Normalization by Evaluation for Typed Weak λ-Reduction

Filippo Sestini
Functional Programming Laboratory, University of Nottingham, United Kingdom
http://www.cs.nott.ac.uk/~psxfs5
filippo.sestini@nottingham.ac.uk

Abstract

Weak reduction relations in the λ-calculus are characterized by the rejection of the so-called ξ-rule, which allows arbitrary reductions under abstractions. A notable instance of weak reduction can be found in the literature under the name *restricted reduction* or *weak λ-reduction*.

In this work, we attack the problem of algorithmically computing normal forms for λ^{wk}, the λ-calculus with weak λ-reduction. We do so by first contrasting it with other weak systems, arguing that their notion of reduction is not strong enough to compute λ^{wk}-normal forms. We observe that some aspects of weak λ-reduction prevent us from normalizing λ^{wk} directly, thus devise a new, better-behaved weak calculus λ^{ex}, and reduce the normalization problem for λ^{w} to that of λ^{ex}. We finally define type systems for both calculi and apply Normalization by Evaluation to λ^{ex}, obtaining a normalization proof for λ^{wk} as a corollary. We formalize all our results in Agda, a proof-assistant based on intensional Martin-Löf Type Theory.

2012 ACM Subject Classification Theory of computation → Lambda calculus

Keywords and phrases normalization, lambda-calculus, reduction, term-rewriting, Agda

Digital Object Identifier 10.4230/LIPIcs.TYPES.2018.6

Supplement Material https://github.com/fsestini/nbe-weak-stlc

Acknowledgements We thank Maria Emilia Maietti, Claudio Sacerdoti Coen, Thorsten Altenkirch, Andreas Abel, and Delia Kesner for helpful discussions and feedback on this work.

1 Introduction

The weak λ-calculus can be described informally as the $\lambda\beta$-calculus without the ξ rule – the congruence rule for λ-abstractions – shown below for the simply-typed case. Without any other modifications, this system is not *confluent* (or Church-Rosser). The property can be recovered with the addition of a substitution rule, labeled (σ) below, which gives rise to a confluent system.

$$\frac{\Gamma, x : A \vdash t \longrightarrow s : B}{\Gamma \vdash \lambda x.t \longrightarrow \lambda x.s : A \to B}\ (\xi) \qquad \frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash a \longrightarrow b : A}{\Gamma \vdash t[a/x] \longrightarrow t[b/x] : B}\ (\sigma)$$

This particular notion of weak reduction was originally formulated by Howard [25], although presented differently, and was later studied by Çağman and Hindley [13] under the name of *weak λ-reduction*. On the one hand, the addition of (σ) restores confluence, but on the other hand it complicates the design of an algorithmic procedure to mechanically compute terms to normal form: despite the absence of the ξ-rule, conversion under abstractions is still provable via the σ-rule for a restricted class of redexes, the so called *weak redexes*. That is, contrary to the non-confluent weak λ-calculus, contractions *can* occur under λ-abstractions. This makes weak λ-reduction *crucially* different from, and more complicated than other weak calculi that implement weak-*head* reduction instead, for which a normalization algorithm never needs to reduce under binders.

These issues must be faced when attempting to prove the normalization theorem for a typed version of the weak λ-calculus, a goal that motivated the work presented here. Indeed, suppose we want to show that every well-typed term t reduces to its normal form, given by a normalization function $[\![t]\!]_\rho$ defined by structural recursion, and in particular such that $[\![\lambda x.t]\!]_\rho = \lambda x.[\![t]\!]_{\rho[x/x]}$. In the case of a term $\Gamma \vdash \lambda x.t : A \Rightarrow B$, we can obtain $\Gamma, x : A \vdash t \longrightarrow [\![t]\!]_{\rho[x/x]} : B$ by induction hypothesis, but then it is not clear how to proceed, since we cannot in general conclude $\Gamma \vdash \lambda x.t \longrightarrow \lambda x.[\![t]\!]_{\rho[x/x]} : A \Rightarrow B$ without ξ.

Another unpleasant aspect of weak λ-reduction is its *relative* notion of redex. Consider the term $t \equiv \lambda z.(\lambda x.y)\ z$. Its subterm $(\lambda x.y)\ z$ is a valid redex under standard β reduction, leading to the reduction $\lambda z.(\lambda x.y)\ z \longrightarrow \lambda z.y$, but is not a valid redex *of* t in the weak λ-calculus, since there is no way to express this reduction in terms of (β) and (σ). The same term $(\lambda x.y)\ z$, instead, does reduce as a subterm of $\lambda w.(\lambda x.y)\ z$, leading to $\lambda w.y$. This relative notion of redex complicates the definition of a normalization algorithm, since it is not clear from the syntactic structure alone whether or not a redex can be justified by the substitution rule and should be contracted.

1.1 Contributions

In this paper we study weak λ-reduction, and propose a way to algorithmically reduce terms to their normal form. We also give a constructive proof of normalization for a simply-typed λ-calculus with weak λ-reduction. More precisely, we make the following contributions:

- We compare the λ-calculus with weak λ-reduction, that we call λ^{wk}, with other *weak* calculi rejecting the ξ-rule. We precisely characterize λ^{wk} normal forms, compare them with the other systems, and show that the problem of normalizing weak λ-reduction does not seem to be reducible to normalization of these calculi, identifying what we think is an unexplored area in the literature;
- We define a new calculus of weak reductions, λ^{ex}, that is inspired by our characterization of λ^{wk} redexes and in particular makes them syntactically explicit (hence the name). This system recovers a version of the ξ-rule, and admits a standard normalization algorithm. We then show λ^{wk} and λ^{ex} equivalent, thus reducing the problem of normalization for the former to normalization for the latter. This provides the first published specification of a normalization algorithm for λ^{wk} in full detail;
- We define type systems for λ^{wk} and λ^{ex}, and prove the latter normalizing via the semantic method of Normalization by Evaluation [11]. Mirroring the untyped case, the two calculi are shown equivalent. Transporting the normalization proof for λ^{ex} along this equivalence yields, as far as we are aware, the first proof of normalization for the simply-typed λ-calculus with weak λ-reduction;
- We include a full formalization of this work in intensional Type Theory, using the Agda proof-assistant [12] [1].

1.2 Meta-theory and notation

We will use a rather informal and foundation-agnostic notation, that can be understood both in Type Theory and in constructive set theory. We do however distinguish between definitional equalities ($\equiv$), which are always decidable, and propositional equalities ($=$), for which decidability needs to be proved.

[1] The formalization can be found at `https://github.com/fsestini/nbe-weak-stlc`

▶ Remark 1. Readers interested in a proof-checked formalization of this work in Type Theory are invited to consult the Agda code, keeping in mind the following differences:

- The partial functions represented here with the usual function notation cannot be defined in Agda as standard type-theoretic functions, since these must be total in order to preserve logical consistency. In the formalization, partial functions are encoded by their graph, i.e. as inductively-defined functional (left-total, right-unique) relations;
- In the Agda code, we use a nameless [20] syntax to represent λ-terms.

2 Weak λ-reduction

The weak λ-calculus is generally defined as a flavor of the λ-calculus whose reduction relation does not include the weak extensionality principle represented by the ξ-rule [9], also referred to as the congruence rule for λ-abstractions. One of the simplest forms of weak reduction is obtained by stripping β-reduction of the ξ-rule.

▶ **Definition 2** (Weak reduction). *Weak reduction is inductively defined as follows:*

$$\frac{}{(\lambda x.t)s \longrightarrow t[s/x]}\ (\beta) \qquad \frac{t \longrightarrow r}{t\ s \longrightarrow r\ s}\ (\nu) \qquad \frac{s \longrightarrow r}{t\ s \longrightarrow t\ r}\ (\mu)$$

Weak reduction, also known as *weak-head reduction*, is of interest in the study of programming languages, as it captures the fact that evaluation of programs does not generally proceed under binders [23]. Weak reduction evaluates terms to *weak-head* normal form (*whnf*):

$$\mathsf{Whnf} \ni d ::= \lambda x.t \mid x d_1 d_2 ... d_n$$

Weak reduction never reduces under binders, and as a consequence the body t of a weak-head normal abstraction may be an arbitrary term, not necessarily a whnf. Unfortunately, this relation is not confluent, hence only specific weak reduction strategies have been studied in the literature [36, 4]. A solution to this problem consists of extending weak reduction with a primitive substitution rule [31], the σ-rule (Section 1).

One of the first uses of this confluent variant of weak reduction dates back to Howard [25], who calls it *restricted reduction*. Here we will refer to it as *weak λ-reduction* after Çağman and Hindley [13].

▶ **Definition 3** (Weak λ-reduction). *The reduction relation $\longrightarrow_w$ is defined as the relation in Definition 2 plus the σ-rule.*

▶ **Theorem 4.** *$\longrightarrow_w^*$, the reflexive-transitive closure of $\longrightarrow_w$, is confluent.*

Proof. See [31], Theorem 1. ◀

In [13], the authors cite an alternative formulation of weak λ-reduction based on the notion of *weak redex*, due to Howard:

▶ **Definition 5.** *Let the redex r be a subterm of a term t. Then, r is a* weak redex *if and only if it does not contain free variables that are bound in t. A one-step weak λ-contraction of t is one that contracts a weak redex inside t.*

For example, the term $\lambda x.(\lambda y.y)z$ contains the weak redex $(\lambda y.y)z$. Conversely, the term $\lambda x.(\lambda y.x)z$ has no weak redexes, and is therefore in normal form. These two definitions give rise to equivalent relations [13].

We call λ^{wk} the λ-calculus with weak λ-reduction, as given in Definition 3, and refer to its normal forms as *weak normal forms*. These are not characterized as easily as whnfs, and the reason is that being a weak normal form is a relative property, since being a weak redex is, and normal terms are just terms with no weak redexes. We observe that the essence of weak redexes can be reduced to the distinction between two *roles* for variables. Given a term t, we say that a variable has *local* role if it is bound somewhere within t, and *global* otherwise. Then, a weak redex is just a redex that is closed w.r.t. local variables (cfr. Definition 5.) More precisely, if $t \equiv C[r]$ for an *enclosing context* $C[\_]$ with a hole and a redex r, then r is a weak redex iff it is closed w.r.t. the local variables bound by abstractions within $C[\_]$. This suggests a notion of normal form that is indexed by the set V of local variables of the enclosing context, whatever this is. For convenience, we define normal terms Nf^V mutually with *neutral terms* Ne^V, both under a set of variables V:

$$\mathsf{Nf}^V \ni d^V ::= e^V \mid \lambda x.d^{V\cup\{x\}}$$

$$\mathsf{Ne}^V \ni e^V ::= x \mid e^V\, d^V \mid (\lambda x.d_1^{V\cup\{x\}})\, d_2^V \text{ s.t. } FV((\lambda x.d_1)d_2) \cap V \neq \emptyset$$

Neutral terms usually correspond to variables and elimination forms whose reduction is "blocked" by the presence of a neutral term in recursive position. In the standard $\lambda\beta$-calculus, neutral terms are only variables and "stuck" applications. In our setting, the definition of neutral term needs to be extended, to account for redexes of the form $(\lambda x.t)s$ that are not *weak*, and therefore "stuck" as well. Given a set V of local variables, this is the case whenever some variables in V are free in the redex, namely $FV((\lambda x.t)s) \cap V \neq \emptyset$.

We define the set of weak normal forms $\mathsf{Nf} :\equiv \mathsf{Nf}^\emptyset$. As an example, we can see that $\lambda xy.(\lambda z.x)w \in \mathsf{Nf}$, since $(\lambda z.x)w \in \mathsf{Ne}^{\{x,y\}}$, but $\lambda x.(\lambda y.y)z \notin \mathsf{Nf}$, since $FV((\lambda y.y)z) \equiv \{z\} \cap \{x\} = \emptyset$.

2.1 Algorithmic weak λ-reduction

Although typed β-reduction is normalizing [24], and we know that weak λ-reduction constitutes a (strict) subset of full β-reduction, these facts alone do not provide us with an algorithm to actually compute normal forms, or even ensure that such algorithm exists.

Here, we are interested in computational solutions to the problem of algorithmic reduction for (typed) calculi based on weak λ-reduction. A way to achieve this goal is to seek a *constructive* proof of the normalization theorem. When formalized in Intuitionistic Type Theory, this proof comes by its nature with a normalization algorithm for untyped terms baked-in. This is quite convenient, but it also means that we cannot hope to proceed any further without some understanding of how to go by implementing such algorithm. The presence of the substitution rule (σ) interacts with this goal in unpleasant ways:

- Despite the absence of the ξ-rule, *some* contractions can still happen under binders via (σ). A normalization algorithm will thus have to proceed, in some way, by recursion on the structure of terms and thus under λs; but this is at odds with weak λ-reduction not being a congruence relation, which prevents us from reasoning about these recursive reductions in an adequate way.
- Weak λ-reduction has a relative notion of redex, defined w.r.t. some term t that contains it as a subterm (Definition 5). What makes a redex *weak* is therefore not evident from its syntactic structure alone, so a standard normalization function that just proceeds by structural recursion does not seem sufficient.

Perhaps because of these difficulties, or perhaps because of the exotic nature of weak λ-reduction, we could not find a complete specification of a normalization algorithm for this notion of reduction anywhere in the literature, nor a proof of normalization for a typed calculus based on it. A first approach towards filling this gap is to try to reduce the problem of normalization for λ^{wk} to normalization of other weak calculi for which a solution is well-known. We attempt to do so in the next sections.

2.2 Combinatory logic

There is a correspondence between λ^{wk} and combinatory logic (CL) [18], made precise in [13] by means of two operations $\_\lambda$ and $\_H$ respectively translating CL terms to λ-terms and vice versa. $\_\lambda$ is the obvious translation of combinators to λ-terms, whereas $\_H$ is essentially combinator (or *bracket*) abstraction ([24], Definition 2.18). We can then show

▶ **Theorem 6.** *For all combinators c, d and λ-terms t, s:*

- $c \rhd_w d \implies c_\lambda \longrightarrow_w d_\lambda$;
- $t \longrightarrow_w s \implies t_H \rhd_w s_H$.

Proof. See [13]. ◀

Here $\rhd_w$ is combinator reduction. Normalization for combinatory logic is well understood, and in particular Normalization by Evaluation has been successfully applied both to the typed [16] and the untyped [21] cases. Since our goal is to algorithmically reduce λ-terms to weak normal form, we may hope to be able to reduce the problem to that of normalizing combinators, by exploiting the correspondence stated in Theorem 6.

Unfortunately, normal forms are not correctly related by the two translation operations. A counter-example is given by the λ-term $t \equiv \lambda x.(\lambda y.x)(II)$, which has a weak normal form $\lambda x.(\lambda y.x)I$ (where I is the identity). The term t translates to $t_H \equiv SK(K(II))$, and its normal form $SK(KI)$ translates back to the λ-term $\lambda w.((\lambda xy.x)w)((\lambda xy.x)Iw)$. But this term is neither normal (since $(\lambda xy.x)I$ is a weak redex) nor convertible to $\lambda x.(\lambda y.x)I$ in the absence of ξ.

This mismatch is also observed in [40], where a version of Martin-Löf Type Theory is compared to a combinator-based formulation. We are not aware of a way to relate CL- and λ-terms that makes it possible to rely on combinatory reduction to fully compute weak λ-reduction.

2.3 Weak explicit substitutions

Explicit substitutions are a way to formulate the syntax and reduction rules of the λ-calculus that turns substitutions into constructors of the syntax of terms, and integrates the substitution operation as part of the reduction relation, rather than an implicit meta-theoretic operation [1]. There have been several attempts at modeling weak reduction with explicit substitutions [16, 17, 31, 7]. In that setting, the *weak* character of weak reduction can be captured by stipulating that substitutions should not be propagated under binders. We describe some attempts at doing this, starting with the weak λ-calculus by Lévy and Maranget [31], whose substitution mechanism is a hybrid between implicit and explicit:

$$\mathsf{Term} \ni t, s ::= x \mid ts \mid (\lambda x.p)[\sigma] \qquad \mathsf{Prog} \ni p, q ::= x \mid pq \mid \lambda x.p$$
$$\mathsf{Subst} \ni \sigma ::= (x_1, t_1), (x_2, t_2), \ldots, (x_n, t_n)$$

Here, we use the metavariables p, q for *programs*, i.e. constant terms, and t, s for ordinary *terms*. Explicit substitutions σ are just lists of variable-term pairs. We write $\sigma[t/x]$ for the extended substitution that maps x to t and behaves like σ otherwise. Terms are formed out of variables, application, and *closures* $(\lambda x.p)[\sigma]$, i.e. pairs made of a functional program $\lambda x.p$ and a substitution σ assigning terms to its free variables. However, we do have implicit substitution $\langle \_ \rangle$ on programs:

$$x\,\langle\sigma\rangle :\equiv \sigma(x) \qquad (p\ q)\langle\sigma\rangle :\equiv p\langle\sigma\rangle\ q\langle\sigma\rangle \qquad (\lambda x.p)\langle\sigma\rangle :\equiv (\lambda x.p)[\sigma]$$

Here $\sigma(x)$ just looks up the term associated to the variable x in a substitution. The dynamics of weak explicit substitutions is defined as follows; closures are used to avoid pushing substitutions under λ-abstractions, since otherwise we would validate the ξ-rule.

▶ **Definition 7** (Weak explicit substitutions)**.**

$$\frac{}{(\lambda x.p)[\sigma]\ s \longrightarrow p\langle\sigma[s/x]\rangle}\ (\beta) \qquad \frac{\sigma \longrightarrow \sigma'}{(\lambda x.p)[\sigma] \longrightarrow (\lambda x.p)[\sigma']}\ (\xi\text{-}subst)$$

$$\frac{t \longrightarrow t'}{ts \longrightarrow t's}\ (\nu) \qquad \frac{s \longrightarrow s'}{ts \longrightarrow ts'}\ (\mu) \qquad \frac{t \longrightarrow s}{\sigma, (x_i, t), \sigma' \longrightarrow \sigma, (x_i, s), \sigma'}$$

A normal form is thus either a variable applied to normal forms, or a functional program together with a normal substitution:

$$\mathsf{Nf} \ni d ::= x\,d_1 \ldots d_n \mid (\lambda x.p)[\rho] \qquad \mathsf{Nfs} \ni \rho ::= (x_1, d_1), (x_2, d_2), \ldots, (x_n, d_n) \tag{1}$$

A related calculus was defined by Martin-Löf in one of the early formulations of his type theory [35]. In that system, terms of function type are not constructed by abstraction, but by introducing a fresh symbol for a combinator. The system includes a primitive substitution rule like the σ-rule in Definition 3, but since functions are just atomic symbols, substitution *de facto* never happens under binders. We do not reproduce Martin-Löf's system here, but instead consider an alternative formulation due to Coquand and Dybjer [16], which is completely equivalent for the purpose of our analysis. The system is based on explicit substitutions and is very similar to that of Definition 7, although they present it using a typed nameless syntax [20]:

$$\mathsf{Term} \ni t, s ::= x \mid ts \mid (\lambda x.t)[\sigma] \qquad \mathsf{Subst} \ni \sigma ::= (x_1, t_1), \ldots, (x_n, t_n)$$

The normal forms can be characterized in the same way as (1), with the only difference that now λ-abstractions in normal form $(\lambda x.t)[\rho]$ have ordinary terms as their body. A further generalization is obtained by allowing explicit substitutions on any term instead of just on functional closures. An example is the *weak $\lambda\sigma$-calculus* of [17], which we will call $\lambda^w\sigma$, and that is also considered in its typed nameless version in [16].

$$\mathsf{Term} \ni t, s ::= x \mid ts \mid \lambda x.t \mid t[\sigma] \qquad \mathsf{Subst} \ni \sigma ::= \langle\rangle \mid (x, t), \sigma \mid \sigma_1 \cdot \sigma_2$$

Here Subst includes an empty substitution $\langle\rangle$, and a composition constructor $\sigma_1 \cdot \sigma_2$, which is used to model terms under multiple substitutions: $t[\sigma_1][\sigma_2] \longrightarrow t[\sigma_1 \cdot \sigma_2]$. Reduction is essentially that of Definition 7, apart from the β-rule which is now $(\lambda x.t)[\sigma]s \longrightarrow t[(x, s), \sigma]$, and the addition of reduction rules for explicit substitutions. Normal forms are characterized exactly as in Martin-Löf's weak λ-calculus just described, namely

$$\mathsf{Nf} \ni d ::= x\,d_1 \ldots d_n \mid (\lambda x.t)[\rho] \qquad \mathsf{Nfs} \ni \rho ::= (x_1, d_1), (x_2, d_2), \ldots, (x_n, d_n)$$

These calculi of weak explicit substitutions are very similar, and in particular they all implement some form of *weak-head* reduction, where computation does not occur under binders. Weak-head normalization is relatively simple and well-understood, and both [35] and [16] include proofs of normalization for their respective systems. We now compare weak explicit substitutions to λ^{wk}, particularly to see whether these can be used to compute λ^{wk}-normal forms. We consider $\lambda^{w}\sigma$ as a representative of the calculi of weak explicit substitutions that we have presented, as it can simulate the others.

Note that terms of the weak λ-calculus can be embedded into $\lambda^{w}\sigma$, so a naive approach would be to just treat weak λ-terms as terms of $\lambda^{w}\sigma$, and normalize them under this reduction relation. $\lambda^{w}\sigma$ normal forms can be turned back into regular λ-terms by just fully applying explicit substitutions as they were implicit. That this translation fails to achieve our goal is already evident by observing the difference in the normal forms of the two calculi. In particular, every λ-abstraction is a normal form in $\lambda^{w}\sigma$, even when its body is not normal, whereas in the implicit weak calculus, abstractions are only normal if they do not contain weak redexes, that is, $\lambda x.d \in \mathsf{Nf} \iff d \in \mathsf{Nf}^{\{x\}}$.

This does not necessarily settle the question negatively, because there could be a way to translate λ^{wk}-terms to $\lambda^{w}\sigma$-terms in a way that makes this method work. In [31], the authors consider a translation via maximal free subterms. A subterm t' of a term t is free whenever $t \equiv C[t']$ for some context $C[\_]$ that does not bind any free variable in t'. A free subterm is maximal whenever it is not a subterm of another free subterm. We define a translation operation $\mathcal{T}_1$ from λ^{wk} to $\lambda^{w}\sigma$:

$$\mathcal{T}_1(x) = x \qquad \mathcal{T}_1(ts) = \mathcal{T}_1(t)\mathcal{T}_1(s)$$
$$\mathcal{T}_1(\lambda x.t) = (\lambda x.C[x_1, \ldots, x_n])[(x_1, \mathcal{T}_1(t_1)), \ldots, (x_n, \mathcal{T}_1(t_n))]$$

where $t_1, \ldots, t_n$ are the maximal free subterms of t. We also define $\mathcal{T}_2$ as the converse translation that again just applies all explicit substitutions. We can now show that a reduction in λ^{wk} translates to a reduction in $\lambda^{w}\sigma$:

▶ **Proposition 8.** *If* $t \longrightarrow_w \mathcal{T}_2(s)$*, then* $\mathcal{T}_1(t) \longrightarrow s$*.*

Unfortunately, this result does not generalize to the reflexive-transitive closure of reduction, so in particular it fails to relate normal forms in the two calculi as we require. In fact, consider the following reduction of a term t in λ^{wk}:

$$t \equiv (\lambda x.\lambda y.xz)(\lambda w.w) \longrightarrow_w \lambda y.(\lambda w.w)z \longrightarrow_w \lambda y.z$$

On the other hand, we have

$$\mathcal{T}_1(t) \equiv (\lambda x.\lambda y.xk)[z/k](\lambda w.w) \longrightarrow (\lambda y.xk)[z/k, (\lambda w.w)/x] \equiv s$$

The term s is a normal form in $\lambda^{w}\sigma$, but translates to the reducible term $\mathcal{T}_2(s) \equiv \lambda y.(\lambda w.w)z$ in λ^{wk}. The problem is that Proposition 8 only holds for terms that are image of $\mathcal{T}_1$, i.e. those terms t such that their explicit substitutions only contain maximal free subterms in $\mathcal{T}_2(t)$. Unfortunately, β-contraction destroys this maximality property, since it can create new weak redexes in $\mathcal{T}_2(t)$ that do not exist in t. We conjecture that there exists a different pair of translation functions that makes this work, but leave the rigorous investigation of this aspect to future work.

3 Two-variable syntax

As anticipated at the end of Section 2, the notion of a weak redex in a term t is essentially about the distinction between two variable *roles*: the *local* variables that may appear bound

within t, and the *global* variables that may not. Therefore, the only information that is really needed by a normalization algorithm to reduce t to weak normal form is the role of all variables in t. We can exploit this fact by choosing a representation of λ-terms where this variable distinction is made syntactically explicit. Deciding whether a redex in t is weak then becomes a straightforward syntactic check. This is reminiscent of the *locally-nameless* representation of λ-terms [14], where a similar syntactic distinction is made between free and bound variables. We define two-variable λ-terms Term as follows

$$\mathsf{Term} \ni t, s ::= x^G \mid x^L \mid \lambda x.t \mid ts$$

We distinguish between global variables x^G and local variables x^L. Abstraction and application are the usual ones. We define the set of *all* free variables $FV(t)$ for a term t in the usual way ([9], Definition 2.1.7), and consider the obvious restrictions FV^L, FV^G to, respectively, local and global variables. We say that a term is *locally-closed* when it contains no free local variables, and define the set of locally-closed terms as $\mathsf{LC} \equiv \{t \in \mathsf{Term} \mid FV^L(t) = \emptyset\}$.

We consider two substitution operations $\_[\_/\_]$ and $\_\langle\_/\_\rangle$, dedicated respectively to local and global variables.

$$x^G\langle a/y\rangle :\equiv \begin{cases} a, & \text{if } x = y \\ x^G, & \text{otherwise} \end{cases} \qquad x^G[a/y] :\equiv x^G$$

$$x^L\langle a/y\rangle :\equiv x^L \qquad x^L[a/y] :\equiv \begin{cases} a, & \text{if } x = y \\ x^L, & \text{otherwise} \end{cases}$$

$$(\lambda x.t)\langle a/y\rangle :\equiv \lambda x.t\langle a/y\rangle \qquad (\lambda\, t)[a/y] :\equiv \lambda x.t[a/y]$$

$$(t\, s)\langle a/x\rangle :\equiv t\langle a/x\rangle\, s\langle a/x\rangle \qquad (t\, s)[a/y] :\equiv t[a/y]\, s[a/y]$$

We assume α-conversion is applied when needed to avoid variable capture, as well as the Barendregt convention ([9], 2.1.13). The two-variable syntax is instrumental in the definition of an auxiliary reduction relation, given in Definition 10 below, later shown equivalent to weak λ-reduction (Theorem 13). However, to facilitate the proof of this equivalence, we restate weak λ-reduction of Definition 3 in terms of the same two-variable syntax, and use this definition from now on. Since the local vs global variable distinction is irrelevant in this case, we restrict our definition to locally-closed terms. For this class of terms, the global (resp. local) variables end up corresponding to free (resp. bound) variables.

▶ **Definition 9** (Two-variable weak λ-reduction). *The binary relation* $\_ \longrightarrow_w \_$ *between* locally-closed *two-variable terms is inductively defined as follows.*

$$\frac{}{(\lambda x.t)\, s \longrightarrow_w t[s/x]}\ (\beta) \qquad \frac{a \longrightarrow_w b}{t\langle a/x\rangle \longrightarrow_w t\langle b/x\rangle}\ (\sigma)$$

It can be seen that Definition 9 is just weak λ-reduction of Definition 3, modulo decorated variables and congruence rules, which are derivable from (σ). From now on, we will consider λ^{wk} to be the system with LC as terms and Definition 9 as reduction relation.

We now take advantage of the two-variable representation to define a normalization algorithm for the two-variable syntax, that will end up corresponding to normalization under weak λ-reduction. We specify it as a recursive traversal on arbitrary two-variable terms, not necessarily locally-closed. In particular, whenever we recursively reduce under a λ-abstraction, we do *not* replace the previously bound local variable with some free global one, but instead we leave it local. As a consequence, when the algorithm is applied to a locally-closed term $t \equiv C[s]$, recursive calls are able to identify which variables that appear free in a subterm s of t are bound by $C[\_]$, and thus which redexes are weak redexes, *without* having to keep track of what $C[\_]$ is.

A normalization function based on these ideas is shown below. We will use *parallel substitutions* $\rho \in \mathsf{Subst}$ to map free local variables to normal terms. These are defined as either an empty substitution $\langle\rangle$, or the extension $\rho[t/x]$ of ρ with the mapping $x \mapsto t$. We write $\rho(x)$ for the (partial) look-up function for the term associated to a variable, defined recursively in the obvious way, $[t/x]$ for the singleton parallel substitution, and $t[\rho]$ for the repeated application of all substitutions in ρ to a term t. Thus, if t is a locally-closed term, the expression $[\![t]\!]\langle\rangle$ gives the normal form of t, if it exists.

$$
\begin{aligned}
[\![x^G]\!]\,\rho &:\equiv x^G \\
[\![x^L]\!]\,\rho &:\equiv \rho(x) \\
[\![\lambda x.t]\!]\,\rho &:\equiv \lambda x.[\![t]\!]\,(\rho[x^L/x]) \\
[\![t\,s]\!]\,\rho &:\equiv [\![t]\!]\,\rho \bullet [\![s]\!]\,\rho
\end{aligned}
\qquad
t \bullet s :\equiv \begin{cases} [\![t']\!][s/x] & \text{if } t \equiv \lambda x.t' \text{ and } t\,s \in \mathsf{LC} \\ t\,s & \text{otherwise} \end{cases}
$$

Note that this normalization function takes untyped λ-terms as arguments, so it is necessarily *partial*, since not every untyped term admits a normal form under weak λ-reduction. As a consequence of this, in the Agda formalization untyped normalization is not defined as a type-theoretic function, but rather as an inductive functional relation (see remark in Section 1.2).

We can distill the function above into a reduction relation $\longrightarrow_{ex}$, that we call *explicit* because it makes use of the explicitly syntactic distinction between variable roles.

▶ **Definition 10** (Explicit two-variable weak λ-reduction)**.** *The binary relation* $\_ \longrightarrow_{ex} \_$ *between* arbitrary *two-variable terms is defined as follows.*

$$
\frac{(\lambda x.t)\,s \in \mathsf{LC}}{(\lambda x.t)\,s \longrightarrow_{ex} t[s/x]}\ (\beta)
\qquad
\frac{t \longrightarrow_{ex} s}{\lambda x.t \longrightarrow_{ex} \lambda x.s}\ (\xi)
\qquad
\frac{t \longrightarrow_{ex} r}{t\ s \longrightarrow_{ex} r\ s}\ (\nu)
\qquad
\frac{s \longrightarrow_{ex} r}{t\ s \longrightarrow_{ex} t\ r}\ (\mu)
$$

Note that $\longrightarrow_{ex}$ recovers the ξ-rule. However, the resulting relation is still *weak* in the sense of Definition 5 on locally-closed terms, because of the β-rule enforcing that λ-abstractions only bind local variables, and that only locally-closed (i.e. weak) redexes are contracted. The two relations are equivalent on locally-closed terms (Theorem 13.) The proof is adapted from [13] (Proposition 4.6).

▶ **Lemma 11.** *If* $t \longrightarrow_{ex} s$*, then there exist a term* C *with free global variable* x *and* locally-closed *terms* a, b *such that* $a \longrightarrow_{w} b$ *and* $C\langle a/x\rangle \equiv t, C\langle b/x\rangle \equiv s$*.*

Proof. By induction on the reduction proof. We consider the two important cases

- Case (β). Then both terms are locally-closed, thus the contraction is valid in $\longrightarrow_w$. We take them as our a and b, and put $C \equiv x^G$ for x fresh;
- Case (ξ). Given $\lambda y.t \longrightarrow_{ex} \lambda y.s$, by induction hypothesis on $t \longrightarrow_{ex} s$ we have C', $a \longrightarrow_w b$ and $C'\langle a/x\rangle \equiv t, C'\langle b/x\rangle \equiv s$. We put $C \equiv \lambda y.C'$ and conclude. ◀

▶ **Lemma 12.** *The following substitution rule is admissible*

$$
\frac{a \longrightarrow_{ex} b}{t\langle a/x\rangle \longrightarrow_{ex} t\langle b/x\rangle}
$$

Proof. By structural induction on t. ◀

▶ **Theorem 13.** $t \longrightarrow_w s \iff t \longrightarrow_{ex} s$ *for any locally-closed terms* t *and* s*.*

Proof. We consider each implication separately.

- ($\Longrightarrow$) By observing that every rule in $\longrightarrow_w$ is admissible in $\longrightarrow_{ex}$.
- ($\Longleftarrow$) By Lemma 11, we get a term C and locally-closed terms a, b such that $a \longrightarrow_w b$ and $C\langle a/x\rangle \equiv t, C\langle b/x\rangle \equiv s$. We conclude by global variable substitution. ◀

Note that the equivalence in Theorem 13 extends to the reflexive transitive closures – since reduction does not affect free local variables – thus in particular the two relations give rise to the same set of normal forms for the same locally-closed term.

This result allows us to use the normalization function defined in this section to compute λ^{wk} normal forms, i.e. $\longrightarrow_w$-normal terms, provided such function computes $\longrightarrow_{ex}$-normal terms as we intended. We do not formally argue for this last point now. Rather, in the next sections we will adapt these ideas to the typed case, and define an "explicit" calculus based on $\longrightarrow_{ex}$ that is equivalent to the standard, "implicit" one. We will then prove the explicit calculus normalizing, and transporting along the equivalence will provide us with a normalization proof for the implicit calculus.

4 Typed weak λ-reduction

4.1 Simply-typed weak λ-calculus: λ^{wk}

We now define λ^{wk}, a simply-typed λ-calculus with weak λ-conversion judgments, that we aim to prove normalizing on well-typed terms. We use the same two-variable syntax as the previous Section. Types Ty and contexts Ctxt are defined as follows

$$\mathsf{Ty} \ni A, B ::= A \Rightarrow B \qquad\qquad \mathsf{Ctxt} \ni \Gamma ::= \cdot \mid \Gamma, x : A$$

Here we write $\Gamma \ni x : A$ for the predicate that is true when $x : A$ is included in Γ. We will assume that variables in contexts have unique names. The type system is shown in Figure 1. Even though we only care about locally-closed terms, we formulate the typing judgments in a slightly more general way, that considers arbitrary, not necessarily locally-closed terms. The judgments are of the form $\Gamma; \Delta \vdash t : A$, with a double context assigning a type to, respectively, "global" and "local" assumptions. We thus call the two contexts *global* and *local*. A well-typed term $\Gamma \vdash t : A$ of λ^{wk} is one where there are no free local variables, or equivalently, one that is typeable under an empty "local" context. That is, $\Gamma \vdash t : A :\equiv \Gamma; \cdot \vdash t : A$. Typed weak λ-conversion judgments are given by the inductive relation $\_ \vdash \_ \sim \_ : \_$, and essentially provide a formulation of the relation in Definition 9 with typed equality judgments.

4.2 Explicit weak λ-calculus: λ^{ex}

We now define a type system for λ^{ex}, with explicit weak λ-reduction (Definition 10) as equality judgments. The motivation for this system is the same behind λ^{ex} in the untyped case, namely to avoid the issues of λ^{wk} in specifying a normalization algorithm and proving it correct (see Section 2.1.) We will establish the following results:

- λ^{wk} and λ^{ex} are equivalent on locally-closed terms;
- λ^{ex} is normalizing on arbitrary (possibly non-locally-closed) well-typed terms.

Since every well-typed λ^{wk}-term is locally-closed, these two results imply what we ultimately seek to prove, namely normalization for λ^{wk} (Section 5.4). The advantage is that the actual normalization proof is carried out on λ^{ex}, which plays better with already-known proof methods that assume the ξ-rule, such as Normalization by Evaluation (as used, for example, in [11, 16].)

$\boxed{\_;\_ \vdash \_ : \_}$

$$\frac{\Gamma \ni x : A}{\Gamma;\Delta \vdash x^G : A} \qquad \frac{\Delta \ni x : A}{\Gamma;\Delta \vdash x^L : A} \qquad \frac{\Gamma;\Delta, x : A \vdash t : B}{\Gamma;\Delta \vdash \lambda x.t : A \Rightarrow B} \qquad \frac{\Gamma;\Delta \vdash t : A \Rightarrow B \quad \Gamma;\Delta \vdash s : A}{\Gamma;\Delta \vdash t\, s : B}$$

$\boxed{\_ \vdash \_ \sim \_ : \_}$

$$\frac{\Gamma;x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda x.t)\, s \sim t[s/x] : B}\ (\beta) \qquad \frac{\Gamma, x : A;\cdot \vdash t : B \qquad \Gamma \vdash a \sim b : A}{\Gamma \vdash t\langle a/x\rangle \sim t\langle b/x\rangle : A}\ (\sigma)$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t \sim t : A} \qquad \frac{\Gamma \vdash t \sim s : A}{\Gamma \vdash s \sim t : A} \qquad \frac{\Gamma \vdash t \sim s : A \qquad \Gamma \vdash s \sim r : A}{\Gamma \vdash t \sim r : A}$$

Figure 1 Simply-typed λ-calculus with weak λ-reduction.

$\boxed{\_;\_ \vdash \_ \longrightarrow \_ : \_}$

$$\frac{\Gamma;x : A \vdash t : B \qquad \Gamma;\cdot \vdash s : A}{\Gamma;\Delta \vdash (\lambda x.t)s \longrightarrow t[s/x] : B}\ (\beta) \qquad \frac{\Gamma;\Delta, x : A \vdash t \longrightarrow s : B}{\Gamma;\Delta \vdash \lambda x.t \longrightarrow \lambda x.s : A \Rightarrow B}\ (\xi)$$

$$\frac{\Gamma;\Delta \vdash s : A}{\Gamma;\Delta \vdash t \longrightarrow t' : A \Rightarrow B} \qquad \frac{\Gamma;\Delta \vdash t : A \Rightarrow B}{\Gamma;\Delta \vdash ts \longrightarrow ts' : B}$$

$\boxed{\_;\_ \vdash \_ \sim \_ : \_}$

$$\frac{\Gamma;\Delta \vdash t \longrightarrow s : A}{\Gamma;\Delta \vdash t \sim s : A} \qquad \frac{\Gamma;\Delta \vdash t : A}{\Gamma;\Delta \vdash t \sim t : A} \qquad \frac{\Gamma;\Delta \vdash t \sim s : A}{\Gamma;\Delta \vdash s \sim t : A} \qquad \frac{\Gamma;\Delta \vdash t \sim s : A \quad \Gamma;\Delta \vdash s \sim r : A}{\Gamma;\Delta \vdash t \sim r : A}$$

Figure 2 Reduction and conversion judgments of λ^{ex}.

The raw syntax and the typing judgments of λ^{ex} are the same as λ^{wk}, with the difference that terms can now be well-typed under an arbitrary local context. Figure 2 shows the definition of typed equality judgments for λ^{ex}, written $\Gamma;\Delta \vdash t \sim s : A$, and again formulated on arbitrary two-variable terms. Note that we do not axiomatize conversion directly, but define it as the equivalence closure of typed one-step reduction $\Gamma;\Delta \vdash t \longrightarrow s : A$, for purely technical reasons. Similarly to Lemma 12, we prove substitution admissible:

▶ **Lemma 14.** *If* $\Gamma, x : A;\Delta \vdash t : B$ *and* $\Gamma;\cdot \vdash a \sim b : A$, *then* $\Gamma;\Delta \vdash t\langle a/x\rangle \sim t\langle b/x\rangle : B$.

Proof. By induction on the derivation of t. ◀

4.3 Equivalence between λ^{wk} and λ^{ex}

We now show that λ^{wk} and λ^{ex} are equivalent on locally-closed terms. $\Gamma;\cdot \vdash t : A \iff \Gamma \vdash t : A$ follows by definition, thus we are left to show that $\Gamma \vdash t \sim s : A \iff \Gamma;\cdot \vdash t \sim s : A$. This is essentially the typed version of Theorem 13, and it relies on an adaptation of Lemma 11 with a proof that the term extraction involved preserves typing.

▶ **Lemma 15.** *For all derivations of a typed reduction* $\Gamma;\Delta \vdash t \longrightarrow s : A$*, there exist terms* C, a, b*, a type* X*, and a fresh variable* x *such that* $\Gamma, x : X;\Delta \vdash C : A$ *is derivable,* $\Gamma \vdash a \sim b : X$ *is derivable, and* $C\langle a/x\rangle \equiv t$*,* $C\langle b/x\rangle \equiv s$*.*

Proof. By induction on the derivation of $t \longrightarrow s$. ◀

▶ **Theorem 16.** *For all* Γ, A, t, s*,* $\Gamma \vdash t \sim s : A \iff \Gamma;\cdot \vdash t \sim s : A$*.*

Proof. Just an adaptation of the proof of Theorem 13 to the typed case. ◀

5 Normalization by Evaluation

Normalization by Evaluation (NbE) is a semantic method to prove normalization for typed λ-calculi. It was originally employed by Martin-Löf, although not under this name, to give a proof of normalization for a weak, combinatory version of his Intuitionistic Type Theory [35]. The method was later applied to the $\lambda\beta\eta$ calculus by Berger and Schwichtenberg [11], as a model construction where the interpretation function is invertible by an operation called *reification*. The composition of interpretation and reification is normalization. An advantage of NbE is that it can be justified by semantic arguments like logical relations [2] or *glueing* [16], rather than cumbersome term rewriting techniques. NbE amounts to establishing the following properties of a normalization function nf:

- Completeness: if $\Gamma \vdash t \sim s : A$, then $nf(t) = nf(s)$;
- Soundness: if $\Gamma \vdash t : A$, then $\Gamma \vdash t \sim nf(t) : A$.

From these properties we get that convertibility is equivalent to syntactic identity of normal forms: $\Gamma \vdash t \sim s : A \iff nf(t) = nf(s)$. Since syntactic identity of normal forms is decidable, so is convertibility.

In the rest of this section we give an overview of a proof of untyped NbE for λ^{ex}. This variant of NbE was originally detailed in [5], and differs from type-directed NbE like that of [35] by its reliance on a normalization function that acts on untyped raw syntax alone.

One motivation for using untyped NbE here stems from our formalization work, which involved several substitution lemmas that we though were easier to carry out on raw syntax. Another reason has to do with our plan to extend this work to dependent types. Dependent well-typed syntaxes have been historically difficult to develop inside Type Theory itself [19]. Work on quotient inductive-inductive types [8] seems to offer a viable solution, although it was too recent to have been considered here.

We employ untyped NbE as described in [2]. Most of the proof replicates [2] quite closely, so we only highlight the parts that are specific to λ^{ex}, and direct the reader to the Agda formalization or the author's MSc Thesis [37] for the details.

5.1 Semantic domain and interpretation

NbE relies on an interpretation function $[\![\_]\!]$ from the syntax to a semantic domain D, given an environment ρ assigning meaning to the free variables of the input term. In the case of weak λ-reduction, we can just use syntactic normal forms as semantic values, with syntactic identity as equality in the model. Hence we put $\mathsf{D} \equiv \mathsf{Term}$, and take the (partial) normalization function from Section 3 as interpretation $[\![\_]\!]$. Part of the proof of normalization will be to show that this function is total on well-typed terms.

5.2 Completeness of NbE

Completeness of NbE amounts to showing that the interpretation of convertible terms yields equal semantic values. In our case, we need to show that judgmentally equal terms *do* have a normal form, and this is the same. We follow [2] and strengthen our syntactic model by defining appropriate *semantic types* $\mathcal{A} \in \mathcal{P}(\mathsf{D})$, that is subsets of normal terms closed under neutral values: $\mathsf{Ne} \subseteq \mathcal{A} \subseteq \mathsf{Nf}$. These sets play a similar role to *saturated sets*, or Tait's sets of computable terms [39]. Given semantic types $\mathcal{A}, \mathcal{B}$, we can form the semantic function space $\mathcal{A} \to \mathcal{B}$ of all normal forms f that map elements $a \in \mathcal{A}$ to elements $f \bullet a \in \mathcal{B}$.

We interpret syntactic types as expected, namely $[\![A \Rightarrow B]\!] = [\![A]\!] \to [\![B]\!]$. Similarly, we interpret contexts Γ as subsets of substitutions $[\![\Gamma]\!] \in \mathcal{P}(\mathsf{Subst})$, namely those that map assumptions $\Gamma \ni x : A$ to values in $[\![A]\!]$. We write $[\![t]\!]\rho \simeq [\![s]\!]\rho \in \mathcal{A}$ whenever t and s evaluate to the same normal form in $\mathcal{A}$. We then define semantic typing and conversion judgments:

$$\Gamma; \Delta \models t : A :\equiv \forall(\rho \in [\![\Delta]\!]),\ [\![t]\!]\rho \simeq [\![t]\!]\rho \in [\![T]\!]$$

$$\Gamma; \Delta \models t \sim s : A :\equiv \forall(\rho \in [\![\Delta]\!]),\ [\![t]\!]\rho \simeq [\![s]\!]\rho \in [\![A]\!]$$

▶ **Theorem 17.** *For all t, s, A, Γ, Δ,*
- *if $\Gamma; \Delta \vdash t : A$, then $\Gamma; \Delta \models t : A$;*
- *if $\Gamma; \Delta \vdash t \longrightarrow s : A$, then $\Gamma; \Delta \models t \sim s : A$;*
- *if $\Gamma; \Delta \vdash t \sim s : A$, then $\Gamma; \Delta \models t \sim s : A$.*

Proof. By induction on the derivations. ◀

▶ **Corollary 18** (Completeness of NbE)**.** *For all Γ, A, t, s,*
1. *If $\Gamma; \Delta \vdash t : A$, then t has a normal form, namely $[\![t]\!]$;*
2. *If $\Gamma; \Delta \vdash t \sim s : A$, then t and s have the same normal form, i.e. $[\![t]\!] = [\![s]\!]$.*

Proof. Both points follow from Theorem 17. ◀

A consequence of completeness of NbE is that $[\![\_]\!]$ is total on well-typed terms. We write $[\![t]\!]$ for the interpretation/normalization function applied to the empty substitution.

5.3 Kripke logical relations and soundness of NbE

Soundness of NbE is the statement that well-typed terms are convertible to their normal form. To prove this, we rely on the definition of a *Kripke logical relation*. Logical relations are families of relations defined by induction on (syntactic) types. Kripke logical relations [28] are additionally indexed by a set of *worlds* together with an accessibility relation, in the sense of Kripke semantics. In our case, worlds are represented by contexts and substitutions. Our logical relation relates well-typed terms with semantic values, i.e. normal forms:

$$\begin{aligned}&\Theta; \Gamma \vdash M \circledR N : A \Rightarrow B :\equiv \\ &\quad M = \lambda x.t \ \wedge\ N = \lambda x.d \wedge \\ &\quad (\text{for all } \Theta; \Delta \vdash \sigma : \Gamma \text{ and } \Theta; \Delta \vdash s \circledR a : A,\ \text{then } \Theta; \Delta \vdash t[\sigma[s/x]] \circledR d[\sigma[a/x]] : B)\end{aligned}$$

where we write $\Theta; \Delta \vdash \sigma : \Gamma$ for substitutions σ mapping assumptions $\Gamma \ni x : A$ to terms $\Theta; \Delta \vdash t : A$. We can show that related objects are convertible.

▶ **Lemma 19.** *If $\Gamma; \Delta \vdash t \circledR a : T$ then $\Gamma; \Delta \vdash t \sim a : T$.*

Proof. By induction on T and on the logical relation. ◀

Note that the proof of Lemma 19 crucially depends on the ξ-rule. We will now prove that every well-typed term is logically related to its semantics. This result is generally called *fundamental lemma of logical relations*, and it is stated below in a generalized version, where terms are related up to some parallel substitutions assigning logically-related pairs of terms/values to free variables. We write $\Gamma; \Delta \vdash_s \sigma \mathrel{®} \rho : \nabla$ for parallel substitutions σ, ρ mapping assumptions in ∇.

▶ **Lemma 20.** *If* $\Theta; \Gamma \vdash t : T$ *and* $\Theta; \Delta \vdash_s \sigma \mathrel{®} \rho : \Gamma$*, then* $\Theta; \Delta \vdash t[\sigma] \mathrel{®} [\![t]\!]\, \rho : T$*.*

Proof. By induction on the derivation of t. ◀

▶ **Theorem 21.** *[Soundness of NbE] If* $p \in \Gamma; \Delta \vdash t : A$*, then* $\Gamma; \Delta \vdash t \sim [\![t]\!] : A$*.*

Proof. By completeness of NbE, Lemma 20, and Lemma 19. ◀

As a consequence of NbE we get the following results (recall the beginning of Section 5):

▶ **Theorem 22** (Normalization of λ^{ex})**.** *If* $\Gamma; \Delta \vdash t : A$*, then* $\exists t'$ *normal s.t.* $\Gamma; \Delta \vdash t \sim t' : A$*.*

▶ **Corollary 23.** *Given* $\Gamma; \Delta \vdash t : A$*,* $\Gamma; \Delta \vdash s : A$*, the judgment* $\Gamma; \Delta \vdash t \sim s : A$ *is decidable.*

5.4 Normalization for λ^{wk}

From the equivalence result between λ^{wk} and λ^{ex} and normalization for the latter, we get

▶ **Theorem 24.** *[Normalization] If* $\Gamma \vdash t : A$*, then* $\exists t' \in \mathsf{Nf}$ *s.t.* $\Gamma \vdash t \sim t' : A$*.*

Proof. By Theorem 22 and Theorem 16. ◀

▶ **Corollary 25.** *If* $\Gamma \vdash t : A$ *and* $\Gamma \vdash s : A$*, then* $\Gamma \vdash t \sim s : A$ *is decidable.*

Proof. By Corollary 23, Theorem 16, and the fact that decidability respects logical equivalence. ◀

6 Conclusions

This article studies the notion of weak reduction originally due to Howard [25], and called here weak λ-reduction after [13]. In particular, it addresses the problem of defining an algorithmic procedure to compute normal forms of weak λ-reduction, and constructively proving the normalization theorem for a simply-typed λ-calculus equipped with this notion of reduction. The first part of this work includes a comparison of weak λ-reduction with other weak notions of reduction for the λ-calculus and combinatory logic. This comparison seems to reveal that these calculi are not, as currently developed, strong enough to compute weak λ-reduction normal forms. The solution proposed here relies instead on the definition of an "explicit" version of weak reduction that is equivalent to the original weak λ-reduction, and that facilitates the definition a normalization algorithm and the reasoning about its correctness. As far as we are aware, this provides the first detailed specification of a normalization algorithm for weak λ-reduction, and a proof of its correctness in the typed case. Our work has been fully formalized in the Agda proof assistant [2].

[2] The formalization can be found at `https://github.com/fsestini/nbe-weak-stlc`

6.1 Related work

Weak λ-reduction seems to have received limited attention in the literature, with some exceptions [25, 13] already mentioned in the previous sections. An early version of Martin-Löf Type Theory [35] had a primitive substitution rule and no ξ-rule. The author argues in [34] that a rule like ξ is unacceptable because stronger than what is intuitively and informally understood as definitional equality.

Weak λ-reduction is also employed in the Minimalist Foundation, a two-level foundation for constructive mathematics in the style of Martin-Löf Type Theory ideated by Sambin and Maietti in [33], and completed by Maietti to a formal system in [32]. There, the ξ-rule is rejected because it makes it easy to give a Kleene realizability interpretation for the theory, a property of ξ-free systems that was already noted in [34]. The realizability model is then used to show consistency with the formal Church Thesis and the Axiom of Choice [27].

Kesner et al. [29, 30] study weak-head reduction in the context of intersection types and call-by-need reduction strategies. Hyland and Ong [26] use weak reduction to construct a PCA of strongly-normalizing λ-terms as a basis for a general method to prove strong normalization for various type theories. The notion of equality in the PCA is a weak reduction relation similar to weak λ-reduction, that only contracts closed redexes. The same kind of restricted reduction is also employed in [22]. In [6], Akama introduces a translation from λ-terms to combinators, so that a term is strongly-normalizing under β-reduction if and only if its translation is strongly-normalizing under the weak conversion of combinatory logic.

The ideas presented in this article are exposed in more detail in the author's (unpublished) Master's Thesis [37], where normalization is proved for System T rather than simple types. The thesis also contains an analysis of the problem for systems with dependent types, and a proof of NbE for a version of Martin-Löf Type Theory with one universe and weak explicit substitutions. The author later discovered that a similar system had also been developed by Barras et al. [10, 15].

The proofs of normalization shown in this work are based on Normalization by Evaluation. NbE was first employed by Martin-Löf in [35] for his combinatory theory, although not under this name. The method was later rediscovered in [11] in the context of the simply-typed λ-calculus with η equality. Coquand and Dybjer [16] apply NbE for typed combinatory logic and two weak λ-calculi, using a model construction inspired by the categorical notion of *glueing*. Untyped NbE, originally developed in [5], is employed here following [2].

6.2 Future work

We would like to study further the connection between weak λ-reduction, weak explicit substitutions, and combinatory logic. In particular, we conjecture that weak explicit substitutions can be shown to simulate weak λ-reduction, given suitable translation functions between terms of the two calculi that we have sketched but not yet proved correct.

Another direction for the future is the extension of this work to weak systems with dependent types, most notably the intensional level of the Minimalist Foundation. Past attempts seem to suggest that the method exposed here does not scale well to dependent types with typed equality judgments. However, it does if the calculus is based on an untyped conversion relation, like the one considered in [3]. Thus, a solution could be to first prove that the weak Type Theory of interest, defined with typed equality judgments, is equivalent to its formulation based on untyped conversion, possibly using results from [38]. A different approach could be to show that the type theory with weak explicit substitutions in [37] is equivalent to the one with implicit substitutions.

References

1 M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, October 1991.

2 A. Abel. *Normalization by Evaluation, Dependent Types and Impredicativity.* Habilitation thesis, Institut für Informatik, Ludwig-Maximilians-Universität München, 2013.

3 Andreas Abel, Klaus Aehlig, and Peter Dybjer. Normalization by Evaluation for Martin-Löf Type Theory with One Universe. *Electronic Notes in Theoretical Computer Science*, 173:17–39, April 2007.

4 Samson Abramsky. The lazy lambda calculus. In David A. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

5 Klaus Aehlig and Felix Joachimski. Operational aspects of untyped Normalisation by Evaluation. *Mathematical Structures in Computer Science*, 14(4):587–611, 2004.

6 Yohji Akama. A λ-to-CL translation for strong normalization. In Philippe de Groote and J. Roger Hindley, editors, *Typed Lambda Calculi and Applications*, pages 1–10, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

7 Thorsten Altenkirch and James Chapman. Big-step Normalisation. *J. Funct. Program.*, 19(3–4):311–333, July 2009.

8 Thorsten Altenkirch and Ambrus Kaposi. Type Theory in Type Theory Using Quotient Inductive Types. *SIGPLAN Not.*, 51(1):18–29, January 2016. doi:10.1145/2914770.2837638.

9 Hendrik Pieter Barendregt. *The lambda calculus: its syntax and semantics.* North-Holland, 1984.

10 Bruno Barras, Thierry Coquand, and Simon Huber. A generalization of the Takeuti–Gandy interpretation. *Mathematical Structures in Computer Science*, 25(5):1071–1099, 2015. doi:10.1017/S0960129514000504.

11 U. Berger and H. Schwichtenberg. An inverse of the evaluation functional for typed lambda-calculus. In *[1991] Proceedings Sixth Annual IEEE Symposium on Logic in Computer Science*, page 203–211, July 1991.

12 Ana Bove, Peter Dybjer, and Ulf Norell. A Brief Overview of Agda - A Functional Language with Dependent Types. In *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, pages 73–78, 2009. doi:10.1007/978-3-642-03359-9_6.

13 Naim Çağman and J.Roger Hindley. Combinatory weak reduction in lambda calculus. *Theoretical Computer Science*, 198(1):239–247, 1998.

14 Arthur Charguéraud. The Locally Nameless Representation. *Journal of Automated Reasoning*, 49(3):363–408, October 2012.

15 Thierry Coquand. Weak Type Theory (unpublished note). URL: http://www.cse.chalmers.se/~coquand/wmltt.pdf.

16 Thierry Coquand and Peter Dybjer. Intuitionistic Model Constructions and Normalization Proofs. *Preliminary Proceedings of the 1993 TYPES Workshop, Nijmegen*, 1993.

17 Pierre-Louis Curien, Thérèse Hardin, and Jean-Jacques Lévy. Confluence Properties of Weak and Strong Calculi of Explicit Substitutions. *J. ACM*, 43(2):362–397, March 1996.

18 H. B. Curry and R. Feys. Combinatory Logic. *Philosophische Rundschau*, 6(3/4):294, 1958.

19 Nils Anders Danielsson. A Formalisation of a Dependently Typed Language as an Inductive-Recursive Family. In Thorsten Altenkirch and Conor McBride, editors, *Types for Proofs and Programs*, pages 93–109, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

20 N. G de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, January 1972.

21 Peter Dybjer and Denis Kuperberg. Formal Neighbourhoods, Combinatory Böhm Trees, and Untyped Normalization by Evaluation, 2008.

22 J.-Y. Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse de doctorat d'État, Univerité Paris 7, 1972.

23 Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2012.

24 J. Roger Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators: An Introduction*. Cambridge University Press, 2 edition, 2008.

25 W.A. Howard. Assignment of Ordinals to Terms for Primitive Recursive Functionals of Finite Type. In *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1970.

26 J.M.E. Hyland and C. h. L. Ong. Modified Realizability Toposes and Strong Normalization Proofs (Extended Abstract). In *Typed Lambda Calculi and Applications, LNCS 664*, pages 179–194. Springer-Verlag, 1993.

27 Hajime Ishihara, Maria Emilia Maietti, Samuele Maschio, and Thomas Streicher. Consistency of the intensional level of the Minimalist Foundation with Church's thesis and axiom of choice. *Archive for Mathematical Logic*, January 2018.

28 Achim Jung and Jerzy Tiuryn. *A New Characterization of Lambda Definability*. Springer, 1993.

29 Delia Kesner. Reasoning About Call-by-need by Means of Types. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures*, pages 424–441, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

30 Delia Kesner, Alejandro Ríos, and Andrés Viso. Call-by-Need, Neededness and All That. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures*, pages 241–257, Cham, 2018. Springer International Publishing.

31 Jean-Jacques Lévy and Luc Maranget. Explicit Substitutions and Programming Languages. In *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, page 181–200. Springer, Berlin, Heidelberg, December 1999.

32 M.E. Maietti. A minimalist two-level foundation for constructive mathematics. *Annals of Pure and Applied Logic*, 160(3):319–354, 2009.

33 M.E. Maietti and G. Sambin. Toward a Minimalist Foundation for Constructive Mathematics. *From Sets and Types to Topology and Analysis: Practicable Foundations for Constructive Mathematics*, 48, 2005.

34 Per Martin-Löf. About Models for Intuitionistic Type Theories and the Notion of Definitional Equality. *Studies in Logic and the Foundations of Mathematics*, 82, December 1975.

35 Per Martin-Löf. *An Intuitionistic Theory of Types: Predicative Part*, volume 80 of *Logic Colloquium '73*, page 73–118. Elsevier, January 1975.

36 C.-H. Luke Ong. Fully Abstract Models of the Lazy Lambda Calculus. In *FOCS*, 1988.

37 Filippo Sestini. Normalization by Evaluation for Typed Lambda-Calculi with Weak Conversions (MSc's Thesis, unpublished), 2018. URL: `http://www.cs.nott.ac.uk/~psxfs5/msc-thesis.pdf`.

38 Vincent Siles and Hugo Herbelin. Pure Type System conversion is always typable. *Journal of Functional Programming*, 22(2):153–180, 2012.

39 W. W. Tait. Intensional interpretations of functionals of finite type. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

40 A. S. Troelstra and D. van Dalen. Constructivism in Mathematics, Volume 2. *Studia Logica*, 50(2):355–356, 1991.