Well-founded Recursion over Contextual Objects

Shanshan (Sherry) Ruan and Brigitte Pientka

School of Computer Science, McGill University, Montreal, Canada
shanshan.ruan@mail.mcgill.ca bpientka@cs.mcgill.ca

Abstract. Today, we routinely specify and reason about the runtime behavior of software using formal systems such as type systems or logics for access control to establish safety properties. Our work is concentrated at Beluga [Pientka, 2008], a novel powerful programming and proof environment for reasoning about formal systems specified in the logical framework LF [Harper et al., 1993]. It provides direct support for common and tricky routines dealing with variables and allows embedding contextual LF objects, i.e. a LF object paired with a context, in programs. Proofs depending on assumptions are characterized by such contextual objects. In this paper, we develop a well-founded recursion principle over contextual objects, describe a call-by-value small-step semantics, and prove weak normalization, i.e. every program terminates, following Tait's method of logical relations [Tait, 1967] and building on recent work by Lindley and Stark [2005]. This work is an important milestone towards justifying well-founded induction principle over LF specification, a long-standing open problem in the area of mechanizing the reasoning about formal systems. At this point, we concentrate on the simply-typed fragment, but we believe our framework scales to the dependently-typed setting.

1 Introduction

Logical framework (LF) [Harper et al., 1993] is a system for specifying formal system via axioms and inference rules. In practice, we often need to model proof objects which depend on a context of assumptions. We call such objects contextual objects pairing the object together with the context in which they are meaningful, i.e. term M may refer to variables in Ψ. We internalize this notion as a contextual type $[\Psi.A]$ where M has type A in the context Ψ based on contextual modal type theory [Nanevski et al., 2008].

Higher-order abstract syntax (HOAS) is a simple elegant technique in logical frameworks for modelling binders in the object language. The chief idea is that we represent variables of the object language with variables of the meta-language and use meta-level binders to model object-level binders. A particularly important application of Higher-Order Abstract Syntax is that recursive functions can implement inductive proofs and case analysis in the inductive proof agrees with pattern matching against higher-order data in the recursive function. However,

higher-order abstract syntax encodings are not compatible with inductive principle in general, and hence the design of a concise and powerful higher-order recursion principle has remained a long-standing open problem in the area of mechanizing the reasoning about formal systems. For instance, Despeyroux and Hirschowitz [1994] assert that it is impossible to prove the higher-order induction principle at the object level and moreover, they do not know a simple to formulate it at the object level. Despeyroux et al. [1997] propose a modal lambda-calculus with primitive recursive functional to preserves higher-order abstract syntax via iteration and case constructs. However, their system is based on modal types and does not support reasoning with assumptions. Furthermore, their iteration over functions is not consistent with pattern matching and is difficult to extend to contextual objects.

In this work, we present a calculus laying a theoretical foundation for Beluga [Pientka, 2008] which supports programming with contextual objects and has been one of the most advanced programming and proof assistants for specifying and reasoning about formal systems and proofs. The language is based on contextual modal type theory [Nanevski et al., 2008] and allows primitive recursion over contextual objects. Our main contribution is the design of a well-founded recursion principle over contextual objects and weak normalization proof. Our normalization proof is built on the standard logical relations method which was first introduced by Tait [1967] and further enhanced by Girard [1972]. Using such technique, Girard et al. [1990] prove the strong normalization theorem of the simply typed lambda-calculus with function types and cross types. By introducing a notion of continuation, Lindley and Stark [2005] extend the proof to the computational metalanguage λ_{ml} which distinguishes data between computations. However, their calculus is restricted because let terms are confined to T type (box type) and they do not provide support for meta-variables or context variables. In our work, we fill these vacancies and further improve on it by proving weak normalization theorem for contextual objects.

This is an important milestone, since it justifies that well-founded recursive programs implement well-founded inductive proofs. This provides a foundation for writing total functions over contextual objects. More broadly, it provides an important, currently missing piece in the foundation of the proof and programming language Beluga. At this point, we have concentrated the simply-typed fragment, but we believe our framework scales to the dependently-typed setting nicely.

The structure of the rest of the paper is organized as follows: we present a simply-typed calculus based on contextual modal types in section 2. We first introduce the data-level language in 2.1, and lift it to meta-level to give a uniform treatment to all contextual objects in 2.2. Then we define a computation-level language in 3.3. Furthermore, we present a small-step operational semantics in 3.4 and design a splitting and coverage checking algorithm in 3.5. In section 3 we prove weak normalization theorem for this language. We start by defining the notion of reducibility candidates for each type in 3.1 and then prove a series of auxiliary lemmas in 3.2. Finally we reach the weak normalization theorem in

3.3. We conclude this paper and discuss some complications regarding extending our framework to dependent types in section 4.

2 Formalities

In this section, we present the formal representation of our language which is based on contextual modal types [Nanevski et al., 2008] and its extension in Pientka [2008]. We cleanly separate data from computations and restrict the data-level language to the simply-typed modal lambda-calculus for simplicity, however the framework scales to dependent types (see the discussion in section 4). The computation language supports pattern matching on data-level objects and primitive recursion. We also include context variables and abstract over context since the local context may grow as we recursively traverse a data-level binder. Afterwards we describe the corresponding type system and the small-step operational semantics together with the underlying coverage checking algorithm.

2.1 Data-level terms and typing rules

Our contextual LF is a simply-typed modal lambda-calculus closely related to Pientka [2008], but later we will lift data-level objects to meta-level to allow for uniform treatment of meta-objects.

Types	$A, B ::= P \mid A \to B$
Heads	$H ::= c \mid x \mid p[\sigma]$
Neutral Terms	$R ::= H \mid R\ N \mid u[\sigma]$
Normal Terms	$M, N ::= R \mid \lambda x.\, M$
Substitutions	$\sigma ::= \cdot \mid \sigma, M \mid \sigma; H \mid \mathtt{id}_\psi$
Contexts	$\Psi, \Phi ::= \cdot \mid \Psi, x{:}A \mid \psi$

We only characterize normal objects since these are the only meaningful objects in LF. We achieve this by adopting a recent technique for logical frameworks from Watkins et al. [2002]. While the syntax above only verifies that term M does not contain $\beta-$redexes, the typing rules guarantee that a well-typed term is fully $\eta-$expanded. We distinguish between data-level ordinary variables x for ordinary binding in λ-terms, data-level meta-variables u, v for describing patterns when analysing contextual objects, and parameter variable p for representing a variables in a context. Data-level ordinary context Φ, Ψ only contain ordinary data-level variables x. We associate a contextual variable (meta-variable or parameter variable) with a postponed substitution σ and characterize it as a closure. We apply the substitution σ as soon as the contextual variable is instantiated. The domain of the substitutions is kept implicit to avoid the renaming. Substitution σ is comprised of normal forms (σ, M) or heads $(\sigma; H)$. We assume all constants are declared in a well-formed signature Σ which is always suppressed from now on.

Context variable ψ allows us to abstract over context. A context contains at most one context variable ψ and it must appear at the left. We also need to extend the notion of $\mathtt{id}$ to include general local contexts:

$$\mathtt{id}(\cdot) = \cdot \qquad \mathtt{id}(\psi) = \mathtt{id}_\psi \qquad \mathtt{id}(\Psi, x{:}A) = \mathtt{id}(\Psi), x$$

Data-level Typing We use a bidirectional type system to check normal terms against types and to synthesize types for neutral terms.

$\Delta; \Psi \vdash M \Leftarrow A$	Check normal object M against type A
$\Delta; \Psi \vdash R \Rightarrow A$	Synthesize type A for neutral object R
$\Delta; \Psi \vdash \sigma \Leftarrow \Phi$	Check substitution σ against context Φ

Neutral terms $\quad \Delta; \Psi \vdash R \Rightarrow A$

$$\frac{\Psi(x) = A}{\Delta; \Psi \vdash x \Rightarrow A}\ \text{TRvar} \qquad \frac{\Delta(p) = \#\Phi.A \quad \Delta; \Psi \vdash \sigma \Leftarrow \Phi}{\Delta; \Psi \vdash p[\sigma] \Rightarrow A}\ \text{TRpvar}$$

$$\frac{\Sigma(c) = A}{\Delta; \Psi \vdash c \Rightarrow A}\ \text{TRcons} \qquad \frac{\Delta(u) = \Phi.P \quad \Delta; \Psi \vdash \sigma \Leftarrow \Phi}{\Delta; \Psi \vdash u[\sigma] \Rightarrow P}\ \text{TRmvar}$$

$$\frac{\Delta; \Psi \vdash R \Rightarrow A \to B \quad \Delta; \Psi \vdash N \Leftarrow A}{\Delta; \Psi \vdash R\ N \Rightarrow B}\ \text{TRapp}$$

Normal terms $\quad \Delta; \Psi \vdash M \Leftarrow A$

$$\frac{\Delta; \Psi \vdash R \Rightarrow P \quad P = P'}{\Delta; \Psi \vdash R \Leftarrow P'}\ \text{TNneu} \qquad \frac{\Delta; \Psi, x{:}A \vdash M \Leftarrow B}{\Delta; \Psi \vdash \lambda\, x.\, M \Leftarrow A \to B}\ \text{TNlam}$$

Substitutions $\quad \Delta; \Psi \vdash \sigma \Leftarrow \Phi$

$$\frac{}{\Delta; \Psi \vdash \cdot \Leftarrow \cdot}\ \text{TSep} \qquad \frac{\Delta; \Psi \vdash \sigma \Leftarrow \Phi \quad \Delta; \Psi \vdash M \Leftarrow A}{\Delta; \Psi \vdash (\sigma, M) \Leftarrow (\Phi, x{:}A)}\ \text{TSnor}$$

$$\frac{}{\Delta; \psi, \Psi \vdash \mathtt{id}_\psi \Leftarrow \psi}\ \text{TSid} \qquad \frac{\Delta; \Psi \vdash \sigma \Leftarrow \Phi \quad \Delta; \Psi \vdash H \Rightarrow B \quad B = A}{\Delta; \Psi \vdash (\sigma; H) \Leftarrow (\Phi, x{:}A)}\ \text{TShd}$$

We require the usual conditions on bound variables and tacitly apply $\alpha-$renaming. We use a special symbol # to indicate the type of a parameter variable. In order to preserve canonical forms, we require hereditary substitution [Pientka, 2008].

2.2 Meta-level terms and typing rules

We lift contextual LF objects to meta-objects to have a uniform definition of all meta-objects. Meta-objects (both contextual objects $\hat{\Psi}.R$ and context Ψ) can be

used to index computation-level types. But we concentrate on context variables at present to simplify our formalities.

Context Schemas	$G ::= A \mid G_1 + G_2$
Meta Types	$U ::= \Psi.P \mid \#\Psi.A \mid G$
Meta Objects	$C ::= \hat{\Psi}.R \mid \Psi$
Meta Substitutions	$\theta ::= \cdot \mid \theta, C/X$
Meta Contexts	$\Delta ::= \cdot \mid \Delta, X{:}U$

As types classify terms, context schemas are introduced to classify contexts. The above definition gives rise to a compact treatment of meta-context Δ. A meta-variable X can denote a meta-variable, a parameter variable, or a context variable. C/X can represent $\hat{\Psi}.M/u$, Ψ/ψ, $\hat{\Psi}.x/p$, or $\hat{\Psi}.p'[\pi]/p$ (where π is a variable substitution so that $p[\pi]$ always produces a variable). Meta-substitution $X{:}U$ can stand for $u{:}\Psi.P$, $p{:}\#\Psi.A$, or $\psi{:}G$

Meta-level Typing We show the typing rules for meta-objects and meta-substitutions below.
Meta-level Objects $\Delta \vdash C : U$

$$\frac{}{\Delta \vdash \cdot : G}\ \text{TMepctx} \qquad \frac{\Delta; \Psi \vdash R \Leftarrow P}{\Delta \vdash \hat{\Psi}.R : \Psi.P}\ \text{TMmvar}$$

$$\frac{\Delta(\psi) = G}{\Delta \vdash \psi : G}\ \text{TMcvar} \qquad \frac{\Psi(x) = A}{\Delta \vdash \hat{\Psi}.x : \#\Psi.A}\ \text{TMx}$$

$$\frac{\Delta \vdash \Psi : G \quad A \in G}{\Delta \vdash \Psi, x{:}A : G}\ \text{TMctx} \qquad \frac{\Delta(p) = \#\Phi.A \quad \Delta; \Psi \vdash \pi \Leftarrow \Phi}{\Delta \vdash \hat{\Psi}.p[\pi] : \#\Psi.A}\ \text{TMpvar}$$

Meta-level Substitutions $\Delta \vdash \theta : \Delta'$

$$\frac{}{\Delta \vdash \cdot : \cdot}\ \text{TMemp} \qquad \frac{\Delta \vdash \theta : \Delta' \quad \Delta \vdash C : [\![\theta]\!]U}{\Delta \vdash \theta, C/X : \Delta', X{:}U}\ \text{TMc}$$

2.3 Computation-level terms and typing rules

We distinguish data-level objects from computation-level objects. While the data-level language is used to represent data, the computation-level language can describe functions which manipulate data. A meta-object (a context Ψ or a data-level term M together with a context Ψ under which the data object is

closed) is wrapped into computation via a [ ] constructor.

$$
\begin{array}{lll}
\text{Types} & & \tau ::= \tau_1 \to \tau_2 \mid [U] \mid \Pi\psi{:}G.\tau \\
\text{Expressions} & & e ::= y \mid [C] \mid \mathsf{fn}\, y.\, e \mid e_1\ e_2 \mid \Lambda\psi.e \mid e\ [\Psi] \mid \mathtt{let}\ u = e_1\ \mathtt{in}\ e_2 \\
& & \quad \mid \mathtt{rec}^{[U']\Leftarrow[U]:[\![\Psi/\psi]\!]}(e,\ \overrightarrow{b}) \\
\text{Branches} & & b ::= \Pi\,\Delta.\Pi\,\overrightarrow{\mathtt{f}\ [\Phi]\ [q]{:}[\![\Phi/\psi]\!]\tau}.\mathtt{f}\ [\Psi]\ [p] \mapsto e \\
\text{Patterns} & & p, q ::= \hat{\Psi}.M \\
\text{Contexts} & & \Gamma ::= \cdot \mid \Gamma, y{:}\tau
\end{array}
$$

There are two different kinds of function applications. We can apply a computation-level function $\mathsf{fn}\, y.\, e$ to an expression e or apply a dependent function $\Lambda\psi.e$ to a meta-object Ψ. Since we allow abstraction over context, we introduce the dependent type $\Pi\psi{:}G.\tau$ which permits index the type by context variables.

We can pattern match on data by rec expressions. We annotate it with the type of argument $[U']$ and the return type $[U]$ along with an associated context substitution $[\![\Psi/\psi]\!]$ to denote how the context variable ψ is instantiated. Such information is essential for type checking and cannot be eliminated. A Branch i is expressed as $\Pi\,\Delta_i.\Pi\,\overrightarrow{\mathtt{f}\ [\Phi^i]\ [q^i]{:}[\![\Phi^i/\psi]\!]\tau}.\mathtt{f}\ \ [\Psi_i]\ \ [p_i] \mapsto e_i$ where Φ^i and Ψ_i describes the instantiation of the type of the scrutinee in each pattern correspondingly. We explicitly list all variables occurring in a branch as $\Pi\,\Delta$. Note that Δ is always of form $\phi{:}G, \Delta'$, that is, the context variable in a branch is bound at the branch-level instead of the function-level. $\overrightarrow{\mathtt{f}\ [\Phi]\ [q]{:}[\![\Phi/\psi]\!]\tau}$ is a collection of all well-formed primitive recursive call for the pattern $\mathtt{f}\ [\Psi]\ [p]$. Hence, a recursive function can be expressed as: $\Lambda\psi.\mathsf{fn}\, y.\, \mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![\psi/\psi]\!]}(y, \overrightarrow{\Pi\,\phi{:}G, \Delta.\Pi\,\mathtt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U].\mathtt{f}\ [\phi]\ [\phi.M] \mapsto e})$. It has type $\Pi\,\psi{:}G.[\psi.P] \to [U]$.

Computation-level Type System Our type system performs coverage checking and verifies well-formed primitive recursions in the typing rule for `rec` expressions. Based on the given contextual type, the splitting algorithm described in the later section generates the most general set of complete and non-redundant patterns as well as the collection of all well-formed recursive calls for each pattern. The type system will ensure that branches provided in `rec` expressions are consistent with patterns generated by the splitting algorithm.

Computation-level terms $\Delta; \Gamma \vdash e : \tau$

$$\frac{\Gamma(y) = \tau}{\Delta; \Gamma \vdash y : \tau}\ \text{Tvar} \qquad \frac{\Delta \vdash C : U}{\Delta; \Gamma \vdash [C] : [U]}\ \text{Tmeta}$$

$$\frac{\Delta; \Gamma, y{:}\tau_1 \vdash e : \tau_2}{\Delta; \Gamma \vdash \mathsf{fn}\, y.\, e : \tau_1 \to \tau_2}\ \text{Tabs} \qquad \frac{\Delta; \Gamma \vdash e_1 : \tau_2 \to \tau \quad \Delta; \Gamma \vdash e_2 : \tau_2}{\Delta; \Gamma \vdash e_1\ e_2 : \tau}\ \text{Tapp}$$

$$\frac{\Delta, \psi{:}G; \Gamma \vdash e : \tau}{\Delta; \Gamma \vdash \Lambda\psi.e : \Pi\psi{:}G.\tau}\ \text{Tmabs} \qquad \frac{\Delta; \Gamma \vdash e : \Pi\psi{:}G.\tau \quad \Delta \vdash \Psi : G}{\Delta; \Gamma \vdash e\ [\Psi] : [\![C/X]\!]\tau}\ \text{Tcapp}$$

$$\frac{\Delta;\Gamma \vdash e_1 : [\Psi.P] \quad \Delta, u{:}\Psi.P;\Gamma \vdash e_2 : \tau}{\Delta;\Gamma \vdash \mathtt{let}\ u = e_1\ \mathtt{in}\ e_2 : \tau}\ \text{Tlet}$$

$$\frac{\begin{array}{l}\mathcal{S}{=}\mathtt{split}(\psi{:}G \vdash \psi.P){=}\{(\phi{:}G, \Delta_i' \vdash p_i : \phi.P, \overrightarrow{\mathtt{f}\ [\Phi^i]\ [q^i]}) \mid \phi{:}G, \Delta_i' \vdash q_j^i : \Phi_j^i.P\} \\ b_i = \Pi\,\phi{:}G, \Delta_i'.\Pi\,\overrightarrow{\mathtt{f}\ [\Phi^i]\ [q^i]{:}[\![\Phi^i/\psi]\!]\tau}.\mathtt{f}\ [\phi]\ [p_i] \mapsto e_i \qquad |\mathcal{S}| = |\ \overrightarrow{b}\ | \\ \Delta;\Gamma \vdash e : [\![\Psi/\psi]\!][U'] \quad \text{for all i } \Delta;\Gamma \vdash b_i : [U']{\Leftarrow}[U]{:}[\![\Psi/\psi]\!]\end{array}}{\Delta;\Gamma \vdash \mathtt{rec}^{[U']\Leftarrow[U]:[\![\Psi/\psi]\!]}(e,\ \overrightarrow{b}) : [U]}\ \text{Trec}$$

$$\frac{\begin{array}{l}\phi{:}G, \Delta_i' \vdash p_i : [\![\Psi_i/\psi]\!]U' \qquad \text{for all j } \phi{:}G, \Delta_i' \vdash q_j^i : [\![\Phi_j^i/\psi]\!]U' \\ \phi{:}G, \Delta_i', [\![\Psi_i/\psi]\!]\Delta; [\![\Psi_i/\psi]\!]\Gamma, \overrightarrow{\mathtt{f}\ [\Phi^i]\ [q^i] : [\![\Phi^i/\psi]\!][U]} \vdash e_i : [\![\Psi_i/\psi]\!][U]\end{array}}{\Delta;\Gamma \vdash \Pi\,\phi{:}G, \Delta_i'.\Pi\,\overrightarrow{\mathtt{f}\ [\Phi^i]\ [q^i]{:}[\![\Phi^i/\psi]\!][U]}.\mathtt{f}\ [\Psi_i]\ [p_i] \mapsto e_i : [U']{\Leftarrow}[U]{:}[\![\Psi/\psi]\!]}\ \text{Tb}$$

Lemma 2.1 [Substitution Lemma]
1. If $\Delta \vdash C : U$ and $\Delta' \vdash \theta : \Delta$, then $\Delta' \vdash [\![\theta]\!]C : [\![\theta]\!]U$.
2. If $\Delta;\Gamma \vdash e : \tau$ and $\Delta' \vdash \theta : \Delta$, then $\Delta';[\![\theta]\!]\Gamma \vdash [\![\theta]\!]e : [\![\theta]\!]\tau$.
3. If $\Delta;\Gamma \vdash e : \tau$ and $\Delta;\Gamma' \vdash \rho : \Gamma$, then $\Delta;\Gamma' \vdash [\rho]e : \tau$.

Proof. By induction on the first typing derivation.

2.4 Operational Semantics

We adopt a call-by-value strategy and require the evaluation to be deterministic. From the completeness and non-redundancy of the patterns, there exists a unique branch that a well-typed `rec` expression can step to. Since the type system also guarantees that recursive calls are well formed and complete in each branch, we can generate an ordinary computation-level substitution ρ (recursive calls) which is dependent on $\phi{:}G, \Delta_i'$, then we apply the meta-substitution θ to instantiate $\phi{:}G, \Delta_i'$. We only give the evaluation rules regarding meta objects.

$$\frac{e \longrightarrow e'}{e\ [\Psi] \longrightarrow e'\ [\Psi]}\ \text{Emabs} \qquad \frac{}{(\Lambda\psi.e)\ [\Psi] \longrightarrow [\![\Psi/\psi]\!]e}\ \text{Emapp}$$

$$\frac{e_1 \longrightarrow e_1'}{\mathtt{let}\ u = e_1\ \mathtt{in}\ e_2 \longrightarrow \mathtt{let}\ u = e_1'\ \mathtt{in}\ e_2}\ \text{Elet}$$

$$\frac{}{\mathtt{let}\ u = [\hat{\Psi}.M]\ \mathtt{in}\ e_2 \longrightarrow [\![\hat{\Psi}.M/u]\!]e_2}\ \text{Emlet}$$

$$\frac{e \longrightarrow e'}{\mathtt{rec}^{U'\Leftarrow U:[\![\Psi/\psi]\!]}(e,\ \overrightarrow{b}) \longrightarrow \mathtt{rec}^{U'\Leftarrow U:[\![\Psi/\psi]\!]}(e',\ \overrightarrow{b})}\ \text{Erec}$$

$$\frac{\begin{array}{l}\exists\,\text{unique}\, b_i{=}\Pi\,\phi{:}G, \Delta_i.\Pi\,\overrightarrow{\mathtt{f}\ [\Phi^i]\ [\hat{\Phi}^i.N^i]{:}[\![\Phi_i/\psi]\!][U].\mathtt{f}\ [\phi]\ [\phi.M_i] \mapsto e_i} \\ \text{s.t.}\phi{:}G, \Delta_i \vdash \hat{\Psi}.M \doteq \phi.M_i/\theta \qquad \rho = \mathtt{rec}^{[U']\Leftarrow[U]:[\![\Phi^i/\psi]\!]}([\hat{\Phi}^i.N^i],\ \overrightarrow{b})/\overrightarrow{\mathtt{f}\ [\Phi^i]\ [\hat{\Phi}_i.N_i]}\end{array}}{\mathtt{rec}^{[U']\Leftarrow[U]:[\![\Psi/\psi]\!]}([\hat{\Psi}.M],\ \overrightarrow{b}) \longrightarrow [\![\theta]\!][\rho]e_i}\ \text{Emrec}$$

Note that the ordinary substitution ρ depends on $\phi{:}G, \Delta_i$, hence we apply $[\rho]$ first and $[\![\theta]\!]$ second. The existence and the uniqueness of the branch b_i in Emrec is guaranteed by our coverage checking algorithm presented later.

Lemma 2.2 [Soundeness of substitutions]
1. $\cdot \vdash \theta : \phi{:}G, \Delta_i$
2. $[\![\theta]\!](\phi.M_i) = \hat{\Psi}.M$
3. $\phi{:}G, \Delta_i; \cdot \vdash \mathtt{rec}^{U' \Leftarrow U:[\![\Phi^i/\psi]\!]}([\hat{\Phi}^i.N_i], \overrightarrow{b})/\mathtt{f}\ [\Phi^i]\ [\hat{\Phi}_i.N_i] : \overrightarrow{\mathtt{f}\ [\Phi^i]\ [\hat{\Phi}^i.N^i]{:}[\![\Phi^i/\psi]\!][U]}$

Lemma 2.3 [Subject reduction]
If $\Delta; \Gamma \vdash e : \tau$ and $e \longrightarrow e'$, then $\Delta; \Gamma \vdash e' : \tau$.

2.5 Coverage Checking algorithm

Given a context variable $\psi{:}G$ and a meta-type $\psi.P$ which is dependent on this context variable, the `split` function considers all top-level structures of the contextual type and generates a unique set $\mathcal{S}$ of complete and non-redundant patterns. In a primitive recursive function, each pattern $\phi{:}G, \Delta_i \vdash \phi.M_i : \phi.P$ corresponds to one branch, and $\overrightarrow{\mathtt{f}\ [\Phi^i]\ [\hat{\Phi}^i.N^i]}$ corresponds to all well-formed recursive calls in such branch. Hence, the well-foundness of a primitive recursive function is justified by the splitting algorithm. We also define an auxiliary function `genMV` to generate appropriate lambda-abstractions and meta-variables. The algorithm is relatively straightforward as we only consider simply-typed case here. See Schürmann and Pfenning [2003] for issues regarding dependent types.

Splitting and pattern generating: We define `split` function as follows:

$$\mathcal{S}{=}\mathtt{split}(\psi{:}G \vdash \psi.P){=}\{(\phi{:}G, \Delta_i \vdash p_i : \phi.P, \overrightarrow{\mathtt{f}\ [\Phi^i]\ [q^i]}) \mid \phi{:}G, \Delta_i \vdash q^i_j : \Phi^i_j.P\}$$

where $\mathtt{f}\ [\Phi^i]\ [q^i]$ is the set of all possible recursive calls in each pattern.
We consider each $c{:}A$ declared in Σ and the context schema G of the context variable:
1. If c has the same atomic type as P (i.e.$\Sigma(c) = P$) then $(\phi{:}G \vdash \phi.c : \phi.P, \emptyset) \in \mathcal{S}$
2. If c has function type $A_1 \to \cdots \to A_n \to P$ $(n \geq 1)$, we apply `genMV` on each type A_i to obtain a collection of objects M_i and corresponding meta-variables $u_i{:}\Phi_i.P_i$.

$$\text{For all i: } \mathtt{genMV}(\psi{:}G; \psi \vdash A_i) = (M_i, u_i{:}\Phi_i.P_i)$$

Then $(\phi{:}G, u_1{:}\Phi_1.P_1, \cdots, u_n{:}\Phi_n.P_n \vdash \phi.c\, M_1 \cdots M_n : \phi.P, \overrightarrow{\mathtt{f}\ [\Phi^i]\ [\hat{\Phi}_i.u_i[\mathtt{id}(\Phi^i)]]})$
(for all $P_i = P$ and $\Phi_i \in G$)) $\in \mathcal{S}$
3. If c has other type, then it cannot become a head of an object which has type P in the local context ϕ, so we simply discard this case.
4. If $P \in G$, then a pattern can be generated by a parameter variable. That is, $(\phi{:}G, p{:}\#\phi.P \vdash \phi.p[\mathtt{id}_\phi] : \phi.P, \emptyset) \in \mathcal{S}$

Lemma 2.4 [Soundness of splitting algorithm]
Let $\texttt{split}(\phi{:}G \vdash \phi.P){=}\mathcal{S}$, for each $(\phi{:}G, \Delta_i \vdash \phi.M_i : \phi.P, \overrightarrow{\texttt{f}\ [\Phi^i]\ [\hat{\Phi}^i.N^i]}) \in \mathcal{S}$ $\phi{:}G, \Delta_i \vdash \phi.M_i : \phi.P$ and $\phi{:}G, \Delta_i \vdash \hat{\Phi}^i.N^i : \Phi^i.P$

Proof. By case analysis on each step.

Lemma 2.5 [Completeness and non-redundancy]
If $\texttt{split}(\phi{:}G \vdash \phi.P) = \mathcal{S}$, then for each $(\phi{:}G, \Delta_i \vdash \phi.M_i : \phi.P, \overrightarrow{\texttt{f}\ [\Phi^i]\ [\hat{\Phi}^i.N^i]})$, then the set of patterns $(\phi{:}G, \Delta_i \vdash \phi.M_i : \phi.P, \overrightarrow{\texttt{f}\ [\Phi^i]\ [\hat{\Phi}^i.N^i]})$ is complete and non-redundant, and the set of recursive calls $\overrightarrow{\texttt{f}\ [\Phi^i]\ [\hat{\Phi}^i.N^i]}$ is well-formed and complete.

Proof. The set of patterns is non-redundant since all patterns have distinct heads. Since we have considered every possible constant in Σ and every possible variable from the local context, the set is also complete.

Generating meta-variables Given a context variable $\phi{:}G$, a local context Ψ and a type A, we define $\texttt{genMV}(\phi{:}G; \Psi \vdash A) = (M, u{:}\Phi.P)$ as follows:

$$\texttt{genMV}(\phi{:}G; \Psi \vdash P) = (u[\texttt{id}(\Psi)], u{:}\Psi.P)$$

$$\frac{\texttt{genMV}(\phi{:}G; \Psi, x{:}A' \vdash A) = (M, u{:}\Phi.P)}{\texttt{genMV}(\phi{:}G; \Psi \vdash A' \to A) = (\lambda\, x.\, M, u{:}\Phi.P)}$$

In the splitting algorithm step (2), if $\Sigma(c) = A_1 \to \cdots A_n \to P$, then the type A in $\texttt{genMV}(\phi{:}G; \Psi \vdash A)$ is instantiated by each A_i respectively.

Lemma 2.6 [Soundness of genMV]
If $\texttt{genMV}(\phi{:}G; \Psi \vdash A) = (M, u{:}\Phi.P)$, then $\phi{:}G, u{:}\Phi.P; \Psi \vdash M : A$.

Proof. By induction on the type of A.

3 Weak normalization theorem

3.1 Reducibility Candidates

Introducing term collections Recall that we can write primitive recursive functions over contextual objects in a general form.

$$\Lambda\psi.\texttt{fn}\, y.\, \texttt{rec}^{[U']\Leftarrow[U]:[\![\psi/\psi]\!]}(y,\ \Pi\, \phi{:}G, \Delta.\Pi\, \overrightarrow{\texttt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U].\texttt{f}\ [\phi]\ [\phi.M] \mapsto e})$$

This primitive recursive function takes a contextual object of type $[U] = [\psi.P]$ as input and returns an object of type $[U']$ ($[U']$ may depend on the context

variable ψ). Thus, the type of the above general function form is $\Pi\psi{:}G.[\psi.P] \to [U]$.

In our normalization proof, we will treat all branches as a whole and use B to denote the collection of all branches:

$$B = \Pi\,\phi{:}G, \Delta.\Pi\,\overrightarrow{\mathtt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U].\mathtt{f}\ [\phi]\ [\phi.M] \mapsto e}$$

We call such B a branch collection.

Introducing @ symbol Since every branch b_i has the same type, i.e. $[U']{\Leftarrow}[U]{:}[\![\Psi/\psi]\!]$, we can assign the branch collection B a universal type $[U']{\Leftarrow}[U]{:}[\![\Psi/\psi]\!]$. For simplicity, we use @ symbol to relate a branch collection B of type $[U']{\Leftarrow}[U]{:}[\![\Psi/\psi]\!]$ to a contextual object e of type $[\![\Psi/\psi]\!][U]$. For instance,

$$e \mathbin{@} B = \mathtt{rec}^{[U']{\Leftarrow}[U]{:}[\![\Psi/\psi]\!]}(e,\, B)$$

The typing rule and evaluation rule for `rec` can be expressed by @ as:

$$\frac{\Delta;\Gamma \vdash B : [U']{\Leftarrow}[U]{:}[\![\Psi/\psi]\!] \quad \Delta;\Gamma \vdash e : [\![\Psi/\psi]\!][U']}{\Delta;\Gamma \vdash e \mathbin{@} B : [\![\Psi/\psi]\!][U]}\ \text{Tat}$$

$$\frac{e \longrightarrow e'}{e \mathbin{@} B \longrightarrow e' \mathbin{@} B}\ \text{Eat}$$

Note that there is no evaluation rule for a single branch collection.

Type Size Since the definition of reducibility candidates is given inductively on types, we need to give a formal definition of the size of types (including computation-level types, meta-level types, and term abstraction types).
Size of data-level types

$$\mathtt{size}(U) := 0$$

Size of computation-level types

$$\begin{aligned}
\mathtt{size}([U]) &:= 1\\
\mathtt{size}([U']{\Leftarrow}[U]{:}[\![\Psi/\psi]\!]) &:= 1\\
\mathtt{size}(\tau_1 \to \tau_2) &:= 1 + \mathtt{size}(\tau_1) + \mathtt{size}(\tau_2)\\
\mathtt{size}(\Pi\,\phi{:}G.\tau) &:= 1 + \mathtt{size}(\tau)
\end{aligned}$$

Note that the size of Ψ is irrelevant: $\mathtt{size}([\Psi.P]) == 1$

Reducibility Candidates for Terms Finding an apposite definition of reducibility candidates is the essence of the proof. For each closed computation-level term e (or each closed meta-term C, or each closed branch collection B) of type τ (or type U, or type $[U']{\Leftarrow}[U]{:}[\![\Psi/\psi]\!]$), we define a corresponding set $\mathcal{R}_\tau$

(or $\mathcal{R}_U$, or $\mathcal{R}_{[U']\Leftarrow[U]:[\![\Psi/\psi]\!]}$) consisting of some closed terms of the same type. We call such set $\mathcal{R}$ a reducibility candidate for corresponding types. Our definitions are inspired by the notion of continuation from Lindley and Stark [2005]. However, Lindley and Stark only consider modal types in their work. We give a complete definition including contextual types and dependent context types. The definition of reducibility candidates is given inductively on types.

Meta-Type	$\mathcal{R}_U := \{C \mid \cdot \vdash C : U\}$
Branch Collection	$\mathcal{R}_{[U']\Leftarrow[U]:[\![\Psi/\psi]\!]} := \{B \mid \cdot;\cdot \vdash B : [U']\Leftarrow[U]:[\![\Psi/\psi]\!]$ and $\forall C \in \mathcal{R}_{[\![\Psi/\psi]\!]U}.[C] \mathbin{@} B$ halts$\}$
Contextual Type	$\mathcal{R}_{[\Psi.P]} := \{e \mid \cdot;\cdot \vdash e : [\Psi.P]$ and e halts and $\forall B \in \mathcal{R}_{[\psi.P]\Leftarrow[U]:[\![\Psi/\psi]\!]}.e \mathbin{@} B$ halts$\}$
Context Schema	$\mathcal{R}_{[G]} := \{[\Psi] \mid \cdot \vdash \Psi : G\}$
Function Type	$\mathcal{R}_{\tau_2\to\tau} := \{e \mid \cdot;\cdot \vdash e : \tau_2 \to \tau$ and e halts and $\forall e_2 \in \mathcal{R}_{\tau_2}.e\ e_2 \in \mathcal{R}_\tau\}$
Context Abstraction	$\mathcal{R}_{\Pi\psi:G.\tau} := \{e \mid \cdot;\cdot \vdash e : \Pi\psi{:}G.\tau$ and e halts and $\forall \Psi \in \mathcal{R}_G.e\ [\Psi] \in \mathcal{R}_{[\![\Psi/\psi]\!]\tau}\}$

Note that we denote the reducibility candidate for a meta-level type as $\mathcal{R}_{\Psi.P}$ which is different from the denotation of a computation-level type (i.e. $\mathcal{R}_{[\Psi.P]}$). Also note that in the definition of meta-abstractions $\mathcal{R}_{\Pi\psi:G.\tau}$, the size of type $\Pi\psi{:}G.\tau$ is $1+\mathtt{size}(\tau)$, which is always greater than $\mathtt{size}(\tau)$ (i.e. the size of type $[\![\Psi/\psi]\!]\tau$). Hence the definition above is reasonable.

Reducibility Candidates for Substitutions Reducibility candidates can be extended to substitutions (meta-substitution θ, computation-level substitution ρ) and contexts (meta-context Δ, computation-level context Γ). Definitions are given as follows:
Computation-level Suppose $\Gamma = y_1{:}\tau_1, \cdots, y_n{:}\tau_n$ and $\rho = e_1/y_1, \cdots, e_n/y_n$ is a computation-level substitution for the context Γ, then

Computation-level Contexts $\quad \mathcal{R}_\Gamma := \{\rho \mid \cdot;\cdot \vdash \rho : \Gamma$ and $e_i \in \mathcal{R}_{\tau_i}$ for $i = 1, \cdots, n\}$

Meta-level Suppose $\Delta = X_1{:}U_1, \cdots, X_n{:}U_n$ and $\theta = C_1/X_1, \cdots, C_n/X_n$ is a meta-level substitution for the meta-context Δ, then

Meta-level Contexts $\qquad \mathcal{R}_\Delta := \{\theta \mid \cdot \vdash \theta : \Delta\}$

3.2 Auxiliary Lemmas

Lemma 3.1 [Weak Normalizing]
If a computation-level term $e \in \mathcal{R}_\tau$, then e halts.

Proof. Immediately from definitions of reducibility candidates for computation-level terms above.

Lemma 3.2 [Halting]
If a computation-level term $e \longrightarrow e'$, then e halts if and only if e' halts

Proof. Immediately from the determinacy of evaluation and definitions of weak normalization.

Lemma 3.3 [Backward closed]
If $\cdot;\cdot \vdash e : \tau$ and $e \longrightarrow e'$ and $e' \in \mathcal{R}_\tau$, then $e \in \mathcal{R}_\tau$.

Proof. By induction on the structure of type τ.

Contextual Type $[\Psi.P]$

$\cdot;\cdot \vdash e : [\Psi.P]$	by assumption
$e' \in \mathcal{R}_{[\Psi.P]}$	by assumption
e' halts	by def. of $\mathcal{R}_{[\Psi.P]}$
e halts	by lemma
$\forall B \in \mathcal{R}_{[\psi.P] \Leftarrow [U']:[\![\Psi/\psi]\!]}.e' \mathbin{@} B$ halts	by def. of $\mathcal{R}_{[\Psi.P]}$
$e \longrightarrow e'$	by assumption
$e \mathbin{@} B \longrightarrow e' \mathbin{@} B$	by evaluation rule
$e \mathbin{@} B$ halts	by lemma
$e \in \mathcal{R}_{[\Psi.P]}$	by definition of $\mathcal{R}_{[\Psi.P]}$

Context Schema Type $[G]$
Since a term of type $[G]$ does not step, this case is trivial.

Context Abstraction Type $\Pi\psi{:}G.\tau$

$\cdot;\cdot \vdash e : \Pi\psi{:}G.\tau$	by assumption
$e' \in \mathcal{R}_{\Pi\psi:G.\tau}$	by assumption
e' halts	by def. of $\mathcal{R}_{\Pi\psi:G.\tau}$
e halts	by lemma
$\forall \Psi \in \mathcal{R}_G.e'\ [\Psi] \in \mathcal{R}_{[\![\Psi/\psi]\!]\tau}$	by def. of $\mathcal{R}_{\Pi\psi:G.\tau}$
$e \longrightarrow e'$	by assumption
$e\ [\Psi] \longrightarrow e'\ [\Psi]$	by evaluation rule
$\cdot \vdash \Psi : G$	by def. of $\mathcal{R}_G$
$\cdot;\cdot \vdash e\ [\Psi] : [\![\Psi/\psi]\!]\tau$	by typing rule
$e\ [\Psi] \in \mathcal{R}_{[\![\Psi/\psi]\!]\tau}$	by i.h. on $[\![\Psi/\psi]\!]\tau$
$e \in \mathcal{R}_{\Pi\psi:G.\tau}$	by definition of $\mathcal{R}_{\Pi\psi:G.\tau}$

Function Type $\tau_1 \to \tau_2$
classical proof

Lemma 3.4 [Fundamental Lemma]
If $\Delta;\Gamma \vdash e : \tau$ and $\theta \in \mathcal{R}_\Delta, \rho \in \mathcal{R}_{[\![\theta]\!]\Gamma}$ then $[\rho]([\![\theta]\!]e) \in \mathcal{R}_{[\![\theta]\!]\tau}$.

Proof. By induction on the computation-level typing derivation.

Case $\mathcal{D} = \dfrac{\Delta, \psi{:}G; \Gamma \vdash e : \tau}{\Delta; \Gamma \vdash \Lambda\psi.e : \Pi\psi{:}G.\tau}$ Tmabs

$\theta \in \mathcal{R}_\Delta$	by assumption
$\cdot \vdash \theta : \Delta$	by definition of $\mathcal{R}_\Delta$
$\cdot; [\![\theta]\!]\Gamma \vdash [\![\theta]\!](\Lambda\psi.e) : [\![\theta]\!](\Pi\psi{:}G.\tau)$	by substitution lemma
$\rho \in \mathcal{R}_{[\![\theta]\!]\Gamma}$	by assumption
$\cdot; \cdot \vdash \rho : [\![\theta]\!]\Gamma$	by def. of $\mathcal{R}_{[\![\theta]\!]\Gamma}$
$\cdot; \cdot \vdash [\rho][\![\theta]\!](\Lambda\psi.e) : [\![\theta]\!](\Pi\psi{:}G.\tau)$	by substitution lemma
$\cdot; \cdot \vdash \Lambda\psi.[\rho][\![\theta, \psi/\psi]\!]e : \Pi\psi{:}G.[\![\theta, \psi/\psi]\!]\tau$	by substitution property
$\Lambda\psi.[\rho][\![\theta, \psi/\psi]\!]e$ halts	by def. of halts
let $\Psi \in \mathcal{R}_G$ be arbitrary	
$\cdot \vdash \Psi : G$	by def. of $\mathcal{R}_G$
$\cdot \vdash \theta, \Psi/\psi : \Delta, \psi{:}G$	by typing rule
$\theta, \Psi/\psi \in \mathcal{R}_{\Delta,\psi:G}$	by definition of $\mathcal{R}_{\Delta,\psi:G}$
$\rho \in \mathcal{R}_{[\![\theta,\Psi/\psi]\!]\Gamma}$	since Γ is independent of ψ
$[\rho][\![\theta, \Psi/\psi]\!]e \in \mathcal{R}_{[\![\theta,\Psi/\psi]\!]\tau}$	by i.h.
$[\![\Psi/\psi]\!]([\rho][\![\theta, \psi/\psi]\!]e) \in \mathcal{R}_{[\![\Psi/\psi]\!]([\![\theta,\psi/\psi]\!]\tau)}$	by substitution property
$(\Lambda\psi.[\rho][\![\theta, \psi/\psi]\!]e)\ [\Psi] \longrightarrow [\![\Psi/\psi]\!]([\rho][\![\theta, \psi/\psi]\!]e)$	by evaluation rule
$\cdot; \cdot \vdash (\Lambda\psi.[\rho][\![\theta, \psi/\psi]\!]e)\ [\Psi] : [\![\Psi/\psi]\!]([\![\theta, \psi/\psi]\!]\tau)$	by typing rule
$(\Lambda\psi.[\rho][\![\theta, \psi/\psi]\!]e)\ [\Psi] \in \mathcal{R}_{[\![\Psi/\psi]\!]([\![\theta,\psi/\psi]\!]\tau)}$	by backward closed lemma
$\Lambda\psi.[\rho][\![\theta, \psi/\psi]\!]e \in \mathcal{R}_{\Pi\psi:G.[\![\theta,\psi/\psi]\!]\tau}$	by definition of $\mathcal{R}_{\Pi\psi:G.[\![\theta,\psi/\psi]\!]\tau}$
$[\rho][\![\theta]\!](\Lambda\psi.e) \in \mathcal{R}_{[\![\theta]\!](\Pi\psi:G.\tau)}$	by substitution property

Case $\mathcal{D} = \dfrac{\Delta; \Gamma \vdash e : \Pi\psi{:}G.\tau \qquad \Delta \vdash \Psi : G}{\Delta; \Gamma \vdash e\ [\Psi] : [\![\Psi/\psi]\!]\tau}$ Tmapp

$\theta \in \mathcal{R}_\Delta$	by assumption
$\rho \in \mathcal{R}_{[\![\theta]\!]\Gamma}$	by assumption
$[\rho][\![\theta]\!]e \in \mathcal{R}_{[\![\theta]\!](\Pi\psi:G.\tau)}$	by i.h.
$[\rho][\![\theta]\!]e \in \mathcal{R}_{\Pi\psi:G.[\![\theta,\psi/\psi]\!]\tau}$	by substitution property
$\cdot \vdash \theta : \Delta$	by definition of $\mathcal{R}_\Delta$
$\cdot \vdash [\![\theta]\!]\Psi : [\![\theta]\!]G$	by substitution lemma
$\cdot \vdash [\![\theta]\!]\Psi : G$	by substitution property
$[\![\theta]\!]\Psi \in \mathcal{R}_G$	by def. of $\mathcal{R}_G$
$([\rho][\![\theta]\!]e)\ [[\![\theta]\!]\Psi] \in \mathcal{R}_{[\![([\![\theta]\!]\Psi)/\psi]\!]([\![\theta,\psi/\psi]\!]\tau)}$	by def. of $\mathcal{R}_{\Pi\psi:G.[\![\theta,\psi/\psi]\!]\tau}$
$[\rho][\![\theta]\!](e\ [\Psi]) \in \mathcal{R}_{[\![\theta]\!]([\![\Psi/\psi]\!]\tau)}$	by substitution property

Case $\mathcal{D} = \dfrac{\Delta \vdash C : U}{\Delta; \Gamma \vdash [C] : [U]}$ Tmeta

Subcase $\mathcal{D} = \dfrac{\Delta \vdash \Psi : G}{\Delta; \Gamma \vdash [\Psi] : [G]}$ Tmeta

$\theta \in \mathcal{R}_\Delta$	by assumption
$\cdot \vdash \theta : \Delta$	by definition of $\mathcal{R}_\Delta$
$\cdot \vdash [\![\theta]\!]\Psi : G$	by substitution lemma
$\cdot; \cdot \vdash [[\![\theta]\!]\Psi] : [G]$	by evaluation rule

$[[\theta]\Psi] \in \mathcal{R}_{[G]}$ by definition
$[\rho][\theta][\Psi] \in \mathcal{R}_{[[\theta][G]}$ by substitution property

$$\textit{Subcase } \mathcal{D} = \frac{\Delta \vdash \hat{\Psi}.M : \Psi.P}{\Delta; \Gamma \vdash [\hat{\Psi}.M] : [\Psi.P]}\ \text{Tmeta}$$

$\theta \in \mathcal{R}_\Delta$ by assumption
$\cdot \vdash \theta : \Delta$ by definition of $\mathcal{R}_\Delta$
$\cdot \vdash [[\theta](\hat{\Psi}.M) : [[\theta](\Psi.P)$ by substitution lemma
$\cdot; \cdot \vdash [[[\theta](\hat{\Psi}.M)] : [[[\theta](\Psi.P)]$ by typing rule
$[[\theta](\hat{\Psi}.M) \in \mathcal{R}_{[[\theta]\Psi.P}$ by def. of $\mathcal{R}_{[[\theta]\Psi.P}$
$[[\theta](\hat{\Psi}.M)]$ halts by def. of halts
Let $B \in \mathcal{R}_{[\psi.P] \Leftarrow [U']:[[\Psi/\psi]]}$ be arbitrary
$[[\theta](\hat{\Psi}.M)] \mathbin{@} B$ halts by def. of $\mathcal{R}_{[\psi.P] \Leftarrow [U']:[[\Psi/\psi]]}$
$[[\theta](\hat{\Psi}.M)] \in \mathcal{R}_{[[\theta]\Psi.P]}$ by def. of $\mathcal{R}_{[[\theta]\Psi.P]}$
$[\rho][\theta][(\hat{\Psi}.M)] \in \mathcal{R}_{[[\theta][\Psi.P]}$ by substitution property

$$\textit{Case } \mathcal{D} = \frac{\Gamma(y) = \tau}{\Delta; \Gamma \vdash y : \tau}\ \text{Tvar}$$

$\theta \in \mathcal{R}_\Delta$ by assumption
$\cdot \vdash \theta : \Delta$ by definition of $\mathcal{R}_\Delta$
$\cdot; [[\theta]\Gamma \vdash [[\theta]y : [[\theta]\tau$ by substitution lemma
$\cdot; [[\theta]\Gamma \vdash y : [[\theta]\tau$ by substitution property
$\rho \in \mathcal{R}_{[[\theta]\Gamma}$ by assumption
$[\rho]y \in \mathcal{R}_{[[\theta]\tau}$ by def. of $\mathcal{R}_{[[\theta]\Gamma}$
$[\rho][\theta]y \in \mathcal{R}_{[[\theta]\tau}$ by substitution property

$$\textit{Case } \mathcal{D} = \frac{\Delta; \Gamma \vdash e_1 : [\Psi.P] \qquad \Delta, u{:}\Psi.P; \Gamma \vdash e_2 : \tau}{\Delta; \Gamma \vdash \texttt{let}\ u = e_1\ \texttt{in}\ e_2 : \tau}\ \text{Tlet}$$

$\theta \in \mathcal{R}_\Delta$ by assumption
$\rho \in \mathcal{R}_{[[\theta]\Gamma}$ by assumption
$[\rho][\theta]e_1 \in \mathcal{R}_{[[\theta][\Psi.P]}$ by i.h.
$\cdot; \cdot \vdash [\rho][\theta]e_1 : [[\theta][\Psi.P]$ by def. of $\mathcal{R}_{[[\theta][\Psi.P]}$
$[\rho][\theta]e_1$ halts by def. of $\mathcal{R}_{[[\theta][\Psi.P]}$
suppose $[\rho][\theta]e_1 \longrightarrow^* v$
$\cdot; \cdot \vdash v : [[\theta][\Psi.P]$ by type preservation
$\cdot; \cdot \vdash v : [[[\theta]\Psi.P]$ by substitution property
let $\Psi' = [[\theta]\Psi$
$v = [\hat{\Psi}'.M]$ by canonical forms lemma
$\cdot \vdash \hat{\Psi}'.M : [[\theta]\Psi.P$ by typing inversion
$\cdot \vdash \theta : \Delta$ by def. of $\mathcal{R}_\Delta$
$\cdot \vdash \theta, \hat{\Psi}'.M/u : \Delta, u{:}\Psi.P$ by typing rule
$\theta, \hat{\Psi}'.M/u \in \mathcal{R}_{\Delta, u:\Psi.P}$ by def. of $\mathcal{R}_{\Delta,u:\Psi.P}$
$\rho \in \mathcal{R}_{[[\theta,\hat{\Psi}'.M/u]]\Gamma}$ Γ is independent of u
$[\rho][[\theta, \hat{\Psi}'.M/u]]e_2 \in \mathcal{R}_{[[\theta,\hat{\Psi}'.M/u]]\tau}$ by i.h.

$\cdot;\cdot \vdash [\rho][\![\theta]\!](\texttt{let}\ u = e_1\ \texttt{in}\ e_2) : [\![\theta]\!]\tau$ by substitution lemma
$\texttt{let}\ u = [\rho][\![\theta]\!]e_1\ \texttt{in}\ [\rho][\![\theta, u/u]\!]e_2 \longrightarrow^* [\rho][\![\theta, \hat{\Psi}'.M/u]\!]e_2$ by evaluation rule
$\texttt{let}\ u = [\rho][\![\theta]\!]e_1\ \texttt{in}\ [\rho][\![\theta, u/u]\!]e_2 \in \mathcal{R}_{[\![\theta,\hat{\Psi}'.M/u]\!]\tau}$ by backward closed lemma
$[\rho][\![\theta]\!](\texttt{let}\ u = e_1\ \texttt{in}\ e_2) \in \mathcal{R}_{[\![\theta]\!]\tau}$ by substitution property

Case

$$\frac{\begin{array}{l}\mathcal{S}{=}\texttt{split}(\psi{:}G \vdash \psi.P){=}\{(\phi{:}G, \Delta_i' \vdash p_i : \phi.P, \overrightarrow{\texttt{f}\ [\Phi^i]\ [q^i]}) \mid \phi{:}G, \Delta_i' \vdash q_j^i : \Phi_j^i.P\} \\ b_i = \Pi\,\phi{:}G, \Delta_i'.\Pi\,\overrightarrow{\texttt{f}\ [\Phi^i]\ [q^i]{:}[\![\Phi^i/\psi]\!]\tau}.\texttt{f}\ [\phi]\ [p_i] \mapsto e_i \qquad |\mathcal{S}| = |\overrightarrow{b}| \\ \Delta; \Gamma \vdash e : [\![\Psi/\psi]\!][U'] \quad \text{for all i } \Delta; \Gamma \vdash b_i : [U'] {\Leftarrow} [U]{:}[\![\Psi/\psi]\!]\end{array}}{\Delta; \Gamma \vdash \texttt{rec}^{[U'] \Leftarrow [U]:[\![\Psi/\psi]\!]}(e, \overrightarrow{b}) : [U]}\ \text{Trec}$$

$$\frac{\begin{array}{l}\phi{:}G, \Delta_i' \vdash p_i : [\![\Psi_i/\psi]\!]U' \qquad \text{for all j } \phi{:}G, \Delta_i' \vdash q_j^i : [\![\Phi_j^i/\psi]\!]U' \\ \phi{:}G, \Delta_i', [\![\Psi_i/\psi]\!]\Delta; [\![\Psi_i/\psi]\!]\Gamma, \overrightarrow{\texttt{f}\ [\Phi^i]\ [q^i] : [\![\Phi^i/\psi]\!][U]} \vdash e_i : [\![\Psi_i/\psi]\!][U]\end{array}}{\Delta; \Gamma \vdash \Pi\,\phi{:}G, \Delta_i'.\Pi\,\overrightarrow{\texttt{f}\ [\Phi^i]\ [q^i]{:}[\![\Phi^i/\psi]\!][U]}.\texttt{f}\ [\Psi_i]\ [p_i] \mapsto e_i : [U'] {\Leftarrow} [U]{:}[\![\Psi/\psi]\!]}\ \text{Tb}$$

We assume the meta-context contains a context variable ψ, i.e. $\psi{:}G, \Delta$. Then the context substitution in the type annotation just becomes $[\![\psi/\psi]\!]$ and we write $[U']$ as $[\psi.P]$. The proof for the concrete meta-context is analogous.
$\theta \in \mathcal{R}_{\psi:G,\Delta}$ by assumption
$\cdot \vdash \theta : \psi{:}G, \Delta$ by def.
Without loss of generality, we can write the substitution for the context variable ψ explicitly in θ. i.e.
$\theta = \Psi/\psi, \theta'$
It is easy to verify that $\cdot \vdash \Psi/\psi, \theta' : \psi{:}G, \Delta$
Then we apply the substitution to the above derivation and obtain the following by substitution lemma:
$\cdot; [\![\theta]\!]\Gamma \vdash \texttt{rec}^{[\psi.P] \Leftarrow [U]:[\![\Psi/\psi]\!]}([\![\theta]\!]e,$
$\overrightarrow{\Pi\,\phi{:}G, \Delta'.\Pi\,\overrightarrow{\texttt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U]}.\texttt{f}\ [\phi]\ [\phi.M] \mapsto [\![\texttt{id}(\phi{:}G, \Delta'), \theta']\!]e'}) : [\![\theta]\!][U]$
$\rho \in \mathcal{R}_{[\![\theta]\!]\Gamma}$ by assumption
$\cdot; \cdot \vdash \rho : [\![\theta]\!]\Gamma$ by def.
$\cdot; \cdot \vdash \texttt{rec}^{[\psi.P] \Leftarrow [U]:[\![\Psi/\psi]\!]}([\rho][\![\theta]\!]e,$
$\overrightarrow{\Pi\,\phi{:}G, \Delta'.\Pi\,\overrightarrow{\texttt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U]}.\texttt{f}\ [\phi]\ [\phi.M] \mapsto [\rho][\![\texttt{id}(\phi{:}G, \Delta'), \theta']\!]e'}) : [\![\theta]\!][U]$
by substitution lemma
For simplicity, we write

$$\overrightarrow{b} = \overrightarrow{\Pi\,\phi{:}G, \Delta'.\Pi\,\overrightarrow{\texttt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U]}.\texttt{f}\ [\phi]\ [\phi.M] \mapsto e'}$$

$$\overrightarrow{[\rho][\![\theta]\!]b} = \overrightarrow{\Pi\,\phi{:}G, \Delta'.\Pi\,\overrightarrow{\texttt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U]}.\texttt{f}\ [\phi]\ [\phi.M] \mapsto [\rho][\![\texttt{id}(\phi{:}G, \Delta), \theta']\!]e'}$$

$[\rho][\![\theta]\!]e \in \mathcal{R}_{[\![\Psi/\psi,\theta']\!][\psi.P]}$ by i.h.
$[\rho][\![\theta]\!]e \in \mathcal{R}_{[\Psi.P]}$ by substitution property
$[\rho][\![\theta]\!]e$ halts by def.

Suppose $[\rho]\llbracket\theta\rrbracket e \longrightarrow^* v$ for some value v

$\cdot;\cdot \vdash [\rho]\llbracket\theta\rrbracket e : [\Psi.P]$ by def.

$\cdot;\cdot \vdash v : [\psi.P]$ by preservation

$v = [\hat{\Psi}.M]$ by canonical forms lemma

$\cdot;\Psi \vdash M : P$ by inversion Tbox

We then proceed by inner induction on the structure of M

Subcase $M = c$ (M is a data-level constant)

$\cdot;\cdot \vdash \mathtt{rec}^{[U']\Leftarrow[U]:\llbracket\Psi/\psi\rrbracket}([\rho]\llbracket\theta\rrbracket e, \overrightarrow{[\rho]\llbracket\theta\rrbracket b}) : \llbracket\theta\rrbracket[U]$ by previous lines

$\cdot;\cdot \vdash \mathtt{rec}^{[U']\Leftarrow[U]:\llbracket\Psi/\psi\rrbracket}([\hat{\Psi}.c], \overrightarrow{[\rho]\llbracket\theta\rrbracket b}) : \llbracket\theta\rrbracket[U]$ by type preservation

$\exists$ a unique branch $[\rho]\llbracket\theta\rrbracket b_i = \Pi\,\phi{:}G.\mathtt{f}\ [\phi]\ [\phi.c] \mapsto [\rho]\llbracket\phi/\phi, \theta\rrbracket e_i$ by type system

$b_i = \Pi\,\phi{:}G.\mathtt{f}\ [\phi]\ [\phi.c] \mapsto e_i$ by substitution

$\Delta;\Gamma \vdash \Pi\,\phi{:}G.\mathtt{f}\ [\phi]\ [\phi.c] \mapsto e_i : [\psi.P]\Leftarrow[U]:\llbracket\psi/\psi\rrbracket$ by typing inversion

$\phi{:}G, \llbracket\phi/\psi\rrbracket\Delta; \llbracket\phi/\psi\rrbracket\Gamma \vdash e_i : \llbracket\phi/\psi\rrbracket[U]$ by typing inversion

$\mathtt{rec}^{[\psi.P]\Leftarrow[U]:\llbracket\Psi/\psi\rrbracket}([\rho]\llbracket\theta\rrbracket e, \overrightarrow{[\rho]\llbracket\theta\rrbracket b}) \longrightarrow^* \llbracket\theta_i\rrbracket([\rho]\llbracket\phi/\phi, \theta\rrbracket e_i)$ by evaluation rule

$\phi{:}G \vdash \hat{\Psi}.c \doteq \phi.c/\theta_i$ by evaluation inversion

$\cdot \vdash \theta_i : \phi{:}G$ by soundness

$\llbracket\theta_i\rrbracket(\phi.c) = \hat{\Psi}.c$ by soundness

$\theta_i = \Psi/\phi$ by soundness

$\cdot \vdash \Psi/\phi, \theta : \phi{:}G, \llbracket\phi/\psi\rrbracket\Delta$ by renaming

$\Psi/\phi, \theta \in \mathcal{R}_{\phi:G, \llbracket\phi/\psi\rrbracket\Delta}$ by def.

$\llbracket\Psi/\phi, \theta'\rrbracket\llbracket\phi/\psi\rrbracket$
$= \llbracket\Psi/\psi, \theta'\rrbracket$
$= \llbracket\theta\rrbracket$ by substitution property

$\rho \in \mathcal{R}_{\llbracket\Psi/\phi,\theta'\rrbracket\llbracket\phi/\psi\rrbracket\Gamma}$ since $\llbracket\theta\rrbracket\Gamma = \llbracket\Psi/\phi, \theta'\rrbracket\llbracket\phi/\psi\rrbracket\Gamma$

$[\rho]\llbracket\Psi/\phi, \theta'\rrbracket e_i \in \mathcal{R}_{\llbracket\Psi/\phi,\theta'\rrbracket(\llbracket\phi/\psi\rrbracket[U])}$ by i.h.

$[\rho]\llbracket\Psi/\phi, \theta'\rrbracket e_i \in \mathcal{R}_{\llbracket\theta\rrbracket[U]}$ by substitution equality

$\mathtt{rec}^{[\Psi.P]\Leftarrow\llbracket\theta\rrbracket[U]}([\rho]\llbracket\theta\rrbracket e, \overrightarrow{[\rho]\llbracket\theta\rrbracket e}) \longrightarrow^* \llbracket\Psi/\phi\rrbracket[\rho]\llbracket\phi/\phi, \theta'\rrbracket e_i$ by evaluation

$[\rho]\llbracket\theta\rrbracket\mathtt{rec}^{[\psi.P]\Leftarrow[U]:\llbracket\psi/\psi\rrbracket}(e, \overrightarrow{b}) \longrightarrow^* [\rho]\llbracket\Psi/\phi, \theta'\rrbracket e_i$ by substitution

$[\rho]\llbracket\theta\rrbracket\mathtt{rec}^{[\psi.P]\Leftarrow[U]:\llbracket\psi/\psi\rrbracket}(e, \overrightarrow{b}) \in \mathcal{R}_{\llbracket\theta\rrbracket[U]}$ by backward closed lemma

Subcase $M = x$ (M is a data-level variable)

$\cdot;\cdot \vdash \mathtt{rec}^{[\psi.P]\Leftarrow[U]:\llbracket\Psi/\psi\rrbracket}([\rho]\llbracket\theta\rrbracket e, \overrightarrow{[\rho]\llbracket\theta\rrbracket b}) : \llbracket\theta\rrbracket[U]$ by previous lines

$\cdot;\cdot \vdash \mathtt{rec}^{[\psi.P]\Leftarrow[U]:\llbracket\Psi/\psi\rrbracket}([\hat{\Psi}.x], \overrightarrow{[\rho]\llbracket\theta\rrbracket b}) : \llbracket\theta\rrbracket[U]$ by preservation

$\exists$ a unique branch $[\rho]\llbracket\theta\rrbracket b_i = \Pi\,\phi{:}G, p{:}\#\phi.P.\mathtt{f}\ [\phi]\ [\phi.p[\mathtt{id}_\phi]] \mapsto [\rho]\llbracket\phi/\phi, p/p, \theta'\rrbracket e_i$
by type system

$b_i = \Pi\,\phi{:}G, p{:}\#\phi.P.\mathtt{f}\ [\phi]\ [\phi.p[\mathtt{id}_\phi]] \mapsto e_i$ by substitution

$\Delta;\Gamma \vdash \Pi\,\phi{:}G, p{:}\#\phi.P.\mathtt{f}\ [\phi]\ [\phi.p[\mathtt{id}_\phi]] \mapsto e_i : [\psi.P]\Leftarrow[U]:\llbracket\psi/\psi\rrbracket$ by type inversion

$\phi{:}G, p{:}\#\phi.P, \llbracket\phi/\psi\rrbracket\Delta; \llbracket\phi/\psi\rrbracket\Gamma \vdash e_i : \llbracket\phi/\psi\rrbracket[U]$ by type inversion

$\mathtt{rec}^{[\psi.P]\Leftarrow[U]:\llbracket\Psi/\psi\rrbracket}([\rho]\llbracket\theta\rrbracket e, \overrightarrow{[\rho]\llbracket\theta\rrbracket b}) \longrightarrow^* \llbracket\theta_i\rrbracket([\rho]\llbracket\phi/\phi, p/p, \theta'\rrbracket e_i)$ by eval rule

$\phi{:}G, p{:}\#\phi.P \vdash \hat{\Psi}.x \doteq \phi.p[\mathtt{id}_\phi]/\theta_i$ by evaluation inversion

$\cdot \vdash \theta_i : \phi{:}G, p{:}\#\phi.P$ by soundness

$\llbracket\theta_i\rrbracket(\phi.p[\mathtt{id}_\phi]) = \hat{\Psi}.x$ by soundness

$\theta_i = \Psi/\phi, \Psi.x/p$ by soundness

$\cdot \vdash \Psi/\phi, \Psi.x/p, \theta' : \phi{:}G, p{:}\#\phi.P, [\![\phi/\psi]\!]\Delta$ by typing rule
$\Psi/\phi, \Psi.x/p, \theta' \in \mathcal{R}_{\phi:G,p:\#\phi.P,[\![\phi/\psi]\!]\Delta}$ by def.
$\rho \in \mathcal{R}_{[\![\Psi/\phi,\Psi.x/p,\theta']\!][\![\phi/\psi]\!]\Gamma}$ since $[\![\theta]\!]\Gamma = [\![\Psi/\phi, \Psi.x/p, \theta']\!][\![\phi/\psi]\!]\Gamma$
$[\rho][\![\Psi/\phi, \Psi.x/p, \theta']\!]e_i \in \mathcal{R}_{[\![\Psi/\phi,\Psi.x/p,\theta']\!][\![\phi/\psi]\!][U]}$ by i.h.
$[\rho][\![\Psi/\phi, \Psi.x/p, \theta']\!]e_i \in \mathcal{R}_{[\![\theta]\!][U]}$ by substitution property
$[\rho][\![\theta]\!]\mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![\psi/\psi]\!]}(e, \overrightarrow{b}) \longrightarrow^* [\rho][\![\Psi/\phi, \Psi.x/p, \theta']\!]e_i$ by evaluation
$[\rho][\![\theta]\!]\mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![\psi/\psi]\!]}(e, \overrightarrow{b}) \in \mathcal{R}_{[\![\theta]\!][U]}$ by backward closed lemma

Subcase $M = c\ M_1 \cdots M_k$ (where c is a constructor)
we can assume $\Sigma(c) = A_1 \to A_2 \to \cdots \to A_k \to P$
$\cdot; \cdot \vdash \mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![\Psi/\psi]\!]}([\rho][\![\theta]\!]e, \overrightarrow{[\rho][\![\theta]\!]b}) : [\![\theta]\!][U]$ by previous lines
$\cdot; \cdot \vdash \mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![\Psi/\psi]\!]}([\hat{\Psi}.c\ M_1 \cdots M_k], \overrightarrow{[\rho][\![\theta]\!]b}) : [\![\theta]\!][U]$ by preservation
$\exists$ a unique $[\rho][\![\theta]\!]b_i = \Pi\, \phi{:}G, \Delta_i.\Pi\, \overrightarrow{\mathtt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U].\mathtt{f}\ [\phi]\ [\phi.c\ M'_1 \cdots M'_k] \mapsto [\rho][\![\phi/\phi, \mathtt{id}(\Delta_i), \theta']\!]e_i}$ by type system
where $\Delta_i = u_1{:}(\phi, \Psi_1.P_1), u_2{:}(\phi, \Psi_2.P_2) \cdots u_k{:}(\phi, \Psi_k.P_k)$ and M'_j contains u_j for each j, i.e. the type of each meta-variable u_j is $(\phi, \Psi_j.P_j)$
$\overrightarrow{\mathtt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U]}$ is a complete and non-redundant set of all well-formed primitive recursive calls
$b_i = \Pi\, \phi{:}G, \Delta_i.\Pi\, \overrightarrow{\mathtt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U].\mathtt{f}\ [\phi]\ [\phi.c\ M'_1 \cdots M'_k]} \mapsto e_i$ by substitution
$\Delta; \Gamma \vdash \Pi\, \phi{:}G, \Delta_i.\Pi\, \overrightarrow{\mathtt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U].\mathtt{f}\ [\phi]\ [\phi.c\ M'_1 \cdots M'_k] \mapsto e_i} : [\psi.P]\Leftarrow[U]:[\![\psi/\psi]\!]$ by inversion
$\phi{:}G, \Delta_i, [\![\phi/\psi]\!]\Delta; [\![\phi/\psi]\!]\Gamma, \overrightarrow{\mathtt{f}\ [\Phi]\ [\hat{\Phi}.N]{:}[\![\Phi/\psi]\!][U]} \vdash e_i : [\![\phi/\psi]\!][U]$ by inversion
$\mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![\Psi/\psi]\!]}([\rho][\![\theta]\!]e, \overrightarrow{[\rho][\![\theta]\!]b}) \longrightarrow^* [\![\theta_i]\!][\rho_i]([\rho][\![\phi/\phi, \mathtt{id}(\Delta_i), \theta']\!]e_i)$ by eval
$\phi{:}G, \Delta_i \vdash \hat{\Psi}.c\ M_1 \cdots M_k \doteq \phi.c\ M'_1 \cdots M'_k/\theta_i$ by evaluation inversion
$\cdot \vdash \theta_i : \phi{:}G, \Delta_i$ by soundness
$[\![\theta_i]\!](\phi.c\ M'_1 \cdots M'_k) = \hat{\Psi}.c\ M_1 \cdots M_k$ by soundness
$\theta_i = \Psi/\phi,\ (\Psi, \hat{\Psi}_1.N_1)/u_1,\ \cdots,\ (\Psi, \hat{\Psi}_k.N_k)/u_k = \Psi/\phi, \theta'_i$ ($u_1 \cdots u_k$ refer to variables declared in Δ_i before) by soundness
$\Psi/\phi, \theta'_i \in \mathcal{R}_{\phi:G,\Delta_i}$ by def. of $\mathcal{R}_{\phi:G,\Delta_i}$
$\rho_i = \overrightarrow{\mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![\Psi/\psi]\!]}([\hat{\Phi}.N], \overrightarrow{[\rho][\![\theta]\!]b})/f\ [\Phi]\ [\hat{\Phi}.N]}$ by evaluation rule
$\phi{:}G, \Delta_i; \cdot \vdash \rho_i : \overrightarrow{f\ [\Phi]\ [\hat{\Phi}.N] : [\![\Phi/\psi]\!][U]}$ by soundness
$\phi{:}G, \Delta_i; \cdot \vdash \mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![(\phi,\Psi_j)/\psi]\!]}([\phi, \hat{\Psi}_j.u_j[\mathtt{id}_\phi, \mathtt{id}(\Psi_j)]], \overrightarrow{[\rho][\![\theta]\!]b}) : [\![\phi, \Psi_j/\psi]\!][U]$ for some j by type system
$\cdot; \cdot \vdash [\![\Psi/\phi, \theta'_i]\!]\mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![(\phi,\Psi_j)/\psi]\!]}([\phi, \hat{\Psi}_j.u_j[\mathtt{id}_\phi, \mathtt{id}(\Psi_j)], \overrightarrow{[\rho][\![\theta]\!]b})$
$: [\![\Psi/\phi, \theta'_j]\!][\![\phi, \Psi_j/\psi]\!][U]$ by substitution lemma
$\cdot; \cdot \vdash \mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![(\phi,\Psi_j)/\psi]\!]}([\hat{\Psi}, \hat{\Psi}_j.N_j], \overrightarrow{[\rho][\![\theta]\!]b}) : [\![\Psi, \Psi_j/\psi]\!][U]$ by subs property
$\mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![(\phi,\Psi_j)/\psi]\!]}([\hat{\Psi}, \hat{\Psi}_j.N_j], \overrightarrow{[\rho][\![\theta]\!]b}) \in \mathcal{R}_{[\![\Psi,\Psi_j/\psi]\!][U]}$ by inner i.h. on N_j
$[\![\theta_i]\!]\rho_i \in \mathcal{R}_{[\![\theta_i]\!](\overrightarrow{f\ [\Phi]\ [\hat{\Phi}.N]:[\![\Phi/\psi]\!][U]})}$ by def. and substitution property
$\rho, [\![\theta_i]\!]\rho_i \in \mathcal{R}_{[\![\theta_i,\theta']\!]([\![\phi/\psi]\!]\Gamma, \overrightarrow{f\ [\Phi]\ [\hat{\Phi}.N]:[\![\Phi/\psi]\!][U]})}$ by substitution property
$\cdot \vdash \theta_i, \theta' : \phi{:}G, \Delta_i, [\![\phi/\psi]\!]\Delta$ by typing rule

$\theta_i, \theta' \in \mathcal{R}_{\phi:G,\Delta_i,[\![\phi/\psi]\!]\Delta}$ by def.
$[\![\theta_i, \theta']\!][\rho, [\![\theta_i]\!]\rho_i]e_i \in \mathcal{R}_{[\![\theta_i,\theta']\!][\![\phi/\psi]\!][U]}$ by outer i.h.
$[\![\theta_i]\!][\rho_i][\![\theta']\!][\rho]e_i \in \mathcal{R}_{[\![\theta]\!][U]}$ by substitution property
$[\rho][\![\theta]\!]\mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![\psi/\psi]\!]}(e, \overrightarrow{b}) = \mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![\Psi/\psi]\!]}([\rho][\![\theta]\!]e, \overrightarrow{[\rho][\![\theta]\!]e}) \longrightarrow^* [\![\theta_i]\!][\rho_i][\rho][\![\theta']\!]e_i$
by evaluation
$[\rho][\![\theta]\!]\mathtt{rec}^{[\psi.P]\Leftarrow[U]:[\![\psi/\psi]\!]}(e, \overrightarrow{b}) \in \mathcal{R}_{[\![\theta]\!][U]}$ by backward closed lemma

3.3 Weak Normalization Theorem

If $\cdot; \cdot \vdash e : \tau$ then e halts.

Proof. Taking both empty meta-context Δ and empty ordinary computation-level context Γ, we obtain the theorem directly from the fundamental lemma and lemma 3.1.

4 Conclusion

In this work, we concentrate on the simply-typed fragment of Beluga, a novel powerful language with support for specifying formal systems in LF and reasoning with contextual objects. We develop a well-founded recursion principle over contextual objects, describe a call-by-value small-step semantics, and prove weak normalization. This is an important milestone towards establishing a framework for programming proofs, since it justifies well-founded recursion principles over LF specification. It has a wide range of implications in certified programming, proof-carrying architectures, mechanizing meta-theory of programming languages and so forth.

In future, we plan to explore the following extensions. Firstly, we are interested in extending the simply-typed framework to the dependently-typed setting. We believe our framework scales to dependent types nicely because by permitting contextual types $[\psi.P]$ to depend on context variables ψ, the current language already achieves certain extent of dependency among computation-level types. Therefore, our next step is to expand the dependency to allow both context variables and contextual objects as index objects. The substitution refinement may become a potential complication as we need to take account into how the type of the scrutinee in rec expressions is instantiated. One solution is to incorporate environment-based evaluation model into our operational semantics to support evaluation in context [Cave and Pientka, 2012]. However, the change of the evaluation rule may lead to the modification of the proof of the fundamental lemma. Secondly, we want to progress from primitive recursive functions to more general recursive functions. Thirdly, we are also intrigued by developing a well-founded recursion principle for context variables, which can allow us to write more powerful functions such as converting a de Bruijn term into a HOAS term .

Bibliography

Andrew Cave and Brigitte Pientka. Programming with binders and indexed datatypes. In *39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'12)*, pages 413–424. ACM Press, 2012.

Joëlle Despeyroux and André Hirschowitz. Higher-order abstract syntax with induction in coq. In *Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning*, LPAR '94, pages 159–173, London, UK, UK, 1994. Springer-Verlag. ISBN 3-540-58216-9.

Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. In *Proceedings of the Third International Conference on Typed Lambda Calculus and Applications (TLCA'97)*, pages 147–163. Springer, 1997. Extended version available as Technical Report CMU-CS-96-172, Carnegie Mellon University.

J. Y Girard. *Interprtation fonctionnelle et elimination des coupures de l'arithmtique d'ordre suprieur*. These d'tat, Universit de Paris 7, 1972.

J.-Y. Girard, Y. Lafont, and P. Tayor. *Proofs and types*. Cambridge University Press, 1990.

Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

Sam Lindley and Ian Stark. Reducibility and $\top\top$-lifting for computation types. In *Typed Lambda Calculi and Applications: Proceedings of the Seventh International Conference TLCA 2005*, number 3461 in Lecture Notes in Computer Science, pages 262–277. Springer-Verlag, 2005.

Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.

Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'08)*, pages 371–382. ACM Press, 2008.

Carsten Schürmann and Frank Pfenning. A coverage checking algorithm for LF. In D. Basin and B. Wolff, editors, *Proceedings of the 16th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'03)*, pages 120–135. Springer, 2003.

William Tait. Intensional interpretations of functionals of finite type i. *J. Symb. Log.*, 32(2):198–212, 1967.

Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002.