt{tt}}\ \texttt{else}\ N_{\texttt{ff}} \Downarrow_A V}$$

$$\textsc{app}\ \frac{M[V] \Downarrow_B U}{(\lambda_A\, M)\ V \Downarrow_B U} \qquad \textsc{let}\ \frac{M \Downarrow_A V \qquad N[V] \Downarrow_B U}{\texttt{let}_A\ M\ N \Downarrow_B U}$$

Figure 5. $\lambda^{\rightarrow}_{FG}$ big-step operational semantics

is used when we define frame stack evaluation in Section 4. Both relations are type-correct by construction; the proof of type preservation is immediate.

Figure 5 presents $\lambda^{\rightarrow}_{FG}$ evaluation semantics as a type-indexed relation between closed terms and values. We omit typing annotations where unambiguous. Evaluation is closed under the reduction relations: proof is straightforward by induction over derivations.

Lemma 2.3 (Reduction Respects Evaluation)**.**
If $M \rightsquigarrow M'$ or $M \Rightarrow M'$ then $M \Downarrow V$ if and only if $M' \Downarrow V$.

2.2 Formalising Contexts

As a step towards formalising observational approximation, we treat contexts as an extension of $\lambda^{\rightarrow}_{FG}$ syntax; and thanks to ACMM the grammar of these concrete contexts is directly expressible as an inductive definition. A *term context* $\mathcal{C}$ is a possibly-open term of arbitrary type containing zero or more occurrences of a well-typed and well-scoped hole. Analogously, a *value context* $\mathcal{V}$ is a possibly-open value of arbitrary type containing zero or more occurrences of such a hole. Naturally enough our formalisation enforces well-typed and well-scoped contexts, by construction, in the same way it does for terms and values.

The usual notion of a *variable-capturing* context (VCC) allows a context to capture variables occurring free in the term that fills the hole. In Figure 6 we define VCCs for $\lambda^{\rightarrow}_{FG}$, with $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$ ranging over term contexts and $\mathcal{V}$, $\mathcal{W}$ over value contexts. The form of a well-typed and well-scoped context $\mathcal{C}$ is $\Delta \vdash \mathcal{C} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\text{VCC}} B$, which says that term context $\mathcal{C}$ has type B in environment Δ and contains zero or more occurrences of a hole of type A in an environment Γ. For such a well-typed term context $\mathcal{C}$, there is an operation, *instantiation*, denoted by $\mathcal{C}\langle\!\langle M \rangle\!\rangle^{\text{VCC}}$ which fills the holes in the term context $\mathcal{C}$ with M. We make a customary abuse of notation in overloading chevrons '$\langle\!\langle\rangle\!\rangle$' to represent both the occurrence of a hole in a context and the operation of instantiation. Instantiation is (yet) another structural traversal, where the only (base) case of interest is that of the hole constructor, replacing the hole with M:

$$\langle\!\langle - \rangle\!\rangle \langle\!\langle M \rangle\!\rangle^{\text{VCC}} = M\,.$$

$$\textsc{vccHole}\ \frac{}{\Gamma \vdash \langle\!\langle - \rangle\!\rangle : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} A} \qquad \textsc{vccTrm}\ \frac{\Delta \vdash M : B}{\Delta \vdash \mathtt{trm}\, M : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} B}$$

$$\textsc{vccFun}\ \frac{\Delta, B \vdash \mathcal{D} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} C}{\Delta \vdash \lambda_B\, \mathcal{D} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} (B \to C)}$$

$$\textsc{vccIfThenElse}\ \frac{\Delta \vdash \mathcal{V} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} \mathtt{bool} \qquad \Delta \vdash C_{\mathtt{tt}} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} B \qquad \Delta \vdash C_{\mathtt{ff}} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} B}{\Delta \vdash \mathtt{if}\ \mathcal{V}\ \mathtt{then}\ C_{\mathtt{tt}}\ \mathtt{else}\ C_{\mathtt{ff}} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} B}$$

$$\textsc{vccApp}\ \frac{\Delta \vdash \mathcal{V} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} (B \to C) \qquad \Delta \vdash \mathcal{W} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} B}{\Delta \vdash \mathcal{V}\, \mathcal{W} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} B}$$

$$\textsc{vccLet}\ \frac{\Delta \vdash \mathcal{D} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} B \qquad \Delta, B \vdash \mathcal{E} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} C}{\Delta \vdash \mathtt{let}_B\, \mathcal{D}\, \mathcal{E} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} C}$$

Figure 6. $\lambda^{\rightarrow}_{FG}$ typing rules for variable-capturing contexts

By virtue of our de Bruijn encoding, we avoid traditional concerns (as in [15, 27, for example]) associated with α-equivalence of VCCs. However, VCCs are still a little inconvenient to work with because they force holes to match exactly the enclosing scope of their context, even up to the ordering of the variables captured (see the vccHole rule in Figure 6).

To manage such concerns we employ a different notion of *value-substituting* contexts (VSCs), where each occurrence of a hole carries an appropriate (well-typed) substitution, as in rule vscHole below. This ensures that the instantiation operation $C\langle\!\langle M \rangle\!\rangle^{\mathrm{VSC}}$ is itself well-typed and well-scoped, and again definable by a simple traversal. Typing judgements for VSCs are the same as for VCCs except we replace the vccHole rule with the following.

$$\textsc{vscHole}\ \frac{\theta : \Gamma \vDash \Delta}{\Delta \vdash \langle\!\langle \theta - \rangle\!\rangle : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VSC}} A}$$

Instantiation for VSCs is as with VCCs except for the base case, where a hole is filled by applying the substitution provided.

$$\langle\!\langle \theta - \rangle\!\rangle \langle\!\langle M \rangle\!\rangle^{\mathrm{VSC}} = \theta(M)$$

Every VCC is trivially a VSC, by annotating each hole with an identity substitution. For the rest of the paper we use unadorned chevrons for instantiation when it is clear from the context which version of contexts is being used.

Lassen defines a notion of variable-capturing context [16] that is slightly different to ours, equivalent to each hole carrying a *renaming* rather than a general value substitution. This lies strictly between our VCCs and VSCs, as any renaming is naturally a substitution. Later, we shall see that relations based on either VCCs or VSCs coincide, and so also with ones based on Lassen's variant.

2.3 Observational Approximation

This section formally defines the notion of observational approximation, alluded to in the introduction, for $\lambda^{\rightarrow}_{FG}$. Its definition can be traced back to Morris [23] and builds on the account of term contexts and substitutions in the previous section.

We begin with a relation transformer lifting any relation on values to a relation on terms.

Definition 2.4. For closed terms M, N of type A and $\mathcal{R}$ a binary relation on closed values of type A,

$$M\ [\mathcal{R}]^T\ N \triangleq \forall V.\, M \Downarrow_A V \implies \exists U.\, N \Downarrow_A U \wedge V\ \mathcal{R}\ U$$

This relation transformer is closed under primitive reduction and expansion. We state and sketch a proof for left-closure under $\rightsquigarrow$.

Lemma 2.5 ($[\ ]^T$ left-closed under $\rightsquigarrow$)**.** *For all closed terms M, N of type A and $\mathcal{R}$ a binary relation on closed values of type A, the following properties hold.*

1. *If $M \rightsquigarrow P$ and $M\ [\mathcal{R}]^T\ N$ then $P\ [\mathcal{R}]^T\ N$*
2. *If $M \rightsquigarrow P$ and $P\ [\mathcal{R}]^T\ N$ then $M\ [\mathcal{R}]^T\ N$*

Proof. By the definition of $[\ ]^T$ and Lemma 2.3. □

We next define a basic relation that drives all our other notions of approximation. This aims to capture what values can be directly distinguished, without further context or evaluation.

Definition 2.6 (Ground equivalence)**.** Let $\cong_A$ be the equivalence relation defined on closed values of type A by:

- $V \cong_\tau U \triangleq V = U$, for all $V, U : \tau$,
- $\lambda_A\, M \cong_{A \to B} \lambda_A\, N$ for all M, N

One reason to factor this out in our development is the potential for incorporating non-trivial equivalences on ground values (such as for adding primitive operations and their definitions) in a similar fashion to Johann et al. [14] (see Section 5 for more on this).

Although this particular relation is symmetric, the definitions we build on it are biased to give approximation preorders. In particular this anticipates calculi with nontermination, where termination is a basic observable.

Now we are in a position to define observational approximation in terms of ground equivalence. First, we have a definition of approximation parametric in the notion of contexts we consider (VSCs, VCCs,. . .). Let $\mathcal{K}$ range over these notions of contexts. A *program* is a closed term of arbitrary

type. A *program $\mathcal{K}$-context* is a closed term $\mathcal{K}$-context ranged over by $\mathcal{P}, \mathcal{Q}$, i.e. $\cdot \vdash \mathcal{P} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathcal{K}} B$; where for convenience we usually omit the empty environment.

Definition 2.7 ($\mathcal{K}$ Approximation)**.** If M, N are terms of type A in context Γ then

$$\Gamma \vdash M \lesssim_A^{\mathcal{K}} N$$

asserts that for all program contexts $\mathcal{P} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathcal{K}} B$,

$$\mathcal{P}\langle\!\langle M \rangle\!\rangle^{\mathcal{K}} \; [\cong_B]^T \; \mathcal{P}\langle\!\langle N \rangle\!\rangle^{\mathcal{K}} .$$

From all $\mathcal{K}$-contextual approximations, we choose as primary the one with the largest class of contexts — thereby able to make the finest discrimination between terms.

Definition 2.8 (Observational Approximation)**.** Let observational approximation, written $\lesssim^{\mathrm{VSC}}$, be Definition 2.7 instantiated with $\mathcal{K} = \mathrm{VSC}$.

Now we can establish our first inclusion of approximations.

Lemma 2.9. *If* $\Gamma \vdash M \lesssim_A^{VSC} N$ *then* $\Gamma \vdash M \lesssim_A^{VCC} N$.

Proof. Assume given terms $\Gamma \vdash M, N : A$ with $\Gamma \vdash M \lesssim_A^{\mathrm{VSC}} N$ and VCC $\mathcal{P} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} B$. From $\mathcal{P}$ we can build matching VSC $\mathcal{Q}$ where each hole carries the identity substitution. The result follows by using our VSC approximation with M and N, and the property that for all terms $\Gamma \vdash M' : A$, $\mathcal{P}\langle\!\langle M' \rangle\!\rangle^{\mathrm{VCC}} \equiv \mathcal{Q}\langle\!\langle M' \rangle\!\rangle^{\mathrm{VSC}}$ (proved by induction on the structure of $\mathcal{P}$). □

Definition 2.7 can readily be extended to an equivalence, denoted by $\simeq^{\mathcal{K}}$:

$$\Gamma \vdash M \simeq_A^{\mathcal{K}} N \;\triangleq\; \Gamma \vdash M \lesssim_A^{\mathcal{K}} N \;\wedge\; \Gamma \vdash N \lesssim_A^{\mathcal{K}} M .$$

In the remainder of this paper results will be stated for the approximations only but can be straightforwardly lifted to equivalences [30, 32].

3 Big Steps for a Context Lemma

In this section we define three approximation relations and prove a triangle of implications which together lead to a context lemma inspired by Milner's original for combinatory logic [20].

As noted in the introduction, proving observational approximation directly can be arduous owing to the requirement to consider all possible program contexts. Happily, Milner [20] showed that for typed combinatory logic it is enough to consider only contexts where the hole occurs in function position applied to some arguments. The following definition captures such contexts in our setting.

Definition 3.1 (Applicative Substituting Contexts)**.** The *applicative substituting contexts* (ASCs) are the restricted class

$$\textsc{betaV-Refl}\ \frac{}{M \rightsquigarrow^{\beta V} M} \qquad \textsc{betaV-Step}\ \frac{M \rightsquigarrow^{\beta V} (\lambda_A\, N)\; V}{M \rightsquigarrow^{\beta V} N[V]}$$

$$\textsc{beta-}\beta\textsc{V}\ \frac{M \rightsquigarrow^{\beta V} N}{M \rightsquigarrow^{\beta} N} \qquad \textsc{beta-Let}\ \frac{M \rightsquigarrow^{\beta} M'}{\mathtt{let}_A\, M\, N \rightsquigarrow^{\beta} \mathtt{let}_A\, M'\, N}$$

Figure 7. $\lambda_{FG}^{\rightarrow}$ iterated beta-reduction with respect to sequencing

of contexts defined by the following inference rules:

$$\textsc{ascHole}\ \frac{\theta : \Gamma \vDash \Delta}{\Delta \vdash \langle\!\langle \theta - \rangle\!\rangle : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{ASC}} A}$$

$$\textsc{ascApp}\ \frac{\Delta \vdash \mathcal{D} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{ASC}} (B \to C) \qquad \Delta \vdash V : B}{\Delta \vdash \mathcal{D}\; V : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{ASC}} C}$$

Definition 3.2 (ASC Approximation)**.** Let ASC approximation be Definition 2.7 with $\mathcal{K} = \mathrm{ASC}$.

Having introduced the required notation we may now formally state the context lemma we aim to prove in this section. Firstly, in our setting, Milner's original context lemma states that for all open $\lambda_{FG}^{\rightarrow}$ terms $\Gamma \vdash M, N : A$,

$$\Gamma \vdash M \lesssim_A^{\mathrm{ASC}} N \iff \Gamma \vdash M \lesssim_A^{\mathrm{VCC}} N.$$

However, we have taken our notion of VSCs as primary, so our context lemma becomes

$$\Gamma \vdash M \lesssim_A^{\mathrm{ASC}} N \iff \Gamma \vdash M \lesssim_A^{\mathrm{VSC}} N.$$

Before proving the above result we define some auxiliary notions.

Definition 3.3 (Beta-Redexes from Substitution)**.** Let θ be a Δ-closing substitution. For $\Delta \vdash \mathcal{D} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{VCC}} B$, define $\mathcal{D}^{\theta}$ by induction on the size of Δ:

$$\mathcal{D}^{\theta} = \begin{cases} \mathcal{D} & \Delta = \cdot \\ ((\lambda_C\, \mathcal{D})\; \theta_0)^{(\mathrm{succ}\, \theta)} & \Delta = \Delta', C \end{cases}$$

$\mathcal{D}^{\theta}$ represents a new (closed) VCC obtained from $\mathcal{D}$ by constructing a sequence of beta-redexes on the outside of $\mathcal{D}$ from the substitution θ. For $\Delta \vdash M : A$, let M^{θ} denote the analogous operation for constructing a closed term from M and θ.

Figure 7 defines iterated beta-reduction with respect to term sequencing. Both relations satisfy analogous properties to Lemmas 2.3 and 2.5. Then we have the obvious result

Lemma 3.4. *For* $\Delta \vdash M : A$, *and* Δ*-closing substitution* θ, *then we have*

$$M^{\theta} \rightsquigarrow^{\beta V} \theta(M)$$

Definition 3.5. Define the operation $\star : ASC \to VCC$, which transforms a closed ASC into a closed VCC, by induction on the structure of the ASC:

$$\star(\langle\!\langle \theta - \rangle\!\rangle) = \langle\!\langle - \rangle\!\rangle^{\theta}$$

$$\star(\mathcal{P}\ V) = \star(\mathcal{P})\ V$$

where the ascApp case uses our syntactic sugar, extended to $\mathcal{K}$ contexts.

Lemma 3.6. *For all $\Gamma \vdash M : A, \Delta \vdash C : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{VCC} B$, and Δ-closing substitutions, θ, we have*

$$(C^{\theta})\langle\!\langle M \rangle\!\rangle \equiv (C\langle\!\langle M \rangle\!\rangle)^{\theta}$$

Proof. By induction on the size of the environment Δ. □

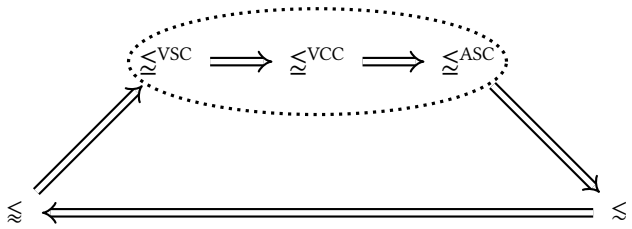
Lemma 3.7. *If $\Gamma \vdash M \lesssim_{\simeq A}^{VCC} N$ then $\Gamma \vdash M \lesssim_{\simeq A}^{ASC} N$.*

Proof. Assume given terms $\Gamma \vdash M, N : A, \Gamma \vdash M \lesssim_{\simeq A}^{\mathrm{VCC}} N$ and ASC $\mathcal{P} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{ASC}} B$. The proof proceeds by induction on the structure of $\mathcal{P}$.

If $\mathcal{P} = \langle\!\langle \theta - \rangle\!\rangle$, for some Γ-closing substitution θ, then the result follows from Lemmas 3.6 and 3.4, using $[\ ]^T$'s left- and right-closure under $\rightsquigarrow^{\beta V}$.

Otherwise, $\mathcal{P} = \mathcal{Q}\ V$, for some $\mathcal{Q} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{ASC}}(B \to C)$ and $V : B$. The result follows by applying the approximation assumption to $\star(\mathcal{P})$ (Definition 3.5) and using $[\ ]^T$'s left- and right-closure under $\rightsquigarrow^{\beta}$. □

Lemmas 2.9 and 3.7 give one direction of our context lemma for $\lambda_{FG}^{\rightarrow}$. Not surprisingly, the other direction is more challenging. This is where we adopt the triangulation proof method described in the introduction. That is, we concern ourselves with establishing the following cycle of implications.



The contextual approximations for VSCs, VCCs and ASCs form the apex of the triangle, while $\lesssim$ is *applicative* approximation and $\lessapprox$ is *logical* approximation, both to be defined shortly. We have already shown the implications within the dotted ellipse (Lemmas 2.9 and 3.7), and the next two sections will address those forming the triangle.

3.1 Applicative Approximation

The definition of applicative approximation roughly follows Stark's definition [32].

Definition 3.8 (Applicative Approximation)**.** We define applicative approximation $\lesssim$ using two mutually recursive definitions, one for values and one for terms.

Define the relation $V_1 \lesssim_A^{val} V_2$ for closed values V_1, V_2 inductively on the structure of type A:

$$\textsc{AppGndApx}\ \frac{V_1 \cong_\tau V_2}{V_1 \lesssim_\tau^{val} V_2} \qquad \textsc{AppAbsApx}\ \frac{\forall\, V : A.\ M_1[V] \lesssim_B^{trm} M_2[V]}{\lambda_A\, M_1 \lesssim_{A\to B}^{val} \lambda_A\, M_2}$$

Define the relation $M_1 \lesssim_A^{trm} M_2$ for closed terms M_1, M_2 of type A by the following definition:

$$M_1 \lesssim_A^{trm} M_2 \ \triangleq\ M_1 \left[\lesssim_A^{val}\right]^T M_2$$

Just like for observational approximation, applicative approximation can be stated for open terms. We employ simultaneous substitutions to close over the environment.

Definition 3.9 (Open Applicative Approximation)**.** Given $\Gamma \vdash M_1 : A$ and $\Gamma \vdash M_2 : A$, we say that M_1 *applicatively approximates* M_2, written $\Gamma \vdash M_1 \lesssim_A M_2$, if and only if for all Γ-closing substitutions θ we have $\theta(M_1) \lesssim_A \theta(M_2)$.

A relationship between ASC approximation and applicative approximation can be established from which we obtain one side of the triangle.

Theorem 3.10. $\Gamma \vdash M \lesssim_{\simeq A}^{ASC} N \implies \Gamma \vdash M \lesssim_A^{trm} N$

Proof. By induction on the type A. □

Lemma 3.11. *Observational approximation implies applicative approximation:*

$$\Gamma \vdash M \lesssim_{\simeq A}^{VSC} N \implies \Gamma \vdash M \lesssim_A^{trm} N$$

Proof. By Lemmas 2.9 and 3.7 with Theorem 3.10. □

3.2 Logical Approximation

Logical approximation differs with respect to applicative approximation only in its handling of functions.

Definition 3.12 (Logical Approximation)**.** Define logical approximation by the mutually recursive relations $\lessapprox^{val}$ and $\lessapprox^{trm}$. The relation $V_1 \lessapprox_A^{val} V_2$ for closed values V_1, V_2 is defined recursively on the structure of type A:

$$\textsc{LogGndApx}\ \frac{V_1 \cong_\tau V_2}{V_1 \lessapprox_\tau^{val} V_2}$$

$$\textsc{LogAbsApx}\ \frac{\forall\, V_1, V_2.\ V_1 \lessapprox_A^{val} V_2 \implies M_1[V_1] \lessapprox_B^{trm} M_2[V_2]}{\lambda_A\, M_1 \lessapprox_{A\to B}^{val} \lambda_A\, M_2}$$

The relation $M_1 \lessapprox_A^{trm} M_2$ for closed terms M_1, M_2 of type A is defined using $[\ ]^T$ and $\lessapprox^{val}$:

$$M_1 \lessapprox_A^{trm} M_2 \ \triangleq\ M_1 \left[\lessapprox_A^{val}\right]^T M_2$$

To extend logical approximation to open terms we define the notion of logical approximation of substitutions.

$$\textsc{LogApxIfThenElse}\quad \frac{B \lessapprox^{val}_{bool} B' \qquad L \lessapprox^{trm}_{A} L' \qquad R \lessapprox^{trm}_{A} R'}{\texttt{if } B \texttt{ then } L \texttt{ else } R \lessapprox^{trm}_{A} \texttt{if } B' \texttt{ then } L' \texttt{ else } R'}$$

$$\textsc{LogApxApp}\quad \frac{F \lessapprox^{val}_{A\to B} G \qquad U \lessapprox^{val}_{A} V}{F\ U \lessapprox^{trm}_{B} G\ V}$$

$$\textsc{LogApxLet}\quad \frac{M \lessapprox^{trm}_{A} M' \qquad x : A \vdash N \lessapprox^{trm}_{B} N'}{\texttt{let}_A\ M\ N \lessapprox^{trm}_{B} \texttt{let}_A\ M'\ N'}$$

Figure 8. $\lessapprox$ is closed under the compound term formers

Definition 3.13 (Logical Approximation of Substitutions)**.** For Γ-closing substitutions, θ, θ', $\theta \lessapprox_\Gamma \theta'$ holds if and only if, for all $k : (A \in \Gamma)$, $\theta_k \lessapprox_A \theta'_k$. For $\theta, \theta' : \Gamma \vDash \Delta$, $\Delta \vdash \theta \lessapprox_\Gamma \theta'$ holds if and only if, for all Δ-closing substitutions, θ_1, θ_2 such that $\theta_1 \lessapprox_\Delta \theta_2$ holds then $\theta_1 \circ \theta \lessapprox_\Gamma \theta_2 \circ \theta'$ holds.

Definition 3.14 (Open Logical Approximation)**.** Assume given $\Gamma \vdash M_1 : A$ and $\Gamma \vdash M_2 : A$, then M_1 logically approximates M_2, written $\Gamma \vdash M_1 \lessapprox_A M_2$, if and only if, for all Γ-closing substitutions θ_1, θ_2 if $\theta_1 \lessapprox_\Gamma \theta_2$ then $\theta_1(M_1) \lessapprox_A \theta_2(M_2)$.

Logical approximation is closed under primitive reduction, and expansion.

Lemma 3.15 ($\lessapprox^{trm}$ left-closed under $\rightsquigarrow$)**.** *If $M \rightsquigarrow P$ and $M \lessapprox^{trm}_A N$ then $P \lessapprox^{trm}_A N$, and if $M \rightsquigarrow P$ and $P \lessapprox^{trm}_A N$ then $M \lessapprox^{trm}_A N$.*

Lemma 3.16 ($\lessapprox^{trm}$ right-closed under $\rightsquigarrow$)**.** *If $N \rightsquigarrow P$ and $M \lessapprox^{trm}_A N$ then $M \lessapprox^{trm}_A P$, and if $N \rightsquigarrow P$ and $M \lessapprox^{trm}_A P$ then $M \lessapprox^{trm}_A N$*

The following lemma follows directly from the above definition of open logical approximation.

Lemma 3.17. *If $\Gamma \vdash M_1 \lessapprox^{trm}_A M_2$ and $\Delta \vdash \theta_1 \lessapprox_\Gamma \theta_2$ then $\Delta \vdash \theta_1(M_1) \lessapprox^{trm}_A \theta_2(M_2)$.*

Lemma 3.18 (Logical Apx. Term Closure)**.** *Logical approximation is closed under the compound term formers (Figure 8).*

Proof. By Lemmas 3.15 and 3.16, and the LET rule for big-step evaluation. □

Lemma 3.19 (Fundamental Theorem of Logical Relations)**.** *Logical approximation is reflexive:*

1. $\Gamma \vdash V \lessapprox^{val}_A V$
2. $\Gamma \vdash M \lessapprox^{trm}_A M$

Proof. Perform simultaneous induction on the typing derivations $\Gamma \vdash V : A$ and $\Gamma \vdash M : A$ using Lemma 3.18. □

3.3 Completing the Triangle

It remains to establish the other two implications of the triangle. First, we link applicative and logical approximation with the following generalised transitivity property [29, 30]:

Lemma 3.20. *The following transitivity properties hold:*

1. *If $U \lesssim^{val}_A V$ and $V \lessapprox^{val}_A W$ then $U \lessapprox^{val}_A W$*
2. *If $M \lesssim^{trm}_A N$ and $N \lessapprox^{trm}_A P$ then $M \lessapprox^{trm}_A P$*

Proof. By simultaneous induction on the structure of type A. □

Using the Fundamental Theorem of Logical Relations and Lemma 3.20, we obtain the base implication of our triangle.

Lemma 3.21. *Applicative approximation implies logical approximation:*

$$\Gamma \vdash M \lesssim^{trm}_A N \implies \Gamma \vdash M \lessapprox^{trm}_A N$$

Now we establish a relationship between observational and logical approximations by first showing that logical approximation is closed under all VSCs.

Lemma 3.22. *If $\Gamma \vdash M_1 \lessapprox^{trm}_A M_2$ and $\Delta \vdash C : \langle\!\langle \Gamma \vdash A \rangle\!\rangle B$ then,*

$$\Delta \vdash C\langle\!\langle M_1 \rangle\!\rangle \lessapprox^{trm}_B C\langle\!\langle M_2 \rangle\!\rangle$$

Proof. By induction on the derivation of $\Delta \vdash C : \langle\!\langle \Gamma \vdash A \rangle\!\rangle B$ using Lemma 3.18 for the compound term formers, Lemma 3.17 for the VSCHOLE case, and Lemma 3.19 for the case where there is no hole in C. □

Lemma 3.23. *Logical approximation implies observational approximation:*

$$\Gamma \vdash M_1 \lessapprox^{trm}_A M_2 \implies \Gamma \vdash M_1 \lesssim^{VSC}_A M_2$$

Proof. Instantiate the context C of Lemma 3.22 with program context $\cdot \vdash \mathcal{P} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle B$ from the definition of VSC approximation. □

We can now complete the cycle, which in turn yields the desired context lemma: that applicative contexts are sufficient to characterise observational approximation.

Theorem 3.24 (Big-Step Triangulation)**.** *Observational, applicative, and logical approximation coincide.*

Proof. Using Lemmas 3.11, 3.21 and 3.23. □

Corollary 3.25 (Context Lemma)**.** *For all open $\lambda^{\to}_{FG}$ terms $\Gamma \vdash M, N : A$, $\Gamma \vdash M \lesssim^{ASC}_A N$ if and only if $\Gamma \vdash M \lesssim^{VSC}_A N$.*

4 Extension to Frame Stacks

Since Milner's original context lemma, a number of similar results have appeared for more complex calculi. In particular, for calculi with effects (such as state) applicative contexts are not sufficient and one requires a more powerful notion known as *Closed Instantiations of Uses* (CIU) approximation [19].

In this section we consider a different triangle of approximations involving *frame stacks* [29]. We parallel the ReFS proofs of Pitts and Stark [30] recasting the applicative and logical approximations described in Section 3 in terms of frame stack evaluation. First though, we define CIU contexts and relate them to VCCs just like that for ASCs. We omit some proofs since they follow similar reasoning. For an alternative account of a similar development, the reader may wish to consult the chapter by Pitts [29] in Pierce's book [26].

Definition 4.1 (CIU Contexts)**.** Let *CIU contexts* be the restricted class of contexts defined by the following inference rules:

$$\textsc{ciuHole}\ \frac{\theta : \Gamma \vDash \Delta}{\Delta \vdash \langle\!\langle \theta - \rangle\!\rangle : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{CIU}} A}$$

$$\textsc{ciuApp}\ \frac{\Delta \vdash \mathcal{D} : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{CIU}} B \qquad \Delta, B \vdash N : C}{\Delta \vdash \mathtt{let}_B\ \mathcal{D}\ N : \langle\!\langle \Gamma \vdash A \rangle\!\rangle^{\mathrm{CIU}} C}$$

It is worth stressing the differences between ASCs and CIU contexts. ASCs consist of a well-typed and well-scoped hole applied to a sequence of closed values. CIU contexts form a nested sequence of let bindings (culminating in the hole) with a term body.

Definition 4.2 (CIU Approximation)**.** Let CIU approximation be Definition 2.7 with $\mathcal{K} = \mathrm{CIU}$.

Definition 4.3. Define the operation $\blacklozenge : \mathit{CIU} \longrightarrow \mathit{VCC}$, which transforms a closed CIU context into a closed VCC, by induction on the structure of the CIU context:

$$\blacklozenge(\langle\!\langle \theta - \rangle\!\rangle) = \langle\!\langle - \rangle\!\rangle^{\theta}$$

$$\blacklozenge(\mathtt{let}\ \mathcal{P}\ N) = \mathtt{let}\ \blacklozenge(\mathcal{P})\ N$$

Lemma 4.4. *If* $\Gamma \vdash M \lesssim_A^{\mathit{VCC}} N$ *then* $\Gamma \vdash M \lesssim_A^{\mathit{CIU}} N$.

4.1 Frame Stack Properties

In this section, we establish some properties of frame stacks, a well-typed construct satisfying the inference rules in Figure 9. A frame stack is a stack consisting of open terms each containing exactly one free variable. The frame stack type $A \multimap B$ denotes a stack with *argument* type A and return type B. For example, ConsFrmTy says that if we have a frame stack S of type $B \multimap C$, expecting an argument of type B, and a term N of type B with a free variable of type A then we can form the frame stack $S \circ_A N$ of type $A \multimap C$, expecting an argument of type A. The argument to a frame stack mimics that of a hole in a closed VCC.

Evaluation of a frame stack is with respect to a *focussed* term whose type corresponds to the argument type of the frame stack. For readability we mostly suppress typing annotations on frame stack evaluation, just as for big-step evaluation.

Well-Typed Syntax

$$\textsc{NilFrmTy}\ \frac{}{\cdot \vdash \mathit{Id} : A \multimap A} \qquad \textsc{ConsFrmTy}\ \frac{\cdot \vdash S : B \multimap C \qquad A \vdash N : B}{\cdot \vdash S \circ_A N : A \multimap C}$$

Semantics

$$\textsc{nilValue}\ \frac{\cdot \vdash V : A}{\langle \mathit{Id}, V \rangle \downarrow_A V} \qquad \textsc{consValue}\ \frac{\langle S, N[U] \rangle \downarrow_B V}{\langle S \circ_A N, U \rangle \downarrow_B V}$$

$$\textsc{primRed}\ \frac{M \rightsquigarrow M' \qquad \langle S, M' \rangle \downarrow_A V}{\langle S, M \rangle \downarrow_A V} \qquad \textsc{seq}\ \frac{\langle S \circ_A N, M \rangle \downarrow_B V}{\langle S, \mathtt{let}_A\ M\ N \rangle \downarrow_B V}$$

Figure 9. $\lambda_{FG}^{\rightarrow}$ frame stack syntax and semantics

Define the (left) action over a frame stack, @, to be the operation that produces a term given a frame and a closed term to fill its hole:

$$\mathit{Id}@M = M$$

$$(S \circ_A N)@M = S@(\mathtt{let}_A\ M\ N)$$

We may extend the @ action to operate on closed CIU contexts instead of terms, denoted by $- \langle\!\langle @ \rangle\!\rangle -$, replacing the case for ConsFrmTy above with:

$$(S \circ_A N)\langle\!\langle @ \rangle\!\rangle \mathcal{P} = S\langle\!\langle @ \rangle\!\rangle(\mathtt{let}\ \mathcal{P}\ N)$$

The type of the holes in the context are left unchanged by the operation. The following lemma establishes that the action over a frame stack commutes with context instantiation.

Lemma 4.5. $(S\langle\!\langle @ \rangle\!\rangle \mathcal{P})\langle\!\langle M \rangle\!\rangle^{\mathit{CIU}} = S@(\mathcal{P}\langle\!\langle M \rangle\!\rangle^{\mathit{CIU}})$

Proof. By induction on the frame stack S. □

The following lemma relates frame stack and big-step evaluation using the action over a frame stack.

Lemma 4.6. $\langle S, M \rangle \downarrow V \iff S@M \Downarrow V$.

Proof. For $\Longleftarrow$, the proof follows by the properties:

1. $M \Downarrow V \implies \langle \mathit{Id}, M \rangle \downarrow V$
2. $\langle \mathit{Id}, S@M \rangle \downarrow U \implies \langle S, M \rangle \downarrow U$

where (1) is by induction on the derivation of $M \Downarrow V$ and (2) is by induction on S.

For $\Longrightarrow$, the proof follows from a standardisation argument [33]: evaluation can be decomposed into evaluation of a *focussed* term M to value W and evaluation of the surrounding context S filled with W.

3. $S@M \Downarrow V \implies \exists\, W. M \Downarrow W \wedge S@W \Downarrow V$
4. $M \Downarrow W \wedge S@W \Downarrow V \implies S@M \Downarrow V$

Both are proved simultaneously by induction on S. □

We define a relation transformer that lifts a relation on values to a relation between frame stack configurations $\langle -, - \rangle$.

Definition 4.7. For frame stacks $S_1, S_2 : A \multimap B$, closed terms $M_1, M_2 : A$, and binary relation $\mathcal{R}$ on closed values of type B, relation $S_1, M_1 \ [\mathcal{R}]^F \ S_2, M_2$ holds if and only if

$$\forall\, V.\ \langle S_1, M_1\rangle \downarrow V \implies \exists U.\ \langle S_2, M_2\rangle \downarrow U \wedge V\ \mathcal{R}\ U\,.$$

4.2 Applicative Frame Approximation

Having set up frame-stack machinery, we move on to the associated approximation relations. In Sections 3.1 and 3.2 we defined value and term relations simultaneously, using $[\ ]^T$ to pass from values to terms. In a CIU setting, frame stacks now provide a bridge between value relations and term relations. This is most obvious with logical approximations, where all three are defined simultaneously. For applicative approximation, our results need only the term relation.

Definition 4.8 (Applicative Frame Approximation)**.** Term M_1 applicatively frame approximates M_2 at type A if evaluation of M_1 in any appropriately-typed frame stack S approximates the evaluation of M_2 in the same S.

$$M_1\ \langle\lesssim\rangle_A^{trm}\ M_2 \ \triangleq\ \forall S : A \multimap B.\ S, M_1\ [\cong_B]^F\ S, M_2$$

Approximation for values mirrors that of Definition 3.8, building on this term approximation; while two frame stacks are related based on their evaluation at all values. We omit the formal descriptions as we do not need them further.

Lemma 4.9. *If* $\Gamma \vdash M \lesssim_A^{CIU} N$ *then* $\Gamma \vdash M\ \langle\lesssim\rangle_A^{trm}\ N$

Proof. Assume a Γ-closing substitution θ with $\langle S, \theta(M)\rangle \downarrow V$ for some S and V. By Lemma 4.6, $S@\theta(M) \Downarrow V$ holds. In the CIU approximation assumption, set the context $\mathcal{P}$ to be $S\langle\!\langle @\rangle\!\rangle\langle\!\langle \theta - \rangle\!\rangle$. It follows that there exists a value W such that $(S\langle\!\langle @\rangle\!\rangle\langle\!\langle \theta - \rangle\!\rangle)\langle\!\langle N\rangle\!\rangle^{\mathrm{CIU}} \Downarrow W$ and $V \cong W$. The result follows by Lemmas 4.5 and 4.6. □

Lemma 4.10. *Observational approximation implies applicative frame approximation.*

Proof. By Lemmas 2.9, 4.4 and 4.9. □

4.3 Logical Frame Approximation

We now define the frame stack analogue to logical approximation from Section 3.2, using $\top\top$-lifting / biorthogonality for logical relations as demonstrated by Pitts and Stark [30].

Definition 4.11 (Logical Frame Approximation)**.** Logical frame approximation is defined by three mutually recursive relations on pairs of (closed) values, terms and stacks. Define $V_1\ \langle\lessapprox\rangle_A^{val}\ V_2$ for closed values V_1, V_2 inductively by the structure of type A:

$$\textsc{LogFrmGndApx}\ \frac{V_1 \cong_\tau V_2}{V_1\ \langle\lessapprox\rangle_\tau^{val}\ V_2}$$

$$\textsc{LogFrmAbsApx}\ \frac{\forall\, V_1, V_2.\ V_1\ \langle\lessapprox\rangle_A^{val}\ V_2 \implies M_1[V_1]\ \langle\lessapprox\rangle_B^{trm}\ M_2[V_2]}{\lambda_A M_1\ \langle\lessapprox\rangle_{A\to B}^{val}\ \lambda_A M_2}$$

The relation $\langle\lessapprox\rangle^{trm}$ is defined by the $\top\top$-lifting of $\langle\lessapprox\rangle^{val}$ over $\langle\lessapprox\rangle^{stk}$: for closed terms M_1 and M_2, approximation $M_1\ \langle\lessapprox\rangle_A^{trm}\ M_2$ is defined to be

$$\forall S_1, S_2.\ S_1\ \langle\lessapprox\rangle_{A\multimap B}^{stk}\ S_2 \implies S_1, M_1\ [\cong_B]^F\ S_2, M_2$$

where the approximation $S_1\ \langle\lessapprox\rangle_{A\multimap B}^{stk}\ S_2$ for frame stacks S_1 and S_2 is defined as

$$\forall\, V_1, V_2.\ V_1\ \langle\lessapprox\rangle_A^{val}\ V_2 \implies S_1, V_1\ [\cong_B]^F\ S_2, V_2\,.$$

As with applicative frame approximation, $\langle\lessapprox\rangle^{trm}$ uses the $\cong$ relation to relate the final values. In the definition of $\langle\lessapprox\rangle^{stk}$, we implicitly lift the value arguments to terms.

As in Lemmas 3.15, 3.16, and 3.22, logical frame approximation is closed under primitive reduction and expansion, and closed under all VSCs.

Lemma 4.12 (Logical Frame Apx. Lifts)**.** *Logical frame approximation is closed under the compound term formers.*

Proof. The only deviation from the proof of Lemma 3.18 is for `let`, where we extend the approximation of stacks from $S_1\ \langle\lessapprox\rangle_{B\multimap C}^{stk}\ S_2$ to $S_1 \circ_A N\ \langle\lessapprox\rangle_{A\multimap C}^{stk}\ S_2 \circ_A N'$. □

Lemma 4.13. *Logical frame approximation is reflexive:*

1. $\Gamma \vdash V\ \langle\lessapprox\rangle_A^{val}\ V$
2. $\Gamma \vdash M\ \langle\lessapprox\rangle_A^{trm}\ M$
3. $S\ \langle\lessapprox\rangle_A^{stk}\ S$

Proof. For (1) and (2) the proof is similar to Lemma 3.19 using Lemma 4.12. For (3), perform induction on the well-typed derivation of S using (2) for ConsValue. □

Lemma 4.14. *If* $\Gamma \vdash M_1\ \langle\lessapprox\rangle_A\ M_2$ *and* $\Delta \vdash C : \langle\!\langle \Gamma \vdash A\rangle\!\rangle B$ *then*

$$\Delta \vdash C\langle\!\langle M_1\rangle\!\rangle\ \langle\lessapprox\rangle_B^{trm}\ C\langle\!\langle M_2\rangle\!\rangle$$

4.4 Establishing the Triangle

We can now proceed to the following analogue of Lemma 3.23.

Lemma 4.15. *Logical frame approximation implies observational approximation:*

$$\Gamma \vdash M\ \langle\lessapprox\rangle_A^{trm}\ N \implies \Gamma \vdash M \lesssim_A^{VSC} N\,.$$

Proof. By definition, using Lemma 4.14. □

Lemma 4.16. *Applicative and logical frame approximation satisfy the following transitivity property:*

$$\textit{If } M\ \langle\lesssim\rangle_A^{trm}\ N \textit{ and } N\ \langle\lessapprox\rangle_A^{trm}\ P \textit{ then } M\ \langle\lessapprox\rangle_A^{trm}\ P.$$

Proof. By induction on the structure of type A. □

We establish the final implication of the triangle: that applicative frame approximation implies logical frame approximation.

Lemma 4.17.

$$\Gamma \vdash M\ \langle\lesssim\rangle_A^{trm}\ N \implies \Gamma \vdash M\ \langle\lessapprox\rangle_A^{trm}\ N$$

Proof. By Lemmas 4.13 and 4.16. □

Once again, all three notions coincide.

Lemma 4.18 (Frame Stack Triangulation)**.** *Observational, applicative frame and logical frame approximation coincide.*

Proof. Using Lemmas 4.10, 4.17 and 4.15. □

For a simply-typed lambda calculus like $\lambda^{\rightarrow}_{FG}$ it is not in fact essential to consider CIU approximation — we already know from Section 3 that applicative contexts are enough to distinguish terms. Differences, though, arise in any language with features like state, exceptions, or other effects that give program contexts greater discriminating power. In these cases we expect the frame-stack approach to be appropriate: and note that the triangulation proofs are reassuringly similar in shape when moving from Milner's original context lemma to a CIU version. We hope to maintain this in scaling the development to support effectful calculi, in particular those based on algebraic effects and handlers [6, 18].

5 Related Work

Program equivalence has a long history, so an exhaustive account of the literature on the subject is impossible in the space available. We identify brief highlights that have had a direct influence on this work.

Howe developed a general method for proving coincidence of applicative bisimilarity and observational equivalence for a broad class of (untyped) lazy languages [12] and subsequently extended the method to support call-by-value calculi [13]. A key element of the approach is to introduce a relation, called the 'precongruence candidate', which bridges the gap between the applicative and observational notions. The technique has been shown to extend to other languages, including a typed metalanguage with recursive types [11] and extensions of PCF [15, 28]. Howe's method does not reason explicitly about reduction sequences in contrast to others [19, 32]. Instead, the precongruence candidate is sufficient to show that the relation is closed under all contexts, treating contexts abstractly. Our approach is a middle ground in that we reason explicitly about the structure of contexts but do not explicitly analyse reduction sequences.

Since our original submission, we have formalised Howe's construction as (yet another) operator $[\_]^{\mathsf{H}}$ on relations, proving its basic properties (reflexivity, transitivity, substitutivity, and crucially, the analogue of Lemma 3.22). We have further proved that $[\lesssim]^{\mathsf{H}} \subseteq \lessapprox$, making essential use of (an analogue of) Lemma 3.20. Hence, by virtue of our existing (big-step) triangulation, we obtain that $[\lesssim]^{\mathsf{H}}$ is extensionally equivalent to $\lessapprox$, since $R \subseteq [\mathsf{R}]^{\mathsf{H}}$ for R reflexive.

Stark showed a triangle of relations equivalent for the nu-calculus and proved a context lemma for an instantiated hole applied to functions returning a boolean value [32]. However, this special form of context lemma is proved by analysing the process of reduction and the particular forms for closed expressions in the calculus. In contrast, our fine-grained calculus is simple enough for consideration of applicative contexts à la Milner [20] to suffice, and hence the context lemma follows from the triangulation result.

Pitts and Stark prove a context lemma, which includes the CIU theorem, for a functional language with local state [30]. Their use of the triangulation technique inspired this work and our Section 4 essentially isolates their approach, free from considerations of state relations. However, our formal approach gives a more satisfactory account of contexts, their scope and the variable capture involved when instantiating a hole with a term. In particular, there is no requirement for a side-condition in our analogue (Theorem 4.14) of their Theorem 4.9 since by construction every occurrence of a hole (in a VSC) is paired with a well-typed and well-scoped substitution.

Pitts [29] developed a triangle for an ML-like language, but in contrast to ReFS [30], it is based on an extension of Howe's relational approach [11, 15]. The relational approach for representing contexts is favoured in order to mitigate the difficulties involved with concrete contexts and their potential capture of free variables of a term. The folklore belief is that such issues are especially difficult to handle in the setting of machine checked proofs. On the contrary, a careful choice of representation and appropriate use of state-of-the-art technology makes their representation routine. Moreover, the relational approach relies on an inductive characterisation in terms of the language syntax. Some of these rules involve the use of variables from the context. It is not clear how one would formalise such definitions without having to deal with issues of naming and α-convertibility; issues handled automatically by the ACMM framework.

We note that we have in this work used straightforward recursion and structural induction in our definitions and proofs. This is to be expected for a simply-typed lambda-calculus, where all evaluation is terminating. However, previous work on ReFS shows that frame stacks are also sufficient to handle recursive definitions, termination, and unwinding [30, §3]. The key feature is how the structure of frame stacks captures evaluation in fine detail, guiding the corresponding proof structure. We conjecture that this auto-calibration may help postpone or even avoid the need for step-indexing [1, 4] when moving to more elaborate language features.

More generally, we consider that the significant characteristic of all our structural relations, applicative or logical, is not recursion or induction but their closure under inference rules and biorthogonality / $\top\top$-lifting.

Regarding formalisations of context-lemma-like results, the nearest related work we can find is by Ambler and Crole, in Isabelle/HOL [3], subsequently with Momigliano in the Hybrid system [22], then by Momigliano in the Abella theorem prover [21], and by Thibodeau et al. in the Beluga theorem prover [35]. With the exception of Ambler and Crole's work in HOL, also based on a concrete de Bruijn representation, the other authors address higher-order abstract syntax

(HOAS) presentations, together with greater or lesser sophistication in the implementation of the meta-level to account for term contexts and open approximation relations. In each case, the authors use Howe's coinductive method for defining applicative simulation and proving it a pre-congruence, but do not show its coincidence with observational approximation as we do in Lemmas 3.23 and 4.15. Indeed, Thibodeau et al.'s HOAS approach does not support any kind of variable-capturing contexts, while the authors note Beluga requires a non-trivial extension to encode observational approximation using the relational approach [16, 27].

Prior to Thibodeau et al., it seems to have been extremely difficult to judge formalisation work as robust or reusable. Ambler and Crole themselves note how much work went into the machinery of substitution and de Bruijn manipulation; we might conjecture that our use of a dependently-typed metalanguage confers many advantages over HOL formalisations in terms of building-in well-typed invariants to our definitions, while our exploitation of ACMM demonstrates the versatility and abstraction available in a first-order deep embedding combined with shallow HOAS reasoning techniques. Momigliano's suggestive title hints at his reluctance to repeat the experiment. Working in Beluga, Thibodeau et al. seem to have achieved robustness, readability, and (potential) reusability, but at the cost of a much more sophisticated meta-language than our concrete approach in Agda.

Johann et al. study similar results in a general operational setting for a polymorphic calculus extended with algebraic effects [14]. Like us, they parameterise their approximations with respect to a notion of 'basic' preorder for making observations on ground type computations. Their expressive setting provides justification for the parameterisation, permitting different choices of basic preorder to give a semantics with respect to different collections of effects. Their work offers hope that our results will extend to calculi with algebraic effects and handlers.

6 Conclusion

We have drawn out the distinctive proof technique of triangulation utilised in proofs of previous context lemmas. We isolated the method by considering two versions in the setting of a fine-grained call-by-value calculus: the applicative or classical result inspired by Milner; and the CIU-like result required of effectful calculi. Moreover, we formalised our results using state-of-the-art technology for representing well-typed and well-scoped syntax — in particular, the explicit first-order representation of observational contexts.

We have contributed to the exposition of triangulation as a general technique for proving context lemmas; its formalisation provides a robust basis for future work on more exotic calculi. In particular, we see frame stacks as a good model for reasoning about *handler stacks* and the operational semantics of algebraic effects in languages like *Eff* and *Frank* [6, 18].

Acknowledgments

We are grateful to the anonymous referees for their comments and suggestions, and to the Agda developers for their help in ironing out inconsistencies in checking our proofs under versions 2.5.1, 2.5.2 and 2.5.3 of the Agda system. All three authors acknowledge the support of the Laboratory for Foundations of Computer Science (LFCS); the first author is also supported by the EPSRC under doctoral training grant EP/K503034/1.

References

[1] Amal J. Ahmed. 2006. Step-Indexed Syntactic Logical Relations for Recursive and Quantified Types. In *Programming Languages and Systems: Proceedings of the 15th European Symposium on Programming, ESOP 2006 (Lecture Notes in Computer Science)*, Peter Sestoft (Ed.), Vol. 3924. Springer, 69–83. https://doi.org/10.1007/11693024_6

[2] Guillaume Allais, James Chapman, Conor McBride, and James McKinna. 2017. Type-and-scope Safe Programs and Their Proofs. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2017)*. 195–207.

[3] Simon Ambler and Roy L. Crole. 1999. Mechanized Operational Semantics via (Co)Induction. In *TPHOLs'99, Nice, France*. 221–238. https://doi.org/10.1007/3-540-48256-3_15

[4] Andrew W. Appel and David A. McAllester. 2001. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.* 23, 5 (2001), 657–683. https://doi.org/10.1145/504709.504712

[5] Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. 2005. Mechanized Metatheory for the Masses: The PoplMark Challenge. In *Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'05)*. 50–65.

[6] Andrej Bauer and Matija Pretnar. 2013. *CALCO*. Chapter An Effect System for Algebraic Effects and Handlers, 1–16.

[7] Edwin Brady. 2013. Idris, a General-Purpose Dependently Typed Programming Language: Design and implementation. *Journal of Functional Programming* 23, 5 (2013), 552–593. https://doi.org/10.1017/S095679681300018X

[8] N. G. de Bruijn. 1972. Lambda Calculus Notation with Nameless Dummies. In *Indagationes Mathematicae*, Vol. 75. Elsevier, 381–392.

[9] Joëlle Despeyroux, Amy Felty, and André Hirschowitz. 1995. Higher-order abstract syntax in Coq. In *Proceedings of TLCA (LNCS)*, Vol. 902. Springer, 124–138.

[10] Peter Dybjer. 1994. Inductive families. *Formal aspects of computing* 6, 4 (1994), 440–465.

[11] Andrew D. Gordon. 1994. *Functional Programming and Input/Output* (1st ed.). Cambridge University Press, New York, NY, USA.

[12] Douglas J. Howe. 1989. Equality in lazy computation systems. In *Fourth Annual Symposium on Logic in Computer Science*. 198–203.

[13] Douglas J. Howe. 1996. Proving Congruence of Bisimulation in Functional Programming Languages. *Information and Computation* 124, 2 (1996), 103–112.

[14] Patricia Johann, Alex Simpson, and Janis Voigtländer. 2010. A Generic Operational Metatheory for Algebraic Effects. In *LICS '10*. IEEE Computer Society, 209–218.

[15] S. B. Lassen. 1998. Relational Reasoning about Contexts. In *Higher Order Operational Techniques in Semantics*, A. D. Gordon and A. M. Pitts (Eds.). Cambridge University Press, 91–135.

[16] Soren Bogh Lassen. 1998. *Relational Reasoning about Functions and Nondeterminism*. Ph.D. Dissertation. University of Aarhus.

Table 1. Selected notation used in the paper and corresponding Agda definitions

Concept	Notation	Reference	Agda Definition
Simultaneous substitution	$\theta(-)$	Definition 2.1	`*-Val`
Lookup in substitution	θ_k	Definition 2.2	`var`
Successor substitution	succ $-$	Definition 2.2	`succ`
Lift value relation to terms by evaluation	$[-]^T$	Definition 2.4	`_[_]^T_`
Ground equivalence	$\cong$	Definition 2.6	`gnd-eqv`
Variable-capturing contexts	$\langle\!\langle - \vdash - \rangle\!\rangle^{\mathrm{VCC}}$	Figure 6	`VCC⟨⟨_⊢_⟩⟩`
Observational approximation	$\lesssim^{\mathrm{VSC}}$	Definition 2.8	`vsc-apx`
Applicative substituting contexts	$\langle\!\langle - \vdash - \rangle\!\rangle^{\mathrm{ASC}}$	Definition 3.1	`ASC⟨⟨_⊢_⟩⟩`
ASC approximation	$\lesssim^{\mathrm{ASC}}$	Definition 3.2	`asc-apx`
Beta-redexes from substitution	$\mathcal{D}^\theta$	Definition 3.3	`VCC-sub`
Transform ASC to VCC	$\star$	Definition 3.5	`asc-to-vcc`
Applicative approximation	$\lesssim$	Definition 3.8	`app-apx`$_\emptyset$
Open applicative approximation	$\Gamma \vdash M \lesssim_A N$	Definition 3.9	`app-apx`
Logical approximation	$\lessapprox$	Definition 3.12	`log-apx`$_\emptyset$
Open logical approximation	$\Gamma \vdash M \lessapprox_A N$	Definition 3.14	`log-apx`
CIU contexts	$\langle\!\langle - \vdash - \rangle\!\rangle^{\mathrm{CIU}}$	Definition 4.1	`CIU⟨⟨_⊢_⟩⟩`
CIU Approximation	$\lesssim^{\mathrm{CIU}}$	Definition 4.2	`ciu-apx`
Transform CIU Contexts to VCC	$\blacklozenge$	Definition 4.3	`ciu-to-vcc`
Frame stacks relation transformer	$[-]^F$	Definition 4.7	`_[_]^F_`
Applicative Frame Approximation	$\langle\lesssim\rangle$	Definition 4.8	`app-frm-apx`$_\emptyset$
Logical Frame Approximation	$\langle\lessapprox\rangle$	Definition 4.11	`log-frm-apx`$_\emptyset$

[17] Paul Blain Levy. 2004. *Call-By-Push-Value: A Functional/Imperative Synthesis*. Semantics Structures in Computation, Vol. 2. Springer.

[18] Sam Lindley, Conor McBride, and Craig McLaughlin. 2017. Do Be Do Be Do. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*. 500–514.

[19] Ian Mason and Carolyn Talcott. 1991. Equivalence in functional languages with effects. *Journal of Functional Programming* 1, 3 (1991), 287–327.

[20] Robin Milner. 1977. Fully abstract models of typed λ-calculi. *Theoretical Computer Science* 4, 1 (1977), 1–22.

[21] Alberto Momigliano. 2012. A supposedly fun thing I may have to do again: A HOAS encoding of Howe's method. In *Proceedings of LFMTP'12*. ACM. https://doi.org/10.1145/2364406.2364411

[22] Alberto Momigliano, Simon Ambler, and Roy Crole. 2002. A Hybrid Encoding of Howe's Method for Establishing Congruence of Bisimilarity. *ENTCS* 70, 2 (2002), 60–75. https://doi.org/10.1016/S1571-0661(04)80506-1

[23] James Hiram Morris. 1968. *Lambda-Calculus Models of Programming Languages*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[24] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. 2002. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer.

[25] Ulf Norell. 2009. Dependently typed programming in Agda. In *AFP Summer School*. Springer, 230–266.

[26] Benjamin C. Pierce. 2004. *Advanced Topics in Types and Programming Languages*. The MIT Press.

[27] Andrew Pitts. 2011. Howe's Method for Higher-Order Languages. In *Advanced Topics in Bisimulation and Coinduction*, Davide Sangiorgi and Jan Rutten (Eds.). Cambridge University Press, 197–232.

[28] A. M. Pitts. 1997. Operationally-Based Theories of Program Equivalence. In *Semantics and Logics of Computation*, A. Pitts and P. Dybjer (Eds.). Cambridge University Press, 241–298.

[29] A. M. Pitts. 2005. Typed Operational Reasoning. See [26], Chapter 7, 245–289.

[30] A. M. Pitts and I. D. B. Stark. 1998. Operational Reasoning for Functions with Local State. In *Higher Order Operational Techniques in Semantics*, A. D. Gordon and A. M. Pitts (Eds.). Cambridge University Press, 227–273.

[31] Andrei Popescu, Elsa Gunter, and Christopher Osborn. 2010. Strong Normalisation for System F by HOAS on Top of FOAS. In *Proceedings of LICS*. IEEE, 31–40. https://doi.org/10.1109/LICS.2010.48

[32] Ian Stark. 1994. *Names and Higher-Order Functions*. Ph.D. Dissertation. University of Cambridge. https://doi.org/10/cgnm

[33] Masako Takahashi. 1995. Parallel Reductions in λ-Calculus. *Information and Computation* 118, 1 (1995), 120–127.

[34] The Coq Development Team. 1999–2017. *The Coq Proof Assistant Reference Manual*. https://coq.inria.fr/refman/

[35] David Thibodeau, Alberto Momigliano, and Brigitte Pientka. 2017. A Case-Study in programming Coinductive Proofs: Howe's Method. (2017). Talk presented at TTT Workshop, POPL, Paris.