A One-Pass CPS Transform with Simulation on the Nose

Pascal Y. Lasnier[0009−0009−9371−3467], Jeremy Yallop[0009−0002−1650−6340]1, and Magnus O. Myreen[0000−0002−9504−4107]2,3

[1] University of Cambridge
[2] Chalmers University of Technology
[3] University of Gothenburg

Abstract. Danvy & Nielsen's one-pass CPS transform has a straightforward definition, but clashes between the names of variables it introduces make it difficult to mechanically prove correct. Existing mechanical proofs either side-step the issue by using nameless representations, or rely on tedious α-equivalence relations between target terms. This paper presents a new formulation of the transform using evaluation contexts that allows deterministic introduction of fresh names, eliminating the need to work up to α-equivalence. We use our formulation to present a new and straightforward simulation proof of the correctness of the one-pass CPS transform, which we have mechanised in the HOL4 theorem prover.

1 Introduction

Continuation-passing style (CPS) transforms have many useful properties: they make evaluation order explicit, turn all calls into tail calls, name intermediate computations, and support simulation of non-standard control flow. These properties make the CPS transform a key technique in many applications, from defining control operator semantics [7] to intermediate representation in functional languages compilers [12] and languages with first-class continuations [2,13,22].

The classic CPS transform developed by Plotkin [18] introduces many *administrative redexes*, i.e. function applications which are trivial to evaluate and do not correspond to source program reductions. For example, the program fragment $x_1\, x_2\, (x_3\, x_4\, x_5)$ CPS-transforms into the following:

$$\begin{array}{l}\lambda k.(\lambda k.(\lambda k.k\, x_1)\, (\lambda m.(\lambda k.k\, x_2)\, (\lambda n.m\, n\, k)))\\ \quad (\lambda m.(\lambda k.(\lambda k.(\lambda k.k\, x_3)\, (\lambda m.(\lambda k.k\, x_4)\, (\lambda n.m\, n\, k)))\\ \qquad (\lambda m.(\lambda k.k\, x_5)\, (\lambda n.m\, n\, k)))\\ \quad (\lambda n.m\, n\, k))\end{array}$$

Reducing applications of the form $(\lambda k.k\, x)(\lambda y.e)$, to $[x/y]e$ (i.e. e with x substituted for y) straightforwardly produces the following reduced term:

$$\lambda k.(\lambda k.x_1\, x_2\, k)\, (\lambda m.(\lambda k.(\lambda k.x_3\, x_4\, k)\, (\lambda m.m\, x_5\, k))\, (\lambda n.m\, n\, k))$$

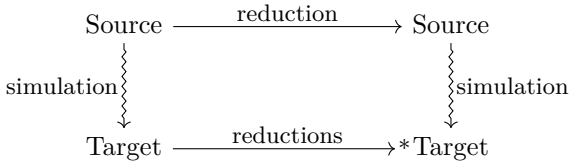

Fig. 1. Simulation as a commutative diagram.

There are three further administrative redexes that can be eliminated in this program. However, reducing these redexes requires renaming one of the m variables to avoid clashes. After reduction and renaming, the result is as follows:

$$\lambda k.x_1\, x_2\, (\lambda m.x_3\, x_4\, (\lambda m'.m'\, x_5\, (\lambda n.m\, n\, k)))$$

There are several optimised CPS transforms [4,9,21] that eliminate these administrative redexes and produce CPS programs such as the final version above. The state-of-the-art for the call-by-value lambda calculus is the first-order one-pass CPS transform by Danvy & Nielsen [9].

Proof of correctness of the CPS transform, which is typically formulated as a simulation proof, involves establishing a relation between source programs and CPS programs which the CPS transform satisfies, and proving that the relation is closed under source and target reductions. Fig. 1 portrays this statement as a commutative diagram. Simulation is simple to prove for the classic CPS transform, but the name clashes seen in the example above introduce challenges in the proofs for the optimised variants.

In their hand-written correctness proof, Danvy & Nielsen simply assert that introduced variables are unique [9]. In contrast, in a typical mechanised proof, binders require explicit reasoning. A naive fresh binder naming scheme will result in CPS programs that do not strictly reduce in the simulation diagram, but instead reduce to α-equivalent programs [15], complicating the simulation proof with a new notion of equivalence.

Paraskevopoulou & Grover [16] summarise the accumulated literature on the variable binder issue for mechanised proofs of the optimised CPS transform. Proofs which represent variable binders with explicit names require establishing some notion of equivalence over programs in CPS, with Minamide & Okuma [15] using an α-equivalence relation, and Paraskevopoulou & Grover [16] developing a more general logical relation over CPS terms. These approaches complicate verified implementation of the CPS transform, since constructing an equivalence relation and proving it for a target language involves significant additional verification work, especially if the target language is complex. Other developments use nameless variable binding system such as de Bruijn indices [10,19] or higher-order abstract syntax [3,23].

This paper contributes a new approach to the problem of name generation in an optimised CPS transform, with a mechanised simulation proof that both uses

named variable binders *and* completes the simulation diagram without requiring an additional notion of equivalence. Our approach to the problem consists of three components:

Evaluation-context-based optimised CPS transform. We refactor Danvy & Nielsen's optimised CPS transform [9] using evaluation contexts, following Sabry & Felleisen's non-compositional optimised transform [21], but preserving compositionality. Reframing the transform in this way highlights a critical feature of variable name introduction: name clashes are *local to the lexical scope* of source language terms. This observation is an instance of a general principle that freshness should not be treated as a global property, but determined relative to some local context [17]. We apply the principle to the challenge at hand, using it to design a name generation strategy that is deterministically dependent *only* on the evaluation context of the local scope.

Modified CEK-machine with lexical scopes. Using the insight that variable binder names may be allocated locally to the lexical scope of the source language, we modify the CEK-machine small-step semantic definition of the call-by-value lambda calculus by rephrasing evaluation contexts as a nested sequence of scopes, where each scope has a distinct, *local* environment and evaluation context. At the top level of the semantic state, the new abstract machine operates on a sequence of scopes, with the innermost evaluation context containing the control string in its hole.

Simulation. We construct a simulation relation between semantic states of the modified CEK-machine for the call-by-value lambda calculus to CPS terms through our evaluation-context-based optimised CPS transform, using the evaluation context of the local scope from the semantic state directly in the transform. Because variable binder names are deterministic from the local lexical scope, the exact variable name can be deduced directly from the simulation relation, and simulation can be easily proven without requiring a notion of equivalence.

We have mechanised our proof in the HOL4 theorem prover, and include the development as supplementary material.

2 Call-by-value lambda calculus CEK-machine

$$
\begin{array}{ll}
e ::= t \mid s & \in \text{Exp} \\
t ::= x \mid \lambda x.e & \in \text{Triv} \\
s ::= e_1\, e_2 & \in \text{Comp} \\
x & \in \text{Ide}
\end{array}
$$

Fig. 2. Lambda calculus grammar

We first present the lambda calculus which we use in our mechanised proof (Fig. 2). Terms e are either trivial terms t, which include variables x and abstractions $\lambda x.e$, or serious terms s, which are applications $e_1\, e_2$.

$$\begin{array}{lll} v & ::= \Lambda x.\{\rho\}e & \in \mathrm{Val} \\ \rho & : \mathrm{Ide} \rightharpoonup \mathrm{Val} & \\ & & \\ E & ::= [] \mid E\,\{\rho\}e \mid v\,E & \in \mathrm{Cont} \\ c & ::= \{\rho\}e \mid v & \in \mathrm{Ctrl} \end{array}$$

$$\begin{array}{c} \mathrm{Cont[Ctrl]} \longrightarrow \mathrm{Cont[Ctrl]} \\ \\ E[\{\rho\}x] \longrightarrow E[\rho(x)] \\ E[\{\rho\}\lambda x.e] \longrightarrow E[\Lambda x.\{\rho\}e] \\ E[\{\rho\}e_1\, e_2] \longrightarrow E[[\{\rho\}e_1]\,\{\rho\}e_2] \\ \\ E[[v]\,\{\rho\}e] \longrightarrow E[v\,[\{\rho\}e]] \\ E[(\Lambda x.\{\rho\}e)\,[v]] \longrightarrow E[\{\rho[x \mapsto v]\}e] \end{array}$$

Fig. 3. CEK-machine

Fig. 3 shows the call-by-value semantics for the calculus as a small-step CEK-machine semantics. The CEK-machine operates on control strings which may be either terms with an attached environment, which we denote similarly to substitution $\{\rho\}e$, or values v, which may only be closures which capture an environment $\Lambda x.\{\rho\}e$. Environments ρ simply map variables to values.

The semantic state of the CEK-machine includes an evaluation context E, which is a nested sequence of stack frames. There are two types of stack frame: fn frames $E\,\{\rho\}e$ enclose an unevaluated function term in an application, where the evaluation context holds the argument term and associated environment $\{\rho\}e$, and arg frames $v\,E$ enclose an unevaluated argument term, where the evaluation context holds the previously evaluated function value v.

The complete definition of the CEK-machine is a relation between semantic states consisting of an evaluation context E with its hole filled by a control string c. We denote the type of a filled evaluation context by figuratively filling the hole of the Cont type, i.e. the type Cont[Ctrl] is that of evaluation contexts with their holes filled by control strings $E[c]$.

3 Revisiting the optimised CPS transform

3.1 Classic transform

Fig. 4 shows Plotkin's classic CPS transform [18]. To distinguish the source and target domains, we write target terms in **bold**. The proof of Plotkin's classic simulation theorem is simple and well-understood, and has been mechanised by Minamide & Okuma [15] and Dargaye & Leroy [10].

A simulation proof technique which has proven quite effective is to relate CEK-machine source semantic states to CPS terms in the target domain. This relation is not arbitrary, but rather arises from the nature of small-step semantics

$$
\begin{aligned}
[\![x]\!] &= \boldsymbol{\lambda k.k\,x} \\
[\![\lambda x.e]\!] &= \boldsymbol{\lambda k.k\,(\lambda x.}[\![e]\!]\boldsymbol{)} \\
[\![e_1\,e_2]\!] &= \boldsymbol{\lambda k.}[\![e_1]\!]\,\boldsymbol{(\lambda m.}[\![e_2]\!]\,\boldsymbol{(\lambda n.m\,n\,k))}
\end{aligned}
$$

Fig. 4. Plotkin's classic CPS transform

as a *defunctionalisation* of the continuation-passing style [1,5,6,8,20], and it underscores the nature of CPS as a *simulation* of control structures (i.e. evaluation contexts in a small-step semantics) by terms in a target domain. This relation generalises to larger source languages: Lasnier, Yallop, & Myreen [14] use the relation to prove simulation for Scheme, including first-class continuations.

3.2 One-pass transform

Fig. 5 shows Danvy & Nielsen's first-order, compositional, one-pass formulation of the optimised CPS transform [9], with an explicit naming scheme we motivate in the following sections. The subscript $\boldsymbol{K}$ in the expression transform $\mathscr{E}$ corresponds to the top-level continuation of a given scope: it may be only either the identity continuation $\mathbf{I}$ or a continuation variable $\boldsymbol{k}$. In contrast, in the serious transform $\mathscr{S}$, the continuation argument $\boldsymbol{K}$ may be an arbitrary continuation. The complete transform of a program p is $\mathscr{E}_{\mathbf{I}}[\![p]\!]$.

$$
\begin{aligned}
\mathscr{E}_{\mathbf{I}}[\![t]\!] &= \mathscr{T}[\![t]\!] \\
\mathscr{E}_{\boldsymbol{k}}[\![t]\!] &= \boldsymbol{k}\,\mathscr{T}[\![t]\!] \\
\mathscr{E}_{\boldsymbol{K}}[\![s]\!] &= \mathscr{S}[\![s]\!]_0\,\boldsymbol{K}
\end{aligned}
\qquad
\begin{aligned}
\mathscr{T}[\![x]\!] &= \boldsymbol{x} \\
\mathscr{T}[\![\lambda x.e]\!] &= \boldsymbol{\lambda xk.}\mathscr{E}_{\boldsymbol{k}}[\![e]\!]
\end{aligned}
$$

$$
\begin{aligned}
\mathscr{S}[\![t_1\,t_2]\!]_i\,\boldsymbol{K} &= \mathscr{T}[\![t_1]\!]\,\mathscr{T}[\![t_2]\!]\,\boldsymbol{K} \\
\mathscr{S}[\![t_1\,s_2]\!]_i\,\boldsymbol{K} &= \mathscr{S}[\![s_2]\!]_i\,\boldsymbol{(\lambda n.}\mathscr{T}[\![t_1]\!]\,\boldsymbol{n\,K)} \\
\mathscr{S}[\![s_1\,t_2]\!]_i\,\boldsymbol{K} &= \mathscr{S}[\![s_1]\!]_i\,\boldsymbol{(\lambda m_i.m_i}\,\mathscr{T}[\![t_2]\!]\,\boldsymbol{K)} \\
\mathscr{S}[\![s_1\,s_2]\!]_i\,\boldsymbol{K} &= \mathscr{S}[\![s_1]\!]_i\,\boldsymbol{(\lambda m_i.}\mathscr{S}[\![s_2]\!]_{i+1}\,\boldsymbol{(\lambda n.m_i\,n\,K))}
\end{aligned}
$$

Fig. 5. Danvy & Nielsen's one-pass CPS transform

3.3 Clash of variable names and α-equivalence

Minamide & Okuma [15] observe that it is not possible to implement Danvy & Nielsen's optimised transform without variable renaming, since it may introduce multiple conflicting instances of the variable $\boldsymbol{m}$. They highlight this property with the example transform for $x_1\,x_2\,(x_3\,x_4\,x_5)$, in which one of the $\boldsymbol{m}$ variables must be renamed:

$$\boldsymbol{x_1\,x_2\,(\lambda m.x_3\,x_4\,(\lambda m'.m'\,x_5\,(\lambda n.m\,n\,k)))}$$

Implementations of the optimised transform must consequently generate fresh variable names. Minamide & Okuma [15] generate names via a counter i attached to the transform similarly to Fig. 5, and Paraskevopoulou & Grover [16] allocate globally fresh names using a state monad. However, Danvy & Nielsen's simulation proof is complicated by these fresh variable name generation schemes, which only fit the simulation diagram up to α-equivalence. Consider Danvy & Nielsen's simulation theorem, which states that if a serious source term reduces to a new term, the CPS transform of the serious term must also reduce to the CPS transform of the new term. Using our definition from Fig. 5 and splitting cases for either a trivial or serious reduced source term, the theorem is formally stated as follows:

$$s \longrightarrow t \implies \mathscr{S}[\![s]\!]_i\, \boldsymbol{K} \longrightarrow^+ \boldsymbol{K}\, \mathscr{T}[\![t]\!]$$
$$s \longrightarrow s' \implies \mathscr{S}[\![s]\!]_i\, \boldsymbol{K} \longrightarrow^+ \mathscr{S}[\![s']\!]_i\, \boldsymbol{K}$$

Decomposing s into an evaluation context and a redex $E[t_1\, t_2]$, the proof [9] follows by induction over E. This proof deals with the lambda calculus using term rewriting with substitutions rather than an abstract machine with an environment, so the evaluation context does not contain environments ρ.

Let us consider the case for an fn stack frame $E\, e$ where the argument expression is also a serious term, so that our source term may be written as $s_1\, s_2$. The CPS transform for this term is $\mathscr{S}[\![s_1]\!]_i\, (\boldsymbol{\lambda m_i.}\, \mathscr{S}[\![s_2]\!]_{i+1} (\boldsymbol{\lambda n.\, m_i\, n\, K}))$.

If s_1 reduces to a trivial term $s_1 \longrightarrow t$, we may apply our induction hypothesis for the reduction to a trivial term, so our CPS term reduces to $(\boldsymbol{\lambda m_i.}\, \mathscr{S}[\![s_2]\!]_{i+1} (\boldsymbol{\lambda n.\, m_i\, n\, K}))\, \mathscr{T}[\![t]\!]$, then further to $\mathscr{S}[\![s_2]\!]_{i+1} (\boldsymbol{\lambda n.}\, \mathscr{T}[\![t]\!]\, \boldsymbol{n\, K})$. This CPS term, though indeed the CPS transform of the reduced source term $t\, s_2$, does not satisfy the simulation theorem, because the fresh variable name counter i has been incremented.

It would be incorrect to increment the counter i in the starting simulation theorem as a way to compensate for this mismatch, because applying the induction hypothesis to the $s_1\, s_2$ case where s_1 reduces to another serious term $s_1 \longrightarrow s_1'$ results in the reduced CPS term $\mathscr{S}[\![s_1']\!]_{i+1}\, (\boldsymbol{\lambda m_i.}\, \mathscr{S}[\![s_2]\!]_{i+1} (\boldsymbol{\lambda n.\, m_i\, n\, K}))$, which is strictly not the CPS transform of $s_1'\, s_2$ and breaks simulation.

It is therefore required, when proving simulation with explicit binder names, to establish α-equivalence over the CPS transform for incrementing fresh variable counters $\mathscr{S}[\![s]\!]_i =_\alpha \mathscr{S}[\![s]\!]_{i+1}$ as described by Minamide & Okuma [15].

3.4 Optimised CPS transform as static reduction

The core issue with existing implementations of the optimised transform is that fresh variable names of an arbitrary program fragment in simulation cannot be determined by the simulation relation from the fragment alone, instead depending on a floating variable counter i or some global fresh variable allocator. In order to eliminate the need for an equivalence relation, we seek a transform with a variable name generation method which is deterministic, whilst still avoiding clashes.

With this aim in mind, we reformulate the optimised transform to more closely correspond to the source semantics, with the goal of facilitating a suitable

simulation relation. Our starting point is the evaluation-context-based optimised transform presented by Sabry & Felleisen [21]. Drawing on the observation that the optimised transform reduces administrative redexes at the point of the transform, we construct the optimised transform as a kind of partial evaluator based on evaluation contexts like the CEK-machine source semantics, which is given in Fig. 6. Whereas evaluation contexts in the source semantics carry values v for arg frames, in the transform evaluation contexts carry trivial terms $\boldsymbol{t}$ in the target domain. We highlight this distinction by denoting evaluation contexts in the transform with a star $E^\star$.

$$\begin{aligned}
\mathscr{E}_{\boldsymbol{K}}[\![E^\star[x]]\!] &= \mathscr{C}_{\boldsymbol{K}}[\![E^\star]\!]\,\boldsymbol{x} \\
\mathscr{E}_{\boldsymbol{K}}[\![E^\star[\lambda x.e]]\!] &= \mathscr{C}_{\boldsymbol{K}}[\![E^\star]\!]\,\boldsymbol{\lambda x k.}\mathscr{E}_{\boldsymbol{k}}[\![e]\!] \\
\mathscr{E}_{\boldsymbol{K}}[\![E^\star[e_1\ e_2]]\!] &= \mathscr{E}_{\boldsymbol{K}}[\![E^\star[[e_1]\ e_2]]\!]
\end{aligned}$$

$$\begin{aligned}
\mathscr{C}_{\mathbf{I}}[\![[]]\!]\,\boldsymbol{t} &= \boldsymbol{t} \\
\mathscr{C}_{\boldsymbol{k}}[\![[]]\!]\,\boldsymbol{t} &= \boldsymbol{k}\,\boldsymbol{t} \qquad\qquad \mathscr{K}_{\boldsymbol{K}}[\![[]]\!] = \boldsymbol{K} \\
\mathscr{C}_{\boldsymbol{K}}[\![E^\star[[]\ e]]\!]\,\boldsymbol{t} &= \mathscr{E}_{\boldsymbol{K}}[\![E^\star[\boldsymbol{t}\ [e]]]\!] \quad \mathscr{K}_{\boldsymbol{K}}[\![E^\star[[]\ e]]\!] = \boldsymbol{\lambda m_i.}\mathscr{E}_{\boldsymbol{K}}[\![E^\star[\boldsymbol{m_i}\ [e]]]\!] \quad i = \mathsf{fresh}\ E^\star \\
\mathscr{C}_{\boldsymbol{K}}[\![E^\star[\boldsymbol{t_1}\ []]]\!]\,\boldsymbol{t_2} &= \boldsymbol{t_1}\,\boldsymbol{t_2}\,\mathscr{K}_{\boldsymbol{K}}[\![E^\star]\!] \quad \mathscr{K}_{\boldsymbol{K}}[\![E^\star[\boldsymbol{t}\ []]]\!] = \boldsymbol{\lambda n.t\,n}\,\mathscr{K}_{\boldsymbol{K}}[\![E^\star]\!]
\end{aligned}$$

Fig. 6. Our evaluation-context-based optimised CPS transform

In viewing the optimised CPS transform as partial evaluation, we conceptually split continuation applications into static and dynamic variants. In a static application $\mathscr{C}_{\boldsymbol{K}}[\![E^\star]\!]\ \boldsymbol{t}$, the argument is a trivial term in the target domain $\boldsymbol{t}$ rather than a value, and is directly substituted into the continuation bodies at the point of the transform. In a dynamic application $\boldsymbol{k}\,\boldsymbol{t}$, $\boldsymbol{k}$ maps to a closure evaluated from $\mathscr{K}_{\boldsymbol{K}}[\![E^\star]\!]$. The definitions of both these continuation transforms derive from the continuation steps of the CEK-machine from Fig. 3, i.e. steps from a value control string with a particular evaluation context.

The evaluation contexts used to form these static and dynamic continuations are grown from the shape of the program by the expression transform $\mathscr{E}$, which operates on an evaluation context filled with an expression, and whose definition is derived from the expression steps of the CEK-machine from Fig. 3.

One key distinction between our $\mathscr{E}$ transform and the transform by Sabry & Felleisen [21] is that $\mathscr{E}$ explicitly recurses into the evaluation context $E^\star$ of an expression, while Sabry & Felleisen's transform does not. Sabry & Felleisen instead implicitly arrive at an evaluation context via unique decomposition [24], the same notion relating CEK-machines and term-rewriting systems [11], and in effect squash our $\mathscr{E}$ and $\mathscr{C}$ transforms into a single $\mathscr{C}$ transform. A second distinction is that $\mathscr{E}$ is compositional, preserving the adjustments Danvy & Nielsen [9] made to the transform.

Benefits of the new transform. As Section 5 shows, constructing the transform using evaluation contexts benefits the simulation proof by allowing us to directly relate the source language semantic evaluation state, including its evaluation context, to a CPS-transformed expression in the target language. We believe that this same construction can be applied to more complex languages with small-step semantics that use evaluation contexts, such as Scheme.

Constructing the transform as we have done also more clearly highlights the name clash issue encountered by Minamide & Okuma [15]. Note the continuation closure definition for an fn continuation $\mathscr{K}_{\boldsymbol{K}}[\![E^\star[[]\,e]]\!]$; this closure introduces a variable $\boldsymbol{m}$ containing the resultant value of application function term to fill the hole in the evaluation context $E^\star[[]\,e]$. However, this variable is not immediately dispatched, and is instead stashed into the new evaluation context from which the next term e is transformed. Consequently, there can be multiple variables $\boldsymbol{m}$ present in the evaluation context; for example the transform of the $x_1\,x_2\,(x_3\,x_4\,x_5)$ expands as follows:

$$\begin{aligned}\mathscr{E}_{\boldsymbol{k}}[\![x_1\,x_2\,(x_3\,x_4\,x_5)]\!] &= \boldsymbol{x_1}\,\boldsymbol{x_2}\,(\boldsymbol{\lambda m.}\mathscr{E}_{\boldsymbol{k}}[\![\boldsymbol{m}\,[x_3\,x_4\,x_5]]\!]) \\ &= \boldsymbol{x_1}\,\boldsymbol{x_2}\,(\boldsymbol{\lambda m.x_3}\,\boldsymbol{x_4}\,(\boldsymbol{\lambda m.}\mathscr{E}_{\boldsymbol{k}}[\![\boldsymbol{m}\,[\boldsymbol{m}\,[x_5]]]\!]))\end{aligned}$$

The result is multiple $\boldsymbol{m}$ variables present in the same evaluation context, which would clash in the result of the transform if not appropriately renamed.

As Minamide & Okuma state, this clash is not clear from the definition of the optimised CPS transform presented by Danvy & Nielsen; our transform makes it clear by how the introduced variables interact with continuations. This clarity is important, as it enables us to thoughtfully design fresh variable name generation which simplifies the simulation proof.

The transform of Fig. 6 introduces new variables in three places. The first, the lambda abstraction $\lambda x.e$ case for $\mathscr{E}$, introduces $\boldsymbol{k}$, which never clashes because the transform only ever considers at most one $\boldsymbol{k}$ at a time, in the subscript of the transform functions. The second, the arg continuation $E^\star[\boldsymbol{t}\,[]]$ case for $\mathscr{K}$, introduces $\boldsymbol{n}$, which never clashes because it is immediately dispatched after its introduction, rather than stashed in the transform parameters for later use. The third, the fn continuation $E^\star[[]\,e]$ case for $\mathscr{K}$, introduces $\boldsymbol{m}$, which is stashed in the evaluation context $E^\star[\boldsymbol{m}\,[]]$ passed to the next $\mathscr{E}$ transform, and hence may only clash with other $\boldsymbol{m}$ variables present in that evaluation context. What is important is that, when transforming a lambda abstraction $\lambda x.e$, we transform its body without the surrounding evaluation context of the abstraction. Intuitively, this means that $\boldsymbol{m}$ variables may only clash within a particular lexical scope. Consequently, variable naming need only depend on the evaluation context of the current scope, and the names of these introduced variables may also be determined uniquely from a program fragment in simulation. We will use this fact to build a simulation relation that can determine the name of introduced variables without depending on an external variable counter or allocator.

The function fresh on the evaluation context $E^\star$ allocates names that are fresh for the context, the simplest implementation being to count the number of existing $\boldsymbol{m}$ variables in the evaluation context $E^\star$. Including fresh variable

naming, the transform of $x_1\, x_2\, (x_3\, x_4\, x_5)$ becomes:

$$\boldsymbol{x_1\, x_2\, (\lambda m_0.x_3\, x_4\, (\lambda m_1.m_1\, x_5\, (\lambda n.m_0\, n\, k)))}$$

With this particular implementation of fresh, our transform (Fig. 6) is the same as Danvy & Nielsen's with an equivalent naming scheme (Fig. 5), as seen by considering the expression transform on an expression in an empty evaluation context $\mathscr{E}_{\boldsymbol{K}}[\![e]\!]$. We prove this property by induction over terms through the mutually recursive definitions of Danvy & Nielsen's transform functions $\mathscr{E}$ and $\mathscr{S}$, with an additional induction lemma establishing that our transform for an application expression with an arbitrary evaluation context $E^\star$ is the same as Danvy & Nielsen's serious term $\mathscr{S}$ transform with a continuation generated by our $\mathscr{K}$ transform, and an appropriate name counter index:

$$\mathscr{E}_{\boldsymbol{K}}[\![E^\star[e_1\, e_2]]\!] = \mathscr{S}[\![e_1\, e_2]\!]_i\, \mathscr{K}_{\boldsymbol{K}}[\![E^\star]\!] \quad i = \mathsf{fresh}\ E^\star$$

We have mechanised this proof alongside our simulation proof.

4 Adjusting the semantics

Having analysed the nature of introduced variable name clashes in the optimised transform, we now turn our attention back to the semantics which inspired the analysis. In order to enable a suitable simulation relation from the side of the source semantics, we divide the evaluation contexts of the source semantics abstract machine into *scopes*, where environments are attached to particular scopes rather than stack frames in an evaluation context.

This division reflects the structure of evaluation contexts in the optimised CPS transform, where each transform function operates on a scoped evaluation context $E^\star$ and an underlying continuation $\boldsymbol{K}$ (which may be either the identity continuation $\mathbf{I}$ or a continuation variable $\boldsymbol{k}$). New scopes are introduced into CPS by the transform for a lambda abstraction $\mathscr{E}_{\boldsymbol{K}}[\![E^\star[\lambda x.e]]\!] = \mathscr{C}_{\boldsymbol{K}}[\![E^\star]\!]\, \boldsymbol{\lambda x k.}\mathscr{E}_{\boldsymbol{k}}[\![e]\!]$, where the expression transform is applied to the lambda body e with an empty evaluation context, but with an underlying continuation captured by variable $\boldsymbol{k}$ which will eventually map to some continuation closure evaluated from $\mathscr{K}$.

We hence may formulate evaluation contexts in the source semantics as a nested sequence of scopes, where each scope has a distinct environment ρ and evaluation context $E^\star$. At the top level of the semantic state, the abstract machine operates on a sequence of scopes, with the innermost evaluation context containing the control string in its hole $O(\{\rho\}E^\star[c^\star])$. We formalise this notion of scopes and define the adjusted abstract machine semantics in Fig. 7. Each step in the semantics corresponds to a step in the original CEK-machine with the scopes flattened into one evaluation context, except for CBV-CONT-ID which is a CEK-machine no-op.

We equip our new, environment-less evaluation contexts $E^\star$ with a parameter α, which corresponds to the class of terms that may reside in an arg stack frame $a\, E^\star$, i.e. α ranges over the term sets Val, Exp, etc. This parameterised

$$
\begin{array}{lll}
E^\star_\alpha & ::= []_\alpha \mid E^\star_\alpha\, e \mid a\, E^\star_\alpha & \in \mathrm{Cont}^\star_\alpha, a \in \alpha \\
O & ::= () \mid \{\rho\} E^\star_{\mathrm{Val}}\, O & \in \mathrm{Scope} \\
\\
p & ::= \{\rho\} E^\star_{\mathrm{Val}} & \in \mathrm{EnvCont} \\
c^\star & ::= e \mid v & \in \mathrm{Ctrl}^\star
\end{array}
$$

$$
\mathrm{Scope}(\mathrm{EnvCont}[\mathrm{Ctrl}^\star]) \longrightarrow^\star \mathrm{Scope}(\mathrm{EnvCont}[\mathrm{Ctrl}^\star])
$$

$$
\begin{array}{rll}
O(\{\rho\}E^\star[x]) \longrightarrow^\star O(\{\rho\}E^\star[\rho(x)]) & & (\textsc{cbv-exp-var}) \\
O(\{\rho\}E^\star[\lambda x.e]) \longrightarrow^\star O(\{\rho\}E^\star[\Lambda x.\{\rho\}e]) & & (\textsc{cbv-exp-lam}) \\
O(\{\rho\}E^\star[e_1\, e_2]) \longrightarrow^\star O(\{\rho\}E^\star[[e_1]\, e_2]) & & (\textsc{cbv-exp-app}) \\
\\
O(\{\rho\}E^\star[[v]\, e]) \longrightarrow^\star O(\{\rho\}E^\star[v\, [e]]) & & (\textsc{cbv-cont-fn}) \\
O(\{\rho\}E^\star[\Lambda x.\{\rho'\}e\, [v]]) \longrightarrow^\star O(\{\rho\}E^\star(\{\rho'[x \mapsto v]\}[e])) & & (\textsc{cbv-cont-arg}) \\
\\
O(\{\rho'\}E^\star(\{\rho\}[v])) \longrightarrow^\star O(\{\rho'\}E^\star[v]) & & (\textsc{cbv-cont-id})
\end{array}
$$

Fig. 7. Our adjusted semantics, defined on sequences of scopes

evaluation context allows us to share the same evaluation context construction with the evaluation-context-based optimised transform, by instantiating it with **Triv**, the set of trivial terms in the target domain, rather than Val, the set of simple values used in the new source semantics. We may hence preserve our optimised transform definition from Fig. 6 and assign the transform functions types based on Cont, with types representing terms from the target domain in **bold** to differentiate from source terms:

$$
\begin{array}{ll}
\mathscr{E}_{\boldsymbol{K}} : \mathrm{Cont}^\star_{\mathbf{Triv}}[\mathrm{Exp}] & \to \mathbf{Exp} \\
\mathscr{C}_{\boldsymbol{K}} : \mathrm{Cont}^\star_{\mathbf{Triv}} \times \mathbf{Triv} & \to \mathbf{Exp} \\
\mathscr{K}_{\boldsymbol{K}} : \mathrm{Cont}^\star_{\mathbf{Triv}} & \to \mathbf{Triv}
\end{array}
$$

We will omit α for the evaluation context $E^\star$ when it is clear from the context.

5 Simulation relation

Using our adjusted semantics, we construct a simulation relation between semantic abstract machine states and CPS terms built from the optimised transform. The relation is defined inductively over components of the semantic state.

Values and trivial terms. The aspect of the optimised transform that enables it to eliminate administrative redexes is that it directly substitutes trivial terms into continuation bodies, which corresponds to statically applying a continuation

to a trivial term at the point of transform through $\mathscr{C}_{\boldsymbol{K}}[\![E]\!]\,\boldsymbol{t}$. However, this static substitution amounts to deferring of evaluation of these trivial terms until they are used in a serious computation, whereas the corresponding reductions in the source semantics involve the value evaluated from these terms. We must therefore relate values from the source semantics to corresponding trivial terms in CPS to prove simulation.

$$\frac{\rho \sim \boldsymbol{\rho}}{\Lambda x.\{\rho\}e \sim \boldsymbol{\Lambda x.\{\rho\}\lambda k.}\mathscr{E}_{\boldsymbol{k}}[\![e]\!]}\ \textsc{cps-val} \qquad \frac{\forall x \in \mathrm{dom}(\rho).\rho(x) \sim \boldsymbol{\rho}(\boldsymbol{x})}{\rho \sim \boldsymbol{\rho}}\ \textsc{cps-env}$$

$$\frac{v \sim \boldsymbol{\rho}(\boldsymbol{x})}{v \rightsquigarrow \boldsymbol{\{\rho\}x}}\ \textsc{sim-triv-var} \qquad \frac{\rho \sim \boldsymbol{\rho}}{\Lambda x.\{\rho\}e \rightsquigarrow \boldsymbol{\{\rho\}\lambda x k.}\mathscr{E}_{\boldsymbol{k}}[\![e]\!]}\ \textsc{sim-triv-lam}$$

Fig. 8. Value relations.

Fig. 8 defines our simulation relations for values from the source semantics. The $\sim$ relation, which we call *CPS equivalence*, relates equivalent semantic constructs between the source and target semantics. The CPS-VAL rule relates closures to CPS closures, ensuring that their respective captured environments are also related. The corresponding environment relation rule CPS-ENV requires all values mapped to by variables in the source semantics to be related to an equivalently mapped value in the target semantics.

The simulation relation $\rightsquigarrow$, which may be read as 'simulated by', relates source semantics constructs to the target CPS terms that simulate those constructs. For values, the simulation relation is the relation described above between source semantic values and target semantic trivial terms. Whereas the CPS equivalence relation $\sim$ on values is primarily used to ensure variables appropriately map to CPS values that have already been evaluated and bound, the simulation relation $\rightsquigarrow$ is used for values in the control string. Depending on how the value was introduced, the relevant trivial term is either a variable by SIM-TRIV-VAR, or a literal lambda abstraction by SIM-TRIV-LAM. The simulation relation $\rightsquigarrow$ includes the target semantics environment, in order to establish the conditions for the simulating terms to evaluate to the correct values. It is trivial to prove the lemma that these trivial terms evaluate under the included environment to values satisfying the CPS equivalence relation:

Lemma 1 (Simulated values evaluate to CPS-equivalent values).

$$\vdash \forall\, v\, \boldsymbol{\rho}\, \boldsymbol{t}\, \boldsymbol{v}\,.\, v \rightsquigarrow \boldsymbol{\{\rho\}t} \Longrightarrow (\boldsymbol{\{\rho\}t} \longrightarrow \boldsymbol{v} \wedge v \sim \boldsymbol{v})$$

Evaluation contexts. Our eventual simulation relation on semantic states is built on the transform functions $\mathscr{E}$, $\mathscr{C}$, and $\mathscr{K}$ from Fig. 6, and these functions operate on evaluation contexts parameterised by trivial terms in CPS $E^{\star}_{\mathbf{Triv}}$. We must therefore define a relation between the value-parameterised evaluation contexts

$$\frac{}{[]_{\mathrm{Val}} \rightsquigarrow \{\boldsymbol{\rho}\}[]_{\mathbf{Triv}}}\ \textsc{sim-cont-id} \qquad \frac{E^\star_{\mathrm{Val}} \rightsquigarrow \{\boldsymbol{\rho}\}E^\star_{\mathbf{Triv}}}{E^\star_{\mathrm{Val}}[[]\, e] \rightsquigarrow \{\boldsymbol{\rho}\}E^\star_{\mathbf{Triv}}[[]\, e]}\ \textsc{sim-cont-fn}$$

$$\frac{E^\star_{\mathrm{Val}} \rightsquigarrow \{\boldsymbol{\rho}\}E^\star_{\mathbf{Triv}} \qquad v \rightsquigarrow \{\boldsymbol{\rho}\}\boldsymbol{t}}{E^\star_{\mathrm{Val}}[v\, []] \rightsquigarrow \{\boldsymbol{\rho}\}E^\star_{\mathbf{Triv}}[\boldsymbol{t}\, []]}\ \textsc{sim-cont-arg-triv}$$

$$\frac{E^\star_{\mathrm{Val}} \rightsquigarrow \{\boldsymbol{\rho}\}E^\star_{\mathbf{Triv}} \qquad v \sim \boldsymbol{\rho}(\boldsymbol{m_i}) \qquad i = \mathsf{fresh}\ E^\star_{\mathbf{Triv}}}{E^\star_{\mathrm{Val}}[v\, []] \rightsquigarrow \{\boldsymbol{\rho}\}E^\star_{\mathbf{Triv}}[\boldsymbol{m_i}\, []]}\ \textsc{sim-cont-arg-comp}$$

Fig. 9. Evaluation context relation.

of the source semantics $E^\star_{\mathrm{Val}}$ and the trivial CPS term-parameterised evaluation contexts used by the transform (Fig. 9).

The empty context and fn stack frames $E^\star[[]\, e]$ trivially relate to their counterparts, as they do not contain any values. For arg stack frames $E^\star[v\, []]$, there are two possibilities for the stashed trivial term. The SIM-CONT-ARG-TRIV rule captures the case that the value was evaluated from a trivial term, and the resultant arg context $E^\star[v\, []]$ arises from a static application of a fn continuation to a trivial term $\mathscr{C}_{\boldsymbol{K}}[\![E^\star[[]\, e]]\!]\, \boldsymbol{t}$.

The SIM-CONT-ARG-COMP rule instead captures the case that the value resulted from a computation, i.e. an application. In this case, it is a dynamic application of a fn continuation given by the closure evaluated from $\mathscr{K}_{\boldsymbol{K}}[\![E^\star[[]\, e]]\!]$ which gives rise to the resulting arg continuation. The computed value is bound to the variable $\boldsymbol{m_i}$ and, importantly, only depends on the context of the local scope $E^\star$, as per the definition of $\mathscr{K}$.

Note that, though the simulation relation $\rightsquigarrow$ usually relates source semantic constructs to CPS terms with an attached environment, the relation on evaluation contexts only goes halfway, relating to similar evaluation contexts but with CPS terms in their value slots. It is the CPS transform applied to these evaluation contexts that produces the final simulating CPS terms, as we show in the following sections.

Scopes. Consider how the value bound to $\boldsymbol{m_i}$ in the relevant case of the evaluation context relation above arises, as the result of a computation. In the optimised transform, such values are bound to introduced variables such as $\boldsymbol{n}$ and $\boldsymbol{m_i}$ in continuation closures evaluated from $\mathscr{K}$, and arise from the application of an underlying continuation $\boldsymbol{k}$ being applied to a trivial term from the definition of $\mathscr{C}$ for an empty evaluation context $\mathscr{C}_{\boldsymbol{k}}[\![[]]\!]\, \boldsymbol{t} = \boldsymbol{k}\, \boldsymbol{t}$. Note, then, that this application of an underlying continuation must correspond to the case in the source semantics of an empty evaluation context with an underlying scope:

$$O(\{\rho'\}E^\star(\{\rho\}[v])) \longrightarrow^\star O(\{\rho'\}E^\star[v])$$

hence it is *changing scope* which results in a value becoming a computation. The SIM-SCOPE-CONT rule in Fig. 10 captures this correspondence by relating scopes to underlying continuation closures evaluated from $\mathscr{K}$ which are bound to $\boldsymbol{k}$.

$$\frac{}{() \rightsquigarrow \{\boldsymbol{\rho}\}\mathbf{I}}\ \textsc{sim-scope-id}$$

$$\frac{O \rightsquigarrow \{\boldsymbol{\rho'}\}\boldsymbol{K} \qquad \rho' \sim \boldsymbol{\rho'} \qquad E^{\star}_{\mathrm{Val}} \rightsquigarrow \{\boldsymbol{\rho'}\}E^{\star}_{\mathbf{Triv}} \qquad \{\boldsymbol{\rho'}\}\mathscr{K}_{\boldsymbol{K}}[\![E^{\star}_{\mathbf{Triv}}]\!] \longrightarrow \boldsymbol{\rho}(\boldsymbol{k})}{O(\{\rho'\}E^{\star}_{\mathrm{Val}}()) \rightsquigarrow \{\boldsymbol{\rho}\}\boldsymbol{k}}\ \textsc{sim-scope-cont}$$

Fig. 10. Scoping relation.

Semantic state relation. Combining the relations covered in this section, we arrive at a full simulation relation which relates source semantic states to CPS terms and an appropriate environment to simulate those semantic states. The relation is given in Fig. 11.

$$\frac{O \rightsquigarrow \{\boldsymbol{\rho}\}\boldsymbol{K} \qquad \rho \sim \boldsymbol{\rho} \qquad E^{\star}_{\mathrm{Val}} \rightsquigarrow \{\boldsymbol{\rho}\}E^{\star}_{\mathbf{Triv}}}{O(\{\rho\}E^{\star}_{\mathrm{Val}}[e]) \rightsquigarrow \{\boldsymbol{\rho}\}\mathscr{E}_{\boldsymbol{K}}[\![E^{\star}_{\mathbf{Triv}}[e]]\!]}\ \textsc{sim-exp}$$

$$\frac{O \rightsquigarrow \{\boldsymbol{\rho}\}\boldsymbol{K} \qquad \rho \sim \boldsymbol{\rho} \qquad E^{\star}_{\mathrm{Val}} \rightsquigarrow \{\boldsymbol{\rho}\}E^{\star}_{\mathbf{Triv}} \qquad v \rightsquigarrow \{\boldsymbol{\rho}\}\boldsymbol{t}}{O(\{\rho\}E^{\star}_{\mathrm{Val}}[v]) \rightsquigarrow \{\boldsymbol{\rho}\}\mathscr{C}_{\boldsymbol{K}}[\![E^{\star}_{\mathbf{Triv}}[]]\!]\,\boldsymbol{t}}\ \textsc{sim-triv}$$

$$\frac{O(\{\rho'\}E^{\star}_{\mathrm{Val}}()) \rightsquigarrow \{\boldsymbol{\rho}\}\boldsymbol{k} \qquad v \rightsquigarrow \{\boldsymbol{\rho}\}\boldsymbol{t}}{O(\{\rho'\}E^{\star}_{\mathrm{Val}}[v]) \rightsquigarrow \{\boldsymbol{\rho}\}\boldsymbol{k}\,\boldsymbol{t}}\ \textsc{sim-comp}$$

Fig. 11. Semantic state relation.

The resultant CPS expressions of the simulation relation are constructed from the definitions of the optimised CPS transform that we began with, each rule corresponding to each transform function. The first, SIM-EXP, relates an expression to its CPS transform by $\mathscr{E}$ in the corresponding evaluation context. The last two rules relate values to either their static application by $\mathscr{C}$ (SIM-TRIV) or dynamic application (SIM-COMP) by the closure evaluated from $\mathscr{K}$.

We choose to include SIM-COMP rather than attempt to use $\mathscr{C}$ with computed values as well as trivial ones, because it allows us to always relate the control string value to a directly related trivial term by the value simulation relation instead of having to also consider the possibility of binding the related value to an introduced variable such as $\boldsymbol{n}$. As a result, this also allows us to ignore introduced variables if they are not stashed in the evaluation context, i.e. we never need to consider the presence of $\boldsymbol{n}$ in the relation.

6 Simulation proof

Theorem 1 (Simulation).

$$\begin{aligned}\vdash \forall\, & O\, O'\, \rho\, \rho'\, E^\star\, E^{\star\prime}\, c^\star\, c^{\star\prime}\, \boldsymbol{\rho}\, \boldsymbol{e}\,. \\ & O(\{\rho\}E^\star[c^\star]) \longrightarrow^\star O'(\{\rho'\}E^{\star\prime}[c^{\star\prime}]) \wedge O(\{\rho\}E^\star[c^\star]) \rightsquigarrow \boldsymbol{\{\rho\}e} \\ & \implies \\ & \exists\, n\, \boldsymbol{\rho'}\, \boldsymbol{e'}\,.\, \boldsymbol{\{\rho\}e} \longrightarrow^n \boldsymbol{\{\rho'\}e'} \wedge O'(\{\rho'\}E^{\star\prime}[c^{\star\prime}]) \rightsquigarrow \boldsymbol{\{\rho'\}e'}\end{aligned}$$

Static reductions. The majority of the cases of Theorem 1 for small steps in the source semantics result in no reductions in the target semantics, which is consistent with the nature of the optimised transform. For example, the CBV-EXP-VAR and CBV-EXP-LAM steps for evaluation of trivial terms to values correspond only to expansions from the expression transform $\mathscr{E}_{\boldsymbol{K}}[\![E^\star[t]]\!]$ to the static continuation application transform $\mathscr{C}_{\boldsymbol{K}}[\![E^\star]\!]\,\boldsymbol{t}$ where the source trivial term t evaluates to a value which is simulated by the target trivial term $v \rightsquigarrow \boldsymbol{\{\rho\}t}$. In effect, these steps which do not involve dynamic reductions instead correspond to static reductions.

Another case where there are no reductions in the target semantics is for the CBV-CONT-ID reduction, which is the application to a value of an empty evaluation context with an underlying scope, as the simulation relation explicitly captures the introduction of a value as a computation by changing scope. The semantic state $O(\{\rho'\}E^\star(\{\rho\}[v]))$ relates either to $\boldsymbol{\{\rho\}}\mathscr{C}_{\boldsymbol{k}}[\![[]]\!]\,\boldsymbol{t}$ which simply expands $\boldsymbol{\{\rho\}k\,t}$ without reduction, or to $\boldsymbol{\{\rho^+\}k\,t}$ with a different environment which maps $\boldsymbol{k}$ to the evaluation of $\mathscr{K}_{\boldsymbol{k}}[\![[]]\!]$ under $\boldsymbol{\rho}$, which is again just whatever $\boldsymbol{k}$ maps do under $\boldsymbol{\rho}$, hence we similarly arrive at $\boldsymbol{\{\rho\}k\,t}$.

Computations. The simulation proof becomes interesting when considering a dynamic application of a continuation to a value, i.e. the value was a computation. The starting point for such a step is from a semantic state under the SIM-COMP rule, where the continuation closure in $\boldsymbol{k}$ is applied to term. Let us consider the case for the CBV-CONT-FN step, where the evaluation context is $E^\star[[]\,e]$; by applying the corresponding closure in $\boldsymbol{k}$, we arrive at the term $\mathscr{E}[\![E^\star[\boldsymbol{m_i}\,[e]]]\!]$ where the value that the continuation was applied to is bound to $\boldsymbol{m_i}$.

To satisfy the resultant simulation relation, we must ensure that the environment with the newly bound variable $\boldsymbol{\rho}[\boldsymbol{m_i} \mapsto \boldsymbol{v}]$ is consistent with the conditions on it imposed by the subrelations of the simulation relation, i.e. we need to prove that various relations on values, evaluation contexts, etc. are monotonic with respect to the additional binding to the introduced variable $\boldsymbol{m_i}$. For the environment CPS equivalence and value simulation relations, this is simple to prove if the transform ensures that variables from the original semantics are always distinct from introduced variables such as $\boldsymbol{m_i}$, $\boldsymbol{k}$, etc.

Lemma 2 (Environment relation monotonicity).

$$\vdash \forall\, \rho\, \boldsymbol{\rho}\, \boldsymbol{x}\, \boldsymbol{v}\,.\, (\forall\, x\,.\, \boldsymbol{x} \neq x) \implies \rho \sim \boldsymbol{\rho} \implies \rho \sim \boldsymbol{\rho}[\boldsymbol{x} \mapsto \boldsymbol{v}]$$

Lemma 3 (Value simulation monotonicity).

$$\vdash \forall\, v\, \boldsymbol{\rho}\, \boldsymbol{t}\, \boldsymbol{x}\, \boldsymbol{v}\,.\, (\forall\, x\,.\, \boldsymbol{x} \neq x) \Longrightarrow v \rightsquigarrow \boldsymbol{\{\rho\}t} \Longrightarrow v \rightsquigarrow \boldsymbol{\{\rho[x \mapsto v]\}t}$$

Proof. Trivial.

Monotonicity for the scoping relation is similarly trivial, as that relation depends only on the binding of $\boldsymbol{k}$.

Lemma 4 (Scoping relation monotonicity).

$$\vdash \forall\, O\, \boldsymbol{\rho}\, \boldsymbol{K}\, \boldsymbol{x}\, \boldsymbol{v}\,.\, \boldsymbol{x} \neq \boldsymbol{k} \Longrightarrow O \rightsquigarrow \boldsymbol{\{\rho\}K} \Longrightarrow O \rightsquigarrow \boldsymbol{\{\rho[x \mapsto v]\}K}$$

Proof. Trivial.

The interesting monotonicity lemma, and really the pin holding together simulation for an optimised transform, is the monotonicity lemma for evaluation contexts. Not only must introduced variables not clash with source variables, but they must also not clash with the *other* introduced variables in the evaluation contexts, i.e. other $\boldsymbol{m_i}$ variables as per the SIM-CONT-ARG-COMP rule. Using the implementation of fresh described in Section 3.4, which counts the number of $\boldsymbol{m}$ variables in $E^\star$, the variable index must simply be at least $\mathsf{fresh}\ E^\star$ to not clash. We then arrive at the evaluation context lemma of monotonicity:

Lemma 5 (Evaluation context relation monotonicity).

$$\begin{aligned}
&\vdash \forall\, E^\star_{\mathrm{Val}}\, \boldsymbol{\rho}\, E^\star_{\mathbf{Triv}}\, \boldsymbol{x}\, \boldsymbol{v}\,. \\
&\quad (\forall\, x\,.\, \boldsymbol{x} \neq x) \wedge (\forall\, i < \mathsf{fresh}\ E^\star_{\mathbf{Triv}}\,.\, \boldsymbol{x} \neq \boldsymbol{m_i}) \\
&\quad \Longrightarrow E^\star_{\mathrm{Val}} \rightsquigarrow \boldsymbol{\{\rho\}} E^\star_{\mathbf{Triv}} \Longrightarrow E^\star_{\mathrm{Val}} \rightsquigarrow \boldsymbol{\{\rho[x \mapsto v]\}} E^\star_{\mathbf{Triv}}
\end{aligned}$$

Proof. By induction over the SIM-CONT rules.

This lemma is the crux of our proof. The deterministic nature of the introduced variable names in the simulation relation, as a result of the local evaluation-context-based fresh name generation in our reformulated transform, enables us to prove monotonicity of the simulation relation with respect to the binding of introduced variables, even with multiple $\boldsymbol{m}$ variables already introduced into the scope.

Application. The CBV-CONT-ARG step, on the evaluation context $E^\star[v\,[]]$, always produces reductions in the target domain, regardless of if the continuation is applied to a trivial or computed term. Indeed, to simulate the source semantics, a completed reduction of an application expression in the source domain must correspond to a reduction of an application in the target domain; the transform produces the application $\boldsymbol{t_1}\, \boldsymbol{t_2}\, \mathscr{K}_{\boldsymbol{K}}[\![E^\star]\!]$ in CPS which must be reduced.

By the simulation relation, $\boldsymbol{t_1}$ will always evaluate to a closure $\boldsymbol{\Lambda x.\{\rho\}\lambda k.}\mathscr{E}_{\boldsymbol{k}}[\![e]\!]$, hence the application leads to the expression $\mathscr{E}_{\boldsymbol{k}}[\![e]\!]$ with $\boldsymbol{x}$ and $\boldsymbol{k}$ freshly bound to the passed value and underlying continuation evaluated from $\mathscr{K}_{\boldsymbol{K}}[\![E^\star]\!]$ respectively. The corresponding environment in the source semantics similarly has the variable x bound, and we may trivially prove that the environment relation is preserved when binding a source variable.

Lemma 6 (Environment relation synchronisation).

$$\vdash \forall\, \rho\, \boldsymbol{\rho}\, x\, v\, \boldsymbol{v}\,.\, v \sim \boldsymbol{v} \Longrightarrow \rho \sim \boldsymbol{\rho} \Longrightarrow \rho[x \mapsto v] \sim \boldsymbol{\rho}[\boldsymbol{x} \mapsto \boldsymbol{v}]$$

Proof. Cases on $x \in \mathrm{dom}(\rho)$.

Before Lemma 6 may be applied, we must first apply the environment monotonicity lemma, Lemma 2, to disregard the binding of $\boldsymbol{k}$.

Note that, in the case that the continuation was applied to a computed value, the value is bound to $\boldsymbol{n}$ then very quickly dispatched. We must then also apply the environment, scoping, and evaluation context monotonicity lemmas, Lemmas 2, 4, and 5, to disregard the binding of $\boldsymbol{n}$.

7 Conclusion

We have presented a reformulation of Danvy & Nielsen's one-pass CPS transform [9] in the style of Sabry & Felleisen's optimised transform based on evaluation contexts [21]. Our transform highlights the nature of the name clashes encountered in Minamide & Okuma's mechanised correctness proof [15]: clashes are local to the lexical scope, and may be reasoned about by the inclusion of the offending variables in the local evaluation context. It is hence possible to use a fresh variable generation method that depends only on the local evaluation context, and allows introduced variable names to be uniquely determined by the simulation relation.

We have proposed a modified CEK-machine to represent the source semantics, with evaluation contexts rephrased as nested sequences of lexical scopes, each with its own local environment and evaluation context. In combination with our new transform, this semantics builds naturally into a simulation relation which can determine introduced variable names, that consequently enables a straightforward simulation proof, with no need to consider α-equivalence. The modified CEK-machine represents only a small departure from a typical CEK-machine or alternative semantics and is easy to prove equivalent, hence we believe that it is a beneficial trade-off for the more difficult proof of α-equivalence.

The nature of our transform to use evaluation contexts from the semantics should allow it to be applied more generally to richer languages. We would particularly like to investigate the use of an evaluation-context-based CPS transform to languages with control operators such as `call/cc` or delimited continuations, which would likely have interesting variable clashing and fresh naming challenges for captured continuation variables.

Acknowledgments We thank the PADL 2026 reviewers for helpful comments.

References

1. Ager, M.S., Biernacki, D., Danvy, O., Midtgaard, J.: A functional correspondence between evaluators and abstract machines. In: Proceedings of the 5th ACM SIG-

PLAN International Conference on Principles and Practice of Declaritive Programming. p. 8–19. PPDP '03, Association for Computing Machinery, New York, NY, USA (2003). https://doi.org/10.1145/888251.888254
2. Baker, H.G.: Cons should not cons its arguments, part ii: Cheney on the M.T.A. SIGPLAN Not. **30**(9), 17–20 (Sep 1995). https://doi.org/10.1145/214448.214454
3. Chlipala, A.: Parametric higher-order abstract syntax for mechanized semantics. In: Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming. p. 143–156. ICFP '08, Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1411204.1411226
4. Danvy, O., Filinski, A.: Representing control: a study of the CPS transformation. Mathematical Structures in Computer Science **2**(4), 361–391 (1992). https://doi.org/10.1017/S0960129500001535
5. Danvy, O.: On evaluation contexts, continuations, and the rest of computation (02 2004)
6. Danvy, O.: Defunctionalized interpreters for programming languages. In: Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming. p. 131–142. ICFP '08, Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1411204.1411206
7. Danvy, O., Filinski, A.: Abstracting control. In: Kahn, G. (ed.) Proceedings of the 1990 ACM Conference on LISP and Functional Programming, LFP 1990, Nice, France, 27-29 June 1990. pp. 151–160. ACM (1990). https://doi.org/10.1145/91556.91622, https://doi.org/10.1145/91556.91622
8. Danvy, O., Nielsen, L.R.: Defunctionalization at work. In: Proceedings of the 3rd ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming. p. 162–174. PPDP '01, Association for Computing Machinery, New York, NY, USA (2001). https://doi.org/10.1145/773184.773202
9. Danvy, O., Nielsen, L.R.: A first-order one-pass CPS transformation. Theoretical Computer Science **308**(1), 239–257 (2003). https://doi.org/10.1016/S0304-3975(02)00733-8
10. Dargaye, Z., Leroy, X.: Mechanized verification of CPS transformations. In: Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning. p. 211–225. LPAR'07, Springer-Verlag, Berlin, Heidelberg (2007)
11. Felleisen, M., Friedman, D.P.: Control operators, the SECD-machine, and the λ-calculus. In: Formal Description of Programming Concepts (1987), https://api.semanticscholar.org/CorpusID:57760323
12. Kelsey, R.A.: A correspondence between continuation passing style and static single assignment form. In: Papers from the 1995 ACM SIGPLAN Workshop on Intermediate Representations. p. 13–22. IR '95, Association for Computing Machinery, New York, NY, USA (1995). https://doi.org/10.1145/202529.202532
13. Kranz, D., Kelsey, R., Rees, J., Hudak, P., Philbin, J., Adams, N.: Orbit: an optimizing compiler for Scheme. In: Proceedings of the 1986 SIGPLAN Symposium on Compiler Construction. p. 219–233. SIGPLAN '86, Association for Computing Machinery, New York, NY, USA (1986). https://doi.org/10.1145/12276.13333
14. Lasnier, P.Y., Yallop, J., Myreen, M.O.: Brack: A verified compiler for Scheme via CakeML. In: Proceedings of the 15th ACM SIGPLAN International Conference on Certified Programs and Proofs. CPP '26, Association for Computing Machinery, New York, NY, USA (2026)

15. Minamide, Y., Okuma, K.: Verifying CPS transformations in Isabelle/HOL. In: Proceedings of the 2003 ACM SIGPLAN Workshop on Mechanized Reasoning about Languages with Variable Binding. p. 1–8. MERLIN '03, Association for Computing Machinery, New York, NY, USA (2003). https://doi.org/10.1145/976571.976576
16. Paraskevopoulou, Z., Grover, A.: Compiling with continuations, correctly. Proc. ACM Program. Lang. 5(OOPSLA) (Oct 2021). https://doi.org/10.1145/3485491
17. Pitts, A.M.: Nominal logic, a first order theory of names and binding. Inf. Comput. 186(2), 165–193 (2003). https://doi.org/10.1016/S0890-5401(03)00138-X, https://doi.org/10.1016/S0890-5401(03)00138-X
18. Plotkin, G.: Call-by-name, call-by-value and the λ-calculus. Theoretical Computer Science 1(2), 125–159 (1975). https://doi.org/10.1016/0304-3975(75)90017-1
19. Pottier, F.: Revisiting the CPS transformation and its implementation (2006)
20. Reynolds, J.C.: Definitional interpreters for higher-order programming languages. In: Proceedings of the ACM Annual Conference - Volume 2. p. 717–740. ACM '72, Association for Computing Machinery, New York, NY, USA (1972). https://doi.org/10.1145/800194.805852
21. Sabry, A., Felleisen, M.: Reasoning about programs in continuation-passing style. In: Proceedings of the 1992 ACM Conference on LISP and Functional Programming. p. 288–298. LFP '92, Association for Computing Machinery, New York, NY, USA (1992). https://doi.org/10.1145/141471.141563
22. Steele, G.L.: Rabbit: A compiler for Scheme. Tech. rep., USA (1978)
23. Tian, Y.H.: Mechanically verifying correctness of CPS compilation. In: Proceedings of the Twelfth Computing: The Australasian Theory Symposium - Volume 51. p. 41–51. CATS '06, Australian Computer Society, Inc., AUS (2006)
24. Xiao, Y., Sabry, A., Ariola, Z.M.: From syntactic theories to interpreters: Automating the proof of unique decomposition. Higher Order Symbol. Comput. 14(4), 387–409 (Dec 2001). https://doi.org/10.1023/A:1014408032446