Abstract machines for higher-order term sharing

Connor Lane Smith
University of Kent
cls204@kent.ac.uk

Abstract

In this paper we take the Krivine machine, a simple abstract machine for weak β-reduction, and augment it with the σ-calculus to unlock strong reduction. We demonstrate that this abstract machine can be used to drive higher-order rewriting, and with some alterations can be used for 'higher-order term sharing': rather than normalising a term at each rewrite step, we view the lambda calculus itself as a sophisticated sharing mechanism, and make use of it so as to avoid needless duplication of rewrite steps.

1. Introduction

Higher-order term rewriting [9] is a powerful generalisation of first-order term rewriting in which rewrite steps are performed modulo the simply-typed λ-calculus. Terms are generally assumed to be normalised after each rewrite step, but doing so loses the sharing of subterms present in the unreduced term. For example, the normalisation $(\lambda A\underline{00})B \to_\beta ABB$ *unshares* B from one instance to two, meaning if we wish to rewrite B to C then we must now do it twice instead of only once.

In this paper we explore two known mechanisms for reducing λ-terms: the Krivine machine [6], a simple abstract machine for weak β-reduction; and the σ-calculus [1], an explicit substitution calculus. We then introduce a new 'Kσ-machine' combining these two approaches into an abstract machine for strong β-reduction.

By further extending this machine, and tracing the provenance of the subterms forming a rewrite redex in a term's normal form, we can compute a reduction that reduces only the part of the term needed to reveal that redex. This allows us to maintain during reduction the sharing present in λ-terms, rather than β-normalising rewritten terms as is generally done. This sharing scheme parallels Wadsworth's [11] 'first-order' mechanism for sharing using dags. Comparatively, we make steps towards 'higher-order term sharing'.

2. Preliminaries

We will assume familiarity with the simply-typed λ-calculus [2], and will avoid the complexities of named variables by using De Bruijn indices [3] exclusively.

Definition 1. The set *simple types* is the closure of a fixed set of *type atoms* under the binary function type constructor $\to$. Notation: $\to$ is right-associative: $\alpha \to \beta \to \gamma = \alpha \to (\beta \to \gamma)$.

Definition 2. We use ϵ for the empty list, and :: for the 'cons' operator — i.e. datatype $\text{List}(\alpha) = \epsilon \mid \alpha :: \text{List}(\alpha)$. We write $\alpha \cdot \beta$ for the concatenation of α and β.

Definition 3. A *basis* Γ is a list of simple types, with which we may derive a simply-typed *term* $t \in \mathbb{T}$, written $\Gamma \vdash t : \tau$.

$$\begin{aligned} \sigma :: \Gamma \vdash \underline{0} : \sigma \\ \Gamma \vdash \underline{n} : \sigma \implies \rho :: \Gamma \vdash \underline{n{+}1} : \sigma \\ \sigma :: \Gamma \vdash t : \tau \implies \Gamma \vdash \lambda t : \sigma \to \tau \\ \Gamma \vdash t_1 : \sigma \to \tau \wedge \Gamma \vdash t_2 : \sigma \implies \Gamma \vdash t_1 t_2 : \tau \\ K : \tau \implies \Gamma \vdash K : \tau \end{aligned}$$

Definition 4. A *context* C is a λ-term containing a single unique 'hole' symbol $\Box$. The hole in C may be 'filled' by a term t, written $C[t]$, replacing the hole with the term t with no variable adjustment. Contexts may be composed, such that $(C_1\,;C_2)[t] \equiv C_2[C_1[t]]$.

Definition 5. A *substitution* θ is a mapping from variables to terms of the same type, which may be lifted to a homomorphism over terms, written $\hat{\theta}(t)$.

Definition 6. We write β-reduction as $\to_\beta$, η-reduction $\to_\eta$, and their union, γ-reduction, $\to_\gamma = \to_\beta \cup \to_\eta$. All are closed under contexts and substitutions. There are a number of subrelations of β-reduction:

- Weak reduction – as β-reduction, but reductions cannot be performed under a lambda. Full β-reduction may in contrast be called 'strong reduction'.
- Head reduction – β-reduction of only the leftmost subterm, such that the *head* — the leftmost atom (constant or variable) applied to a 'spine' of argument terms — is found.
- Weak head reduction – head reduction that is also weak, such that the *weak head* of a term may alternatively be a lambda that does not form a β-redex.

A normal form — be it weak, strong, head, or weak head — is a term that cannot be reduced by the respective relation. Any simply-typed term has a unique normal form for each of these subrelations.

3. Explicit reduction

3.1 K-machine

The *Krivine machine* [6], or 'K-machine', is an abstract machine for the weak β-reduction of λ-terms. The machine $\langle\langle t, e\rangle \mid \mathit{stack}\rangle$ has three components: the *term* t being reduced; the *environment* e, a stack of term–environment pairs with the topmost corresponding to De Bruijn index 0 and the bottom to index n; and the *stack*, likewise a stack of term–environment pairs, but from which items are taken as arguments to applied lambdas. The machine is therefore of

the following type, where the μ operator yields the least fixpoint of a type.

$$(\mu\alpha.\,\mathbb{T} \times \text{List}(\alpha)) \times \text{List}(\mu\alpha.\,\mathbb{T} \times \text{List}(\alpha))$$

The mechanics of the K-machine are defined in terms of a set of *transition rules*, detailed in Figure 1. The machine simulates weak β-reduction, in that, given two machine configurations m and m' and terms t and t', if $m \sim t$ and $m' \sim t'$, then $m \to^* m' \implies t \to^*_\beta t'$, for the relation $\sim$:

$$\langle\langle t_0, e_0\rangle \mid \langle t_1, e_1\rangle :: \ldots :: \langle t_n, e_n\rangle :: \epsilon\rangle \sim \hat{e}_0(t_0)\hat{e}_1(t_1)\ldots\hat{e}_n(t_n)$$

When reducing a term t, we run the machine $\langle\langle t, \epsilon\rangle \mid \epsilon\rangle$ until it halts. There are two possible configurations in which the machine halts: $\langle\langle \underline{n}, \epsilon\rangle \mid \mathit{stack}\rangle$ and $\langle\langle K, e\rangle \mid \mathit{stack}\rangle$. In either case, $t_0 \equiv \hat{e}_0(t_0)\downarrow_\beta$, so the weak head normal form has been found. We can continue on to full weak normal form (i.e. all redexes not under a lambda have been reduced) by recursing over the terms and environments in the stack of the halted machine.

The reason the K-machine cannot perform strong β-reduction is there is no way to represent in its stack the offset in De Bruijn indices that would be required of the environment if the machine were to move under a lambda. For this we need a more sophisticated data structure, and for that we look towards explicit substitution.

3.2 $\lambda\sigma$-calculus

The *$\lambda\sigma$-calculus* [1] is a 'substitution calculus' that renders the higher-order λ-calculus into a first-order term rewriting system, thus formalising the mechanisms of substitution. σ-substitutions may be thought of as a data structure for representing arbitrary λ-calculus substitutions on De Bruijn indices. σ-substitutions have the following constructors:

$$\begin{aligned}
\text{id} &= \{n \mapsto n\} && \text{Identity}\\
\uparrow &= \{n \mapsto n{+}1\} && \text{Shift}\\
t \cdot \sigma &= \{0 \mapsto t, n{+}1 \mapsto \sigma(n)\} && \text{Cons}\\
\rho\,;\sigma &= \{n \mapsto \rho(n)[\sigma]\} && \text{Compose}
\end{aligned}$$

Note that I use the notation $\rho\,;\sigma$ where Abadi, et al. have used $\rho \circ \sigma$, so as to avoid any confusion between left-to-right and right-to-left composition. I also use the De Bruijn indices over the set $\mathbb{N} = \{0, 1, 2, \ldots\}$ rather than $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$. Neither of these notational changes have any effect on the calculus itself.

Definition 7. A σ-substitution σ is applied to a λ-term t, written $t[\sigma]$, like so:

$$\begin{aligned}
(t_1 t_2)[\sigma] &= t_1[\sigma]t_2[\sigma]\\
(\lambda t)[\sigma] &= \lambda(t[\underline{0} \cdot (\sigma\,;\uparrow)])\\
\underline{n}[\text{id}] &= \underline{n}\\
\underline{n}[\uparrow] &= \underline{n{+}1}\\
\underline{0}[t \cdot \sigma] &= t\\
\underline{n{+}1}[t \cdot \sigma] &= \underline{n}[\sigma]\\
\underline{n}[\rho\,;\sigma] &= \underline{n}[\rho][\sigma]\\
K[\sigma] &= K
\end{aligned}$$

β-reduction is defined as the relation $(\lambda s)t \to_\beta s[t \cdot \text{id}]$ closed under contexts and substitution.

Definition 8. We extend simply-typed λ-terms to the $\lambda\sigma$-calculus by introducing a typing for σ-substitutions. We write $\Gamma \vdash \sigma \triangleright \Gamma'$ to say that in the environment Γ the substitution σ has the environment Γ', by the following rules:

$$\begin{aligned}
\Gamma &\vdash \text{id} \triangleright \Gamma\\
\tau :: \Gamma &\vdash \uparrow \triangleright \Gamma\\
\Gamma \vdash t : \tau \wedge \Gamma \vdash \sigma \triangleright \Gamma' &\implies \Gamma \vdash t \cdot \sigma \triangleright \tau :: \Gamma'\\
\Gamma \vdash \rho \triangleright \Gamma' \wedge \Gamma' \vdash \sigma \triangleright \Gamma'' &\implies \Gamma \vdash \rho\,;\sigma \triangleright \Gamma''
\end{aligned}$$

We can then use this to derive simple types for σ-closures.

$$\Gamma \vdash \sigma \triangleright \Gamma' \wedge \Gamma' \vdash t : \tau \implies \Gamma \vdash t[\sigma] : \tau$$

3.3 Kσ-machine

We will now introduce the *Kσ-machine*, an extension of the K-machine in which the environment stack has been generalised to a σ-substitution. This unlocks reduction under lambda, as necessary for strong β-reduction. The Kσ-machine is similar to the machine described in [1], but the terms themselves are not modified in any way, much like the original Krivine machine. With this approach the structures of terms and substitutions are kept entirely separate, which means that the type of the machine's environment and stack is independent from that of the term's closures, if any. This will prove important in §4.2, where the machine's thunks' closures are labelled, but the terms' are not.

In order to keep the machine definition simple, we assume that σ-substitutions are always of the pattern $(\sigma_1\,;(\sigma_2\,;\ldots\,;(\sigma_n\,;\text{id})))$. The initial substitution (id) satisfies this pattern, and the rules of the machine maintain it. Additionally, we parametrise the substitution type, Subst(α), so that instead of 'cons' being of the type $\mathbb{T} \to \text{Subst} \to \text{Subst}$, it is of the type $\alpha \to \text{Subst}(\alpha) \to \text{Subst}(\alpha)$. These substitutions then form maps of the type $\mathbb{N} \to \alpha + \mathbb{N}$. In the case of the K$\sigma$-machine, they are of the type $\mu\alpha.\,\text{Subst}(\mathbb{T} \times \alpha)$. We call these term–substitution pairs *thunks*.

These σ-substitutions alone do not quite give us full strong reduction, however. Analogous to the K-machine halting at weak head normal form, we would expect the Kσ-machine to halt at (strong) head normal form $\lambda\ldots\lambda t_0(t_1[\sigma_1])\ldots(t_n[\sigma_n])$, where t_0 is a variable or constant. But if we are to pass under a lambda, we must discard it from the term, and we cannot know how many lambdas are in the head normal form. The solution used in [1] is to suspend the machine at this point, and to handle the 'Lambda' case external to the machine definition. We take a different tack: as well as generalising environments to substitutions, we generalise the stack to a context. This deviation will prove useful in §4.3, where it drastically simplifies the operation of the machine.

Definition 9. A *zipper* [5] is a term representation of a context, or a suspended traversal through a term. A zipper for the λ-calculus is a first-order term with the following signature:

$$\begin{aligned}
@_l^\alpha &: \text{Context}(\alpha) \times \alpha \to \text{Context}(\alpha)\\
@_r^\alpha &: \alpha \times \text{Context}(\alpha) \to \text{Context}(\alpha)\\
\Lambda^\alpha &: \text{Context}(\alpha) \to \text{Context}(\alpha)\\
\top^\alpha &: \text{Context}(\alpha)
\end{aligned}$$

We omit the superscript type parameter where it may be inferred. Each symbol has a meaning as a context, and may be thought of as a traversal through a term.

- $@_l(C, t)$ represents the context $(\Box t\,; C)$.
- $@_r(t, C)$ represents the context $(t\Box\,; C)$.
- $\Lambda(C)$ represents the context $(\lambda\Box\,; C)$.
- $\top$ represents the trivial context $\Box$.

Example 1. *A context $(\lambda\Box)K$ is represented by the zipper term $\Lambda(@_l(\top, K))$, and $(\lambda\underline{0})\Box$ the zipper term $@_r(\lambda\underline{0}, \top)$.*

Where the K-machine has $\langle t, e\rangle :: \mathit{stack}$, the K$\sigma$-machine has $\Box(t[\sigma])\,; C$, i.e. $@_l(C, t[\sigma])$. Yet we can also add lambdas through

Figure 1. K-machine

$$\langle\langle t_1 t_2, e\rangle \mid \mathit{stack}\rangle \to \langle\langle t_1, e\rangle \mid \langle t_2, e\rangle :: \mathit{stack}\rangle \quad \text{Apply}$$
$$\langle\langle \lambda t_1, e_1\rangle \mid \langle t_2, e_2\rangle :: \mathit{stack}\rangle \to \langle\langle t_1, \langle t_2, e_2\rangle :: e_1\rangle \mid \mathit{stack}\rangle \quad \text{Beta}$$
$$\langle\langle \underline{0}, \langle t, e_2\rangle :: e_1\rangle \mid \mathit{stack}\rangle \to \langle\langle t, e_2\rangle \mid \mathit{stack}\rangle \quad \text{Head}$$
$$\langle\langle \underline{n{+}1}, \langle t, e_2\rangle :: e_1\rangle \mid \mathit{stack}\rangle \to \langle\langle \underline{n}, e_1\rangle \mid \mathit{stack}\rangle \quad \text{Tail}$$

Figure 2. Kσ-machine

$$\langle (t_1 t_2)[\sigma] \mid C\rangle \to \langle t_1[\sigma] \mid \Box(t_2[\sigma])\,;C\rangle \quad \text{Left}$$
$$\langle (\lambda t)[\sigma] \mid \Box(u[\rho])\,;C\rangle \to \langle t[(u[\rho]\cdot\sigma)\,;\mathrm{id}] \mid C\rangle \quad \text{Beta}$$
$$\langle (\lambda t)[\sigma] \mid C\rangle \to \langle t[(\underline{0}[\mathrm{id}]\cdot(\sigma\,;\uparrow))\,;\mathrm{id}] \mid \lambda\Box\,;C\rangle \quad \text{Lambda}$$
$$\langle \underline{n}[(\pi\,;\rho)\,;\sigma] \mid C\rangle \to \langle \underline{n}[\pi\,;(\rho\,;\sigma)] \mid C\rangle \quad \text{Associate}$$
$$\langle \underline{0}[(u[\pi]\cdot\rho)\,;\sigma] \mid C\rangle \to \langle u[\pi\,;\sigma] \mid C\rangle \quad \text{Head}$$
$$\langle \underline{n{+}1}[(u[\pi]\cdot\rho)\,;\sigma] \mid C\rangle \to \langle \underline{n}[\rho\,;\sigma] \mid C\rangle \quad \text{Tail}$$
$$\langle \underline{n}[\uparrow\,;\sigma] \mid C\rangle \to \langle \underline{n{+}1}[\sigma] \mid C\rangle \quad \text{Shift}$$
$$\langle \underline{n}[\mathrm{id}\,;\sigma] \mid C\rangle \to \langle \underline{n}[\sigma] \mid C\rangle \quad \text{Id}$$

$$\langle (t_1 t_2)[\sigma] \mid C\rangle \to \langle t_2[\sigma] \mid (t_1[\sigma])\Box\,;C\rangle \quad \text{Right}$$
$$\langle (t[\rho])[\sigma] \mid C\rangle \to \langle t[\rho\,;\sigma] \mid C\rangle \quad \text{Closure}$$

which we have passed with $\lambda\Box\,;C$, i.e. $\Lambda(C)$. The Kσ-machine is therefore of the type,

$$(\mu\alpha.\,\mathbb{T} \times \mathrm{Subst}(\alpha)) \times \mathrm{Context}(\mu\alpha.\,\mathbb{T} \times \mathrm{Subst}(\alpha))$$

The Kσ-machine is defined in Figure 2. The machine definition has overlapping rules with which it is capable of any *standard reduction* [2], but during normal operation (β-normalisation) we assume that the 'Beta' rule is favoured over 'Lambda', and that the 'Left' rule is used instead of the optional 'Right'. There is also a 'Closure' rule for composing two substitutions into one; this is only necessary if the term structure itself may contain closures of the $\lambda\sigma$-calculus. Note that the machine's 'thunks' are separate from the term structure itself.

When reducing a term t, we run the machine $\langle t[\mathrm{id}] \mid \Box\rangle$ until it halts. Similar to the K-machine, the Kσ-machine simulates strong β-reduction;

$$\langle t[\sigma] \mid C\rangle \sim C[t[\sigma]]$$

4. Higher-order term sharing

Here we use Nipkow's Higher-order Rewrite Systems (HRSs) [9], or strictly speaking *higher-order pattern rewrite systems*, to which they are most commonly restricted.

Definition 10. A *higher-order pattern* [8] is a β-normal term with a constant at its head, and in which any free variable f may only in the form $f t_1 \ldots t_k$ where each t_i is η-equivalent to a distinct bound variable.

Definition 11. A Higher-order Rewrite System $\mathcal{H}$ is a set of *rewrite rules* $\langle l, r\rangle$ where l is a pattern and r a term of the same atomic type, and $\mathrm{FV}(l) \supseteq \mathrm{FV}(r)$. Each rule $\mathcal{R}$ induces a rewrite relation $t \to_{\mathcal{R}} t'$ where $\hat{\theta}(l) \leftrightarrow_\gamma^* t$ and $t' \leftrightarrow_\gamma^* \hat{\theta}(r)$. $\mathcal{H}$ then induces the union $\to_{\mathcal{H}} = \bigcup_{\mathcal{R}\in\mathcal{H}} \to_{\mathcal{R}}$.

Although HRSs rewrite modulo the simply-typed λ-calculus, it is generally assumed that the term is β-normalised after each rewrite step. But if it is not, then the β-reduction potential of the term acts as a kind of term sharing mechanism. For instance, a first-order β-redex $(\lambda t_1)t_2$ 'shares' the term t_2 amongst all instances of the variable $\underline{0}$ in t_1; a rewrite step may take place in t_2 and it will in effect have occurred in any number of positions in t_1 in the term's β-normal form.

Example 2. *Given a rewrite system $\{\langle B, C\rangle\}$, a term $(\lambda A\underline{00})B$ can be written in one step to $(\lambda A\underline{00})C$, the β-normal form of which is ACC. But if we β-normalise the term first to ABB, it takes* two *steps, via either ABC or ACB, to reach ACC. This demonstrates that B is being* shared *by the β-redex in the initial term.*

The sharing arrangement in Example 2 can also be achieved by representing a term as a dag, such that after the β-reduction of a term $(\lambda t_1)t_2$ all residual instances of t_2 in t_1 are references to the same term, which can then be rewritten. This form of sharing was formalised by Wadsworth [11]. However, the sharing offered by the simply-typed λ-calculus as a whole, higher-order β-redexes included, is in general more powerful than Wadsworth's dags. With dags, only full terms may be shared; if two terms are almost identical, but one has one term where the other has another, then they cannot be shared despite their similarities.

Example 3. *Given a term $(\lambda C(\underline{0}A)(\underline{0}B))(\lambda t)$, reducing the root β-redex would yield $C((\lambda t)A)((\lambda t)B)$; with dag sharing the two instances of λt would be shared. But further reduction to Ct_1t_2 would require t_1 and t_2 to become unshared, as the former contains A where the latter does B. That the simply-typed λ-calculus* can *share t_1 and t_2, in the single instance of t, demonstrates that it is in general more powerful than dag sharing.*

The question, then, is how to perform higher-order rewriting without unsharing unnecessarily. This is related to Lévy's *optimal reduction* [7], but is not the same: we are using the $\lambda(\sigma)$-calculus as the sharing mechanism, not as the term rewriting system being shared. The notion that rewriting between non-normalised terms

in a higher-order rewriting system can be seen as a kind of term sharing is mentioned in [10].

4.1 β-traversals

Since β-reduction introduces no term structure (its right-hand side as an HRS comprising only variables and applications), given a reduction $t \to_\beta^* t'$, every atom (variable or constant) present in t' will have originated somewhere in t, and through the process of reduction a copy will have been placed in its new position in t'. During this process, its arguments and the value to which itself is an argument may both have changed in any number of ways. If we are to understand how the atom arrived at its position in t', we would like to keep a track of the context of reductions and substitutions by which it got there.

Definition 12. A *β-traversal* may be thought of as a path through the β-reduction of a term. A β-traversal is a first-order term with the following signature:

$$\begin{aligned} @_l &: A \to A & \top &: A \\ @_r &: A \to A & \mathrm{B} &: A \to A \\ \Lambda &: A \to A & \Sigma &: A \times A \to A \end{aligned}$$

β-traversals are akin to the zipper, but the $@_l$ and $@_r$ symbols have as a subterm only a continuation of the context and not an extra α value as with $@_l^\alpha$ and $@_r^\alpha$. Furthermore, substitutions (Σ) may cause there to be more than one top ($\top$). In addition to the symbols in the signature of the zipper,

- B indicates a β-reduction having occurred in its subterm, which is itself somewhat 'inside-out', the lambda occurring outside the application with which it formed a β-redex.
- Σ indicates a substitution having occurred as a result of some β-reduction. It has two subterms, the first the traversal to the variable, the second to the substitute.

Example 4. *The β-traversal to the head of the reduct of a reduction $(\lambda\underline{0})K \to_\beta^* K$ is $\Sigma(\mathrm{B}(\Lambda(@_l(\top))), @_r(\top))$. The β-traversal $\Lambda(@_l(\top))$ represents the traversal down to the body of the lambda on the left-hand side of the application at the top, i.e. $(\lambda\Box)K$, although unlike Example 1 the right-hand side of that application is not made explicit: $(\lambda\Box)\_$.*

The B *symbol then shows that there has been a β-reduction between the application and the lambda, similar to a proof term, e.g. $\beta((\lambda\Box)\_)$. Finally, $\Sigma(\alpha, \beta)$ substitutes for the variable at traversal α the value at β, in this case effectively $\beta((\lambda\Box)K) \leftarrow_\Sigma (\lambda\underline{0})\Box$. The complete traversal term thus describes the position of the head 'through' the reduction.*

Definition 13. A β-traversal over $t \to_\beta^* t'$ may be converted into a *path*, a list of symbols identifying a subterm. Two paths of a traversal are particularly distinct: the *dynamic path*, the destination of the traversal relative to t'; and the *static path*, that relative to t. The dynamic path for a traversal α may be obtained by $path_1(\alpha, \epsilon)$, the static path $path_2(\alpha, \epsilon)$:

$$\begin{aligned} path_i(\top, p) &= p \\ path_i(@_l(\alpha), p) &= path_i(\alpha, @_l :: p) \\ path_i(@_r(\alpha), p) &= path_i(\alpha, @_r :: p) \\ path_i(\Lambda(\alpha), p) &= path_i(\alpha, \Lambda :: p) \\ path_i(\mathrm{B}(\alpha), p) &= path_i(\alpha, p) \\ path_1(\Sigma(\alpha, \beta), p) &= path_1(\alpha, p) \\ path_2(\Sigma(\alpha, \beta), p) &= path_2(\beta, p) \end{aligned}$$

$t|_p$ represents the subterm of the term t reached via path p.

$$\begin{aligned} t|_\epsilon &= t \\ (t_1 t_2)|_{@_l :: p} &= t_1|_p \\ (t_1 t_2)|_{@_r :: p} &= t_2|_p \\ (\lambda t)|_{\Lambda :: p} &= t|_p \end{aligned}$$

We can also refer to a subterm reached *modulo β-reduction*, $t|_{p/\beta}$, which is the same as $t|_p$ except that β-redexes along the path are reduced. This means that $t \to_\beta^* t{\downarrow_\beta}[u]_p \iff t \to_\beta^* t[u]_{p/\beta}$. Also note that $t|_{p\cdot q} \equiv t|_p|_q$.

In order for a pattern to match in a reduct of t we require that the reductions necessary to assemble its (strict) subterms in the right positions have all been performed. However, not all reductions that can be done need be done, and if we identify the *horizon* of the match — the outermost point at which a substitution contributes to the atoms comprising the match — then we can ignore all reductions beyond that point.

Definition 14. Two paths p and q are *compatible* if there exists a third path r such that both p and q are prefixes of r. The *horizon* of a β-traversal α is the supremum of the lengths of the paths compatible with all elements of the set $reach(\alpha)$:

$$\begin{aligned} reach(\top) &= \{path_1(\top, \epsilon)\} \\ reach(@_l(\alpha)) &= \{path_1(@_l(\alpha), \epsilon)\} \cup reach(\alpha) \\ reach(@_r(\alpha)) &= \{path_1(@_r(\alpha), \epsilon)\} \\ reach(\Lambda(\alpha)) &= \{path_1(\Lambda(\alpha), \epsilon)\} \cup reach(\alpha) \\ reach(\mathrm{B}(\alpha)) &= reach(\alpha) \\ reach(\Sigma(\alpha, \beta)) &= reach(\alpha) \cup reach(\beta) \end{aligned}$$

If all paths in the set are compatible with one another then the set and the length will both be infinite. Thus the horizon is a member of the set of "tropical natural numbers" $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$.

The horizon for the match $t[\hat{\theta}(u)]_{p/\beta}$ of a pattern u is then the minimal horizon of all β-traversals corresponding to non-free-variable atoms in u, treating the head of the match as having the horizon ∞ (since it does not need to be in any particular position for the pattern to match).

Theorem 1. *If the β-traversal α over $t \to_\beta^* t{\downarrow_\beta}[\hat{\theta}(u)]_{p\cdot q}$ has the horizon $|p|$, and $\bar{p}$ is the static path $path_2(\alpha)$, then,*

$$t \to_\beta^* t[\hat{\theta}(u)]_{(p\cdot q)/\beta} \implies t \to_\beta^* t[\hat{\theta}'(u)]_{\bar{p}\cdot(q/\beta)}$$

With this in mind, given a rewrite rule $\mathcal{R} = \langle l, r\rangle$, we see that,

$$\begin{aligned} & t[\hat{\theta}(l)]_{(p\cdot q)/\beta} \to_\mathcal{R} t[\hat{\theta}(r)]_{(p\cdot q)/\beta} \\ \implies & t[\hat{\theta}'(l)]_{\bar{p}\cdot(q/\beta)} \to_\mathcal{R} t[\hat{\theta}'(r)]_{\bar{p}\cdot(q/\beta)} \end{aligned}$$

What is needed is a process by which to calculate the values of $\bar{p}$ and q for a given β-traversal, requiring an alteration to the Kσ-machine. We then need to be able to, given these values, reveal the match so that it may be rewritten; this requires one final alteration to the machine.

4.2 K$\sigma^\ell_\Uparrow$-machine

A variant of the Kσ-machine is the K$\sigma^\ell_\Uparrow$-machine, in which each thunk is labelled with (an algebraic interpretation of) its β-traversal, which is computed as the machine runs. As it stands, introducing labels into the Kσ-machine reveals a potential problem in the σ-calculus: when a substitution passes under a lambda, its variable is not left untouched, but is rather replaced with a new variable constructed from whole cloth and given the appropriate De Bruijn index. Variables in the $\lambda\sigma$-calculus are nothing but their index, so no troubles arise, but if this is not the case — such as in our new

Figure 3. K$\sigma^{\ell}_{\Uparrow}$-machine

$$\langle (t_1 t_2)[\sigma]^{\alpha} \mid C\rangle \to \langle t_1[\sigma]^{@_l(\alpha)} \mid \Box(t_2[\sigma]^{@_r(\alpha)})\,;C\rangle \quad \text{Left}$$
$$\langle (\lambda t)[\sigma]^{\alpha} \mid \Box(u[\rho]^{\beta})\,;C\rangle \to \langle t[(u[\rho]^{\beta}\cdot\sigma)\,;\mathrm{id}]^{\mathrm{B}(\Lambda(\alpha))} \mid C\rangle \quad \text{Beta}$$
$$\langle (\lambda t)[\sigma]^{\alpha} \mid C\rangle \to \langle t[\Uparrow(\sigma)\,;\mathrm{id}]^{\Lambda(\alpha)} \mid \lambda\Box\,;C\rangle \quad \text{Lambda}$$
$$\langle \underline{n}[(\pi\,;\rho)\,;\sigma]^{\alpha} \mid C\rangle \to \langle \underline{n}[\pi\,;(\rho\,;\sigma)]^{\alpha} \mid C\rangle \quad \text{Associate}$$
$$\langle \underline{0}[(u[\pi]^{\beta}\cdot\rho)\,;\sigma]^{\alpha} \mid C\rangle \to \langle u[\pi\,;\sigma]^{\Sigma(\alpha,\beta)} \mid C\rangle \quad \text{Head}$$
$$\langle \underline{n{+}1}[(u[\pi]^{\beta}\cdot\rho)\,;\sigma]^{\alpha} \mid C\rangle \to \langle \underline{n}[\rho\,;\sigma]^{\alpha} \mid C\rangle \quad \text{Tail}$$
$$\langle \underline{0}[\Uparrow(\rho)\,;\sigma]^{\alpha} \mid C\rangle \to \langle \underline{0}[\sigma]^{\alpha} \mid C\rangle \quad \text{Naught}$$
$$\langle \underline{n{+}1}[\Uparrow(\rho)\,;\sigma]^{\alpha} \mid C\rangle \to \langle \underline{n}[(\rho\,;\uparrow)\,;\sigma]^{\alpha} \mid C\rangle \quad \text{Lift}$$
$$\langle \underline{n}[\uparrow\,;\sigma]^{\alpha} \mid C\rangle \to \langle \underline{n{+}1}[\sigma]^{\alpha} \mid C\rangle \quad \text{Shift}$$
$$\langle \underline{n}[\mathrm{id}\,;\sigma]^{\alpha} \mid C\rangle \to \langle \underline{n}[\sigma]^{\alpha} \mid C\rangle \quad \text{Id}$$

machine, where a closure has a label — then a variable with one label is replaced with one with another label, which is unsound. In order to fix this, we will introduce the *lift* operator from the $\lambda\sigma_{\Uparrow}$-calculus of Hardin, et al. [4]. The $\sigma_{\Uparrow}$-substitution $\Uparrow(\sigma)$ is equivalent to the σ-substitution $\underline{0}\cdot(\sigma\,;\uparrow)$, except that it allows us to genuinely not substitute the variable, and so its label is also left unchanged.

The K$\sigma^{\ell}_{\Uparrow}$-machine is defined in Figure 3. If the term structure may contain $\sigma_{\Uparrow}$-closures, those closures' substitutions are unlabelled and have to be mapped into the labelled term space; they cannot just be composed with a labelled substitution. This process is fairly straightforward, mirroring the standard traversal over the term structure, but it is tedious so we shall not discuss it here.

Definition 15. A *cord* is a structure of type $\mathcal{P} \times \mathrm{List}(\mathcal{P}) \times \mathbb{N}^{\infty}$. It acts as a map from steps along the path $p \cdot q$ of a β-traversal, to their static path counterparts, together with the traversal's horizon. The cord may be *split* into a pair of paths corresponding to $\bar{p}$ and q.

$$\begin{aligned}
\mathit{split}(\langle q, ps, \infty\rangle) &= \langle \epsilon, q\rangle \\
\mathit{split}(\langle q, \bar{p} :: ps, 0\rangle) &= \langle \bar{p}, \epsilon\rangle \\
\mathit{split}(\langle F :: q, _ :: ps, k{+}1\rangle) &= \langle \bar{p}, F :: q'\rangle \\
&\quad \text{where } \langle \bar{p}, q'\rangle = \mathit{split}(\langle q, ps, k\rangle)
\end{aligned}$$

By running the K$\sigma^{\ell}_{\Uparrow}$-machine with the algebraic interpretation of β-traversals described in Figure 4, when the machine halts we are left with the cord belonging to an atom of the reduct. When recursing over thunks in the context, we must re-initialise each cord label $\langle q, ps, k\rangle$ to $\langle q, ps, \infty\rangle$, as the horizon of an atom is always relative to the head of the spine to which it belongs.

If in the reduct we find a match for a pattern l (the method for which is outside the scope of this paper, though any higher-order matching algorithm should do), we collect together the cords of all atoms corresponding to non-free-variables in the matching pattern. We can then calculate the cord for the whole match, $\langle q, ps, k\rangle$ where q and ps are from the cord for head of the match and k is the horizon of the match, calculated as described in Definition 14. By splitting this cord we get the paths $\bar{p}$ and q for Theorem 1.

At this point we will not yet have modified the term structure in any way, but rather 'peeked' into the term's β-normal form in order to find a term matching a pattern to reduce. What we need to do now is to take the paths corresponding to that match and with them compute $t \to^{*}_{\beta} t[\hat{\theta}'(l)]_{\bar{p}\cdot(q/\beta)}$.

4.3 K$\sigma^{\sharp}_{\Uparrow}$-machine

The final K$\sigma^{\sharp}_{\Uparrow}$-machine is a simple extension of the Kσ-machine (albeit with the *lift* operator to bring it in line with the calculus of the K$\sigma^{\ell}_{\Uparrow}$-machine), taking a 'track' as an extra component. This track is a path telling the machine the position of the subterm we wish to evaluate to. The K$\sigma^{\sharp}_{\Uparrow}$-machine is described in Figure 5. As an example, we might initialise the machine with the configuration $\langle t|_{\bar{p}}[\mathrm{id}] \mid \Box\rangle^{q}$. Running this machine until it halts will result in a configuration $\langle t'[\sigma] \mid C\rangle^{\epsilon}$, with the components $t'[\sigma] \equiv t|_{\bar{p}\cdot(q/\beta)}$ and $C \equiv t|_{\bar{p}}[\Box]_{q/\beta}$.

If we intend to perform a rewrite step we may now replace the matching $\hat{\theta}'(l)$ with its contractum $\hat{\theta}'(r)$ at this location, as is conventionally done at β-normal form. Assembling these components into the term $t[C[\hat{\theta}'(r)]]_{\bar{p}}$ then completes the full rewrite step.

5. Conclusion

We have introduced an abstract machine, the Kσ-machine, for evaluating terms of the λ-calculus. Building on this, we have introduced a pair of variants, K$\sigma^{\ell}_{\Uparrow}$ and K$\sigma^{\sharp}_{\Uparrow}$, which together enable higher-order term sharing by 'peeking' into a term's normal form to find a subterm matching a pattern, and then reducing enough of the term to reveal the match for rewriting without fully normalising the term. In this way the term sharing described by the simply-typed λ-calculus is made use of, instead of being lost by normalisation, or otherwise restricted to sharing with dags, which are equivalent to first-order β-redexes alone.

The value of the machinery being used here is that, as it in no way modifies the term structure itself, it does not reduce the term as it travels, but instead navigates through the unmodified term by building up a substitution environment. This parallels common approaches to sharing first-order terms with dags, in which a redex is found 'modulo sharing' and then the necessary components are unshared to make the rewrite possible. However, the simply-typed λ-calculus provides more sophisticated sharing than dags, and we make steps towards its treatment as a mechanism for 'higher-order term sharing'.

The way we make use of the sharing of the λ-calculus here is, however, relatively primitive. Although it is necessary to determine the horizon of a match, as the K$\sigma^{\ell}_{\Uparrow}$-machine does, in order to avoid rewriting in one location when it could be done in multiple, sharing information is still lost when subsequent β-reductions are performed by the K$\sigma^{\sharp}_{\Uparrow}$-machine. This is not about optimality *à la* Lévy [7], but about managing higher-order sharing in the same

Figure 4. An algebraic interpretation of a β-traversal

$$\begin{aligned}
\top &= \langle \epsilon, \epsilon :: \epsilon, \infty \rangle && \text{Top} \\
\mathrm{B}(\langle \alpha, ps, k \rangle) &= \langle \alpha, ps, k \rangle && \text{Beta} \\
\Sigma(\langle \alpha, p :: ps, k \rangle, \langle @_r :: \beta, q :: qs, k' \rangle) &= \langle \alpha, q :: ps, \mathit{min}(k, |qs|) \rangle && \text{Substitute} \\
@_l(\langle \alpha, p :: ps, k \rangle) &= \langle @_l :: \alpha, (@_l :: p) :: p :: ps, k \rangle && \text{Left} \\
@_r(\langle \alpha, p :: ps, k \rangle) &= \langle @_r :: \alpha, (@_r :: p) :: p :: ps, k \rangle && \text{Right} \\
\Lambda(\langle \alpha, p :: ps, k \rangle) &= \langle \Lambda :: \alpha, (\Lambda :: p) :: p :: ps, k \rangle && \text{Lambda}
\end{aligned}$$

Figure 5. K$\sigma^{\sharp}_{\Uparrow}$-machine

$$\begin{aligned}
\langle (t_1 t_2)[\sigma] \mid C \rangle^{@_l :: \alpha} &\to \langle t_1[\sigma] \mid \Box(t_2[\sigma]) \,;\, C \rangle^{\alpha} && \text{Left} \\
\langle (t_1 t_2)[\sigma] \mid C \rangle^{@_r :: \alpha} &\to \langle t_2[\sigma] \mid (t_1[\sigma])\Box \,;\, C \rangle^{\alpha} && \text{Right} \\
\langle (\lambda t)[\sigma] \mid \Box(u[\rho]) \,;\, C \rangle^{\Lambda :: \alpha} &\to \langle t[(u[\rho] \cdot \sigma) \,;\, \mathrm{id}] \mid C \rangle^{\alpha} && \text{Beta} \\
\langle (\lambda t)[\sigma] \mid C \rangle^{\Lambda :: \alpha} &\to \langle t[\Uparrow(\sigma) \,;\, \mathrm{id}] \mid \lambda\Box \,;\, C \rangle^{\alpha} && \text{Lambda} \\
\langle \underline{n}[(\pi \,;\, \rho) \,;\, \sigma] \mid C \rangle^{\alpha} &\to \langle \underline{n}[\pi \,;\, (\rho \,;\, \sigma)] \mid C \rangle^{\alpha} && \text{Associate} \\
\langle \underline{0}[(u[\pi] \cdot \rho) \,;\, \sigma] \mid C \rangle^{\alpha} &\to \langle u[\pi \,;\, \sigma] \mid C \rangle^{\alpha} && \text{Head} \\
\langle \underline{n{+}1}[(u[\pi] \cdot \rho) \,;\, \sigma] \mid C \rangle^{\alpha} &\to \langle \underline{n}[\rho \,;\, \sigma] \mid C \rangle^{\alpha} && \text{Tail} \\
\langle \underline{0}[\Uparrow(\rho) \,;\, \sigma] \mid C \rangle^{\alpha} &\to \langle \underline{0}[\sigma] \mid C \rangle^{\alpha} && \text{Naught} \\
\langle \underline{n{+}1}[\Uparrow(\rho) \,;\, \sigma] \mid C \rangle^{\alpha} &\to \langle \underline{n}[(\rho \,;\, \uparrow) \,;\, \sigma] \mid C \rangle^{\alpha} && \text{Lift} \\
\langle \underline{n}[\uparrow \,;\, \sigma] \mid C \rangle^{\alpha} &\to \langle \underline{n{+}1}[\sigma] \mid C \rangle^{\alpha} && \text{Shift} \\
\langle \underline{n}[\mathrm{id} \,;\, \sigma] \mid C \rangle^{\alpha} &\to \langle \underline{n}[\sigma] \mid C \rangle^{\alpha} && \text{Id}
\end{aligned}$$

fashion as Wadsworth [11] does first-order sharing. By comparing the behaviour of Wadsworth's dags, and first-order β-redexes in our sharing scheme, we can see that there is still work to be done on this front. This is the subject of ongoing research.

References

[1] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(04):375–416, 1991.

[2] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North Holland, 1984.

[3] Nicolaas de Bruijn. Lambda calculus notation with nameless dummies: a tool for automatic formula manipulation, with application to the Church–Rosser theorem. In *Indagationes Mathematicae*, volume 75, pages 381–392, 1972.

[4] Thérèse Hardin and Jean-Jacques Lévy. A confluent calculus of substitutions. In *France–Japan Artificial Intelligence and Computer Science Symposium*, volume 106, 1989.

[5] Gérard Huet. Functional pearl: The zipper. *Journal of Functional Programming*, 7(05):549–554, 1997.

[6] Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

[7] Jean-Jacques Lévy. Optimal reductions in the lambda-calculus. *To HB Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 159–191, 1980.

[8] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[9] Tobias Nipkow. Higher-order critical pairs. In *Proceedings of the 6th Annual IEEE Symposium of Logic in Computer Science*, 1991.

[10] Vincent van Oostrom. *Confluence for abstract and higher-order rewriting*. PhD thesis, Vrije Universiteit, 1994.

[11] Christopher Wadsworth. *Semantics and pragmatics of the lambda calculus*. PhD thesis, University of Oxford, 1971.