A

The Formal System $\lambda\delta$ Revised - Stage A: Extending the Applicability Condition

FERRUCCIO GUIDI, Department of Computer Science and Engineering, University of Bologna, Italy

The formal system $\lambda\delta$ is a typed λ-calculus derived from Λ_∞, aiming to support the foundations of Mathematics that require an underlying theory of expressions (for example the Minimal Type Theory).

The system is developed in the context of the Hypertextual Electronic Library of Mathematics as a machine-checked digital specification, that is not the formal counterpart of previous informal material. The first version of the calculus appeared in 2006 and proved unsatisfactory for some reasons.

In this article we present a revised version of the system and we prove three relevant desired properties: the confluence of reduction, the strong normalization of an extended form of reduction, known as the "big tree" theorem, and the preservation of validity by reduction. To our knowledge, we are presenting here the first fully machine-checked proof of the "big tree" theorem for a calculus that includes Λ_∞.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Lambda calculus and related systems*

General Terms: Theory

Additional Key Words and Phrases: Explicit substitutions, extended applicability condition, extended transition system, infinite degrees of terms, preservation of validity, strong normalization, terms as types

ACM Reference Format:
Ferruccio Guidi. 2014. The Formal System $\lambda\delta$ Revised - Stage A: Extending the Applicability Condition. *ACM Trans. Comput. Logic* V, N, Article A (January YYYY), 34 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

The formal system $\lambda\delta$ is a typed λ-calculus aiming to support the foundations of Mathematics that require an underlying theory of expressions (for example mTT of Maietti [2009] and its predecessors). The system is developed in the context of the HELM project of Asperti et al. [2003] as a machine-checked digital specification, that is not the formal counterpart of some previous informal material. The first version of the calculus [Guidi 2006], formalized in the proof management system (p.m.s.) Coq [Coq development team 2002] and published by Guidi [2009], proved unsatisfactory for some reasons. So a revision of the calculus is ongoing since April 2011 and includes a brand new formalization [Guidi 2014] in the p.m.s. Matita of Asperti et al. [2011].

Firstly, the revision aims at this problem: the calculus of Guidi [2009] comes from Λ_∞ [van Benthem Jutting 1994b], a language of the Automath family [Nederpelt et al. 1994], and yet it cannot type every term typed by Λ_∞ since it lacks the "pure" type inference rule for function application [de Bruijn 1991]. If $\Gamma \vdash M : N$ is a type assignment judgment and $\Gamma \vdash M\ !$ is the corresponding validity judgment, this rule states:

$$\frac{\Gamma \vdash f : F \quad \Gamma \vdash F(t)\ !}{\Gamma \vdash f(t) : F(t)}\text{@–pure} \tag{1}$$

DOI:http://dx.doi.org/10.1145/0000000.0000000

This rule is redundant when the terms have three degrees (objects, classes, and sorts) as in Pure Type Systems (PTS's) [Barendregt 1993] and their derivatives. On the contrary it becomes effective when more degrees are available, as in the $\mathrm{Aut}{-}4$ family [de Bruijn 1994b] or in Λ_∞, since $\Gamma \vdash f : F$ and $\Gamma \vdash F : \mathcal{F}$ do not imply that $\mathcal{F}$ is a sort. In this case f can be a function, F a function space, and $\mathcal{F}$ a family of function spaces. If we take t in the domain of f, we might want $\Gamma \vdash f(t)\,!$ even when f and F are given abstractly as variables declared in Γ. Rule (1) is designed to realize this situation.

In the mathematical language we express a large variety of concepts, each with its own requirements. When we translate this language to typed λ-calculus, a widely accepted policy suggests that expressions denoting concepts with different requirements should correspond to λ-terms with different degrees. Consider typical concepts of interest: sets, elements, propositions and proofs. While well-established similarities between elements and proofs support their representation with terms of the same degree, significant differences arise as well, playing in favor of representing them differently.

Mainly, identifying two proofs of a proposition (also known as "proof irrilevance") is sensible, while identifying two elements of a set generally is not. de Bruijn [1994b] approaches this problem by advocating a calculus in which two terms inhabiting the same type of degree 3 are definitionally equal. This is to say that terms of degree 4 are provided for representing irrelevant proofs. Similarly, subtle differences can be found in the requirements for sets and propositions. So it seams that a calculus with many degrees for its terms, may allow flexible interpretations of the mathematical language.

We note that $\lambda\delta$ has a disadvantage in this sense because of its "isotropy", by which we mean that the features of its terms do not depend on their degree.

Secondly, the revised $\lambda\delta$ aims at other improvements some of which were advocated already by Guidi [2009]. Simpler "arities" make the arity assignment judgment decidable for all values of the sort hierarchy parameter. The reaxiomatized step of environment-dependent parallel reduction allows to remove the substitution operator and provides for the long-awaited Rule (2). Tait-style reducibility candidates [Tait 1975] in place of Girard-style ones [Girard et al. 1989] simplify the strong normalization theorem. Simpler environments allow to remove some ancillary operators.

$$\frac{\Gamma \vdash f_1 \rightrightarrows f_2 \quad \Gamma \vdash t_1 \rightrightarrows t_2}{\Gamma \vdash f_1(t_1) \rightrightarrows f_2(t_2)}\,\mathrm{appl} \tag{2}$$

The main contributions of this article are the so-called "big tree" theorem [de Vrijer 1994] for $\lambda\delta$, which yields the subject reduction theorem for its stratified validity.

The "big tree" theorem states that valid terms are strongly normalizing with respect to a relation comprising reduction steps, type steps, subtraction steps, and more. It generalizes ordinary strong normalization and gives a very powerful induction principle for proving properties on valid terms. We are confident that this tool may prove useful in systems other than $\lambda\delta$ as well.

Stratified validity (*i.e.*, validity up to a specified degree) replaces type assignment as a primitive notion in the revised $\lambda\delta$. This choice is motivated by the subject reduction theorem, which, in presence of Rule (1), is proved more easily for validity (the property of having an unspecified type) than for type assignment (the property of having a specified type) since types in $\lambda\delta$, as well as in other systems, are not specified uniquely but up to conversion. The same situation arises for Λ_∞ [van Daalen 1994].

At this stage the revised $\lambda\delta$ does not include a type judgment and the exclusion binder χ of Guidi [2009], however our notion of validity should imply Rule (1).

The revised $\lambda\delta$ is defined in Section 2 and its properties are presented in Section 3. Our conclusions are in Section 4. Appendix A gives a summary of the notation we introduce, while Appendix B gives the pointers to the digital version of our results.

We agree that the symbol ▲ terminates our definitions and our proofs in the text.

natural number	i, j, k	starting at 0
term	T, U, V, W	$::= \star k \mid \#i \mid \delta V.T \mid \lambda W.T \mid @V.T \mid ©W.T$
environment	K, L	$::= \star \mid L.\delta V \mid L.\lambda W$

Fig. 1. Terms and environments.

$$@\overline{\circ}.T = T \qquad @(V \,\bar{;}\, \overline{V}).T = @V.(@\overline{V}.T)$$

Fig. 2. Multiple application.

2. DEFINITION OF $\lambda\delta$

In this section we define the revised $\lambda\delta$ from scratch presenting its language (Section 2.1), its reduction rules (Section 2.3), and its validity rules (Section 2.6). These rules depend on some ancillary notions: relocation (Section 2.2), static type assignment (Section 2.4), and degree assignment (Section 2.5). Other notions are introduced to state or prove the main theorems of this article: closures (Section 2.7), extended reduction (Section 2.8), atomic arity assignment (Section 2.9), reducibility candidates (Section 2.10), lazy equivalence (Section 2.11), and an extension of "big trees" termed here "very big trees" (Section 2.12).

We shall use some logical constants: $\forall$ (universal quantification), $\exists$ (existential quantification), $\Rightarrow$ (implication), $\&$ (conjunction), and natural numbers with standard operators: $\le$, $<$, $+$, and $-$. We shall need lists for the normalization theorem. Metavariables for lists will be overlined, like $\overline{c}$. The empty list will be $\overline{\circ}$, and the infix semicolon will denote concatenation, like $c \,\bar{;}\, \overline{c}$.

Contrary to Guidi [2009], in this presentation we want to follow the digital specification of the calculus strictly, especially in the treatment of variables, and we make some notational changes with respect to that article. The reader will find a summary of the revised notation in Appendix A.

2.1. Language

The grammar of $\lambda\delta$ features two syntactic categories: terms and environments, and uses natural numbers. Terms are presented in the "item notation" of Kamareddine and Nederpelt [1996b], and include sorts, variable occurrences, abbreviations, typed abstractions, applications, and type annotations. Contrary to Guidi [2009], environments contain just (nonrecursive) definitions, and typed declarations.

Definition 2.1 (*terms and environments*). Terms and environments are defined in Figure 1. $\star k$ is the sort of index k, $\#i$ is the reference to the variable introduced at depth i [de Bruijn 1994a] (so i is a "de Bruijn index"), $\delta V.T$ is the abbreviation "$\text{let } \#0 = V \text{ in } T$", $\lambda W.T$ is the function "$(\#0 : W) \mapsto T$", $@V.T$ is the application "$T(V)$", and $©W.T$ is the type annotation "$(T : W)$". $\star$ is the empty environment, $L.\delta V$ is L with the definition "$\text{let } \#0 = V$", and $L.\lambda W$ is L with the declaration "$(\#0 : W)$". ▲

Convention: the symbol δ/λ means: "either δ, or λ". If the symbol occurs many times in a statement, it means: "either δ in every occurrence, or λ in every occurrence". The same convention holds for similar symbols we will use, like $\star/\#$ and $©/@$.

The application can be extended to take a list $\overline{V}$ of arguments.

Definition 2.2 (*multiple application*). $@\overline{V}.T$ defined in Figure 2, denotes the application of T the arguments in the list $\overline{V}$ starting from the rightmost term in $\overline{V}$. ▲

Environments are lists so some standard operators can be defined on them.

Definition 2.3 (*length*). Figure 3 defines the length $|L|$ of an environment L. ▲

$$|\star| = 0 \qquad |L.\delta/\lambda W| = |L|$$

Fig. 3. Length of an environment.

$$K.\star = K \qquad K.(L.\delta/\lambda W) = (K.L).\delta/\lambda W$$

Fig. 4. Concatenation of two environments.

$$\mathbb{S}(\star/\#i) \qquad \mathbb{S}(©/@V.T)$$

Fig. 5. Simple (or neutral) terms.

Definition 2.4 (*concatenation*). Figure 4 defines the concatenation $K.L$ of L before K. In particular we write $\delta/\lambda W.L$ for $(\star.\delta/\lambda W).L$. ▲

Normalization requires two predicates: see Definition 2.33 and Theorem 3.5(7).

Definition 2.5 (*neutrality*). $\mathbb{S}(T)$ states that the term T is simple (or neutral) as defined in Figure 5. Specifically, T is neither an abbreviation, nor an abstraction. ▲

Definition 2.6 (*top structure*). $T_1 \approx T_2$ states that the terms T_1 and T_2 have the same top structure as defined in Figure 6. Specifically, T_1 and T_2 are the same atomic term or start with the same constructor. ▲

2.2. Relocation

Managing variables referred by depth requires a well-known function $\uparrow^{\langle l,m\rangle} T$ connected to the function $\tau_m(T)$ of de Bruijn [1994a]. In particular, when the term T enters the scope of m binders, $\uparrow^{\langle 0,m\rangle} T$ relocates the indexes of its free variables. The composition of such functions is of interest as well.

Definition 2.7 (*relocation*). The relation $\uparrow^{\langle l,m\rangle} T_1 = T_2$ defined in Figure 7, states that T_2 is the relocation of T_1 at level l with depth (or "height") m.
We term the pair $\langle l,m\rangle$ a "relocation pair". ▲

Definition 2.8 (*vector relocation*). The relation $\uparrow^{\langle l,m\rangle} \overline{T}_1 = \overline{T}_2$ defined in Figure 8, applies $\langle l,m\rangle$ to the components of the list $\overline{T}_1$ preserving their order in the list $\overline{T}_2$. ▲

Definition 2.9 (*multiple relocation*). The relation $\uparrow^{\overline{c}} T_1 = T_2$ defined in Figure 9, applies the list $\overline{c}$ of relocation pairs to T_1 starting from the leftmost pair in $\overline{c}$. ▲

If $\uparrow^{\langle l,m\rangle} T_1 = T_2$, notably, T_2 does not refer to the variables introduced at depth i with $l \le i < l + m$. So a relation $\downarrow_{\langle l,m\rangle} L_1 = L_2$ is provided for removing the i-th entries of L_1 such that $l \le i < l + m$, while relocating the i-th entries such that $i < l$. The relation is defined only when this relocation is possible, that is when an i-th entry with $i < l$ does not refer to an i-th entry with $l \le i < l + m$. The $0 - th$ entry of L_1 is the head of L_1. We term this relation "drop" as opposed to relocation, which is sometimes termed "lift".
Notice that if $\downarrow_{\langle 0,i\rangle} L_1 = L_2$, then the head of L_2 contains the i-th entry of L_1.

Definition 2.10 (*drop*). The relation $\downarrow_{\langle l,m\rangle} L_1 = L_2$ defined in Figure 10, states that L_2 is L_1 without the i-th entries such that $l \le i < l + m$, and with the i-th entries such that $i < l$ relocated accordingly. ▲

Figure 10(atom) generalizes "drop" of Guidi [2006] allowing $\downarrow_{\langle l,0\rangle} L = L$ when $|L| \le l$.

$$\star/\#i \approx \star/\#i \qquad \delta/\lambda/©/@V_1.T_1 \approx \delta/\lambda/©/@V_2.T_2$$

Fig. 6. Terms with the same top structure.

$$\begin{array}{lcl} \text{natural number} & l, m & \text{starting at } 0 \\ \text{relocation pair} & c & ::= \langle l, m \rangle \end{array}$$

$$\frac{}{\uparrow^{\langle l,m\rangle} \star k = \star k}\text{sort} \qquad \frac{i < l}{\uparrow^{\langle l,m\rangle} \#i = \#i}\text{lref\_lt} \qquad \frac{l \le i}{\uparrow^{\langle l,m\rangle} \#i = \#(i+m)}\text{lref\_ge}$$

$$\frac{\uparrow^{\langle l,m\rangle} W_1 = W_2 \quad \uparrow^{\langle l+1,m\rangle} T_1 = T_2}{\uparrow^{\langle l,m\rangle} \delta/\lambda W_1.T_1 = \delta/\lambda W_2.T_2}\text{bind} \qquad \frac{\uparrow^{\langle l,m\rangle} V_1 = V_2 \quad \uparrow^{\langle l,m\rangle} T_1 = T_2}{\uparrow^{\langle l,m\rangle} ©/@V_1.T_1 = ©/@V_2.T_2}\text{flat}$$

Fig. 7. Relocation.

$$\frac{}{\uparrow^{\langle l,m\rangle} \overline{\circ} = \overline{\circ}}\text{empty} \qquad \frac{\uparrow^{\langle l,m\rangle} T_1 = T_2 \quad \uparrow^{\langle l,m\rangle} \overline{T}_1 = \overline{T}_2}{\uparrow^{\langle l,m\rangle} (T_1 \,\bar{;}\, \overline{T}_1) = T_2 \,\bar{;}\, \overline{T}_2}\text{cons}$$

Fig. 8. Vector relocation.

Definition 2.11 (*multiple drop*). The relation $\downarrow_{\overline{c}} L_1 = L_2$ defined in Figure 11, applies the list $\overline{c}$ of relocation pairs to L_1 starting from the leftmost pair in $\overline{c}$. ▲

The next equivalence relation appears in Theorem 3.9(3).

Definition 2.12 (*ranged equivalence*). The relation $L_1 \;\tilde{\approx}_{\langle l,m\rangle}\; L_2$ defined in Figure 12, states that L_1 and L_2 have the same length and the same i-th entries for $l \le i < l + m$. ▲

2.3. Reduction

$\lambda\delta$ features a transition system with five schemes of reducible expressions (redexes). Care is taken to design a deterministic and confluent system with disjoint redex schemes, in which the call-by-value β-reduction is broken into its basic components.

Definition 2.13 (*transitions*). Figure 13 defines the redexes and their transitions β, δ, ϵ, ζ, and θ, which depend on an environment L. The β-reduction is delayed (call-by-name style), the δ-expansion expands a definition in L, the ϵ-contraction removes a type annotation, the ζ-contraction removes an unreferenced abbreviation, and the θ-reduction [Curien and Herbelin 2000] swaps an application and an abbreviation. ▲

Notice that the β-redex contains a type annotation W that, contrary to Guidi [2009], remains in the β-reductum. This choice is connected with the revised form of the normalization theorem. Also notice that δ-expansion, contrary to Guidi [2009], does not mention substitution. In the light of next Definition 2.14, delayed parallel substitution is seen as a special case of reduction.

Following Guidi [2009], we present parallel reduction to ease the proof of the confluence theorem, but here we take environment-dependent reduction as primitive.

Definition 2.14 (*parallel reduction for terms*). The relation $L \vdash T_1 \rightrightarrows T_2$ defined in Figure 14, indicates one step of parallel reduction from T_1 to T_2 in L. ▲

We compute a call-by-value β-reduction in two steps, as we illustrate by computing the term $\Delta(\Delta)$. In particular we set $\Delta_T = \lambda T.@\#0.\#0$ and we agree that $\uparrow^{\langle 0,1\rangle} T = U$.

$$\frac{}{\uparrow^{\bar{\circ}} T = T}\,\text{empty} \qquad \frac{\uparrow^{c} T_1 = T \quad \uparrow^{\bar{c}} T = T_2}{\uparrow^{c;\bar{c}} T_1 = T_2}\,\text{cons}$$

Fig. 9. Multiple relocation.

$$\frac{}{\downarrow_{\langle l,0\rangle} \star = \star}\,\text{atom} \qquad \frac{\downarrow_{\langle 0,0\rangle} L_1 = L_2}{\downarrow_{\langle 0,0\rangle} L_1.\delta/\lambda W = L_2.\delta/\lambda W}\,\text{pair}$$

$$\frac{\downarrow_{\langle 0,m\rangle} L_1 = L_2}{\downarrow_{\langle 0,m+1\rangle} L_1.\delta/\lambda W = L_2}\,\text{drop} \qquad \frac{\downarrow_{\langle l,m\rangle} L_1 = L_2 \quad \uparrow^{\langle l,m\rangle} W_2 = W_1}{\downarrow_{\langle l+1,m\rangle} L_1.\delta/\lambda W_1 = L_2.\delta/\lambda W_2}\,\text{skip}$$

Fig. 10. Drop.

$$\frac{}{\downarrow_{\bar{\circ}} L = L}\,\text{empty} \qquad \frac{\downarrow_{c} L_1 = L \quad \downarrow_{\bar{c}} L = L_2}{\downarrow_{c;\bar{c}} L_1 = L_2}\,\text{cons}$$

Fig. 11. Multiple Drop.

$$\begin{array}{rl} \beta & L \vdash @\Delta_T.\Delta_T \rightrightarrows \delta(©T.\Delta_T).@\#0.\#0 \\ \epsilon, \delta, \zeta & L \vdash \delta(©T.\Delta_T).@\#0.\#0 \rightrightarrows @\Delta_T.\Delta_T \\ & \text{by } L.\delta(©T.\Delta_T) \vdash \#0 \rightrightarrows \Delta_U \text{ and } L \vdash ©T.\Delta_T \rightrightarrows \Delta_T \end{array}$$

The advantage of environment-dependent parallel reduction over the approach of Guidi [2009] lies in the increased parallelism of δ-expansions, which we need for the "big tree" theorem. Suppose that $[m{\leftarrow}V\,]\,T$ replaces with V some references in T to the variable introduced at depth m, and compare Figure 14(bind) and Figure 14(δ) with Rule (3) (*i.e.*, their environment-free counterpart). When we replace many variable instances in one step with this rule, each instance receives the same reduct V_2 of V_1. Whereas, by Figure 14(δ) each instance may receive a different reduct of V_1.

$$\frac{V_1 \rightrightarrows V_2 \quad T_1 \rightrightarrows T \quad [0{\leftarrow}V_2]\,T = T_2}{\delta V_1.T_1 \rightrightarrows \delta V_2.T_2}\,\delta\text{-free} \tag{3}$$

Notice that the subsystem of rules: Figure 14(bind), Figure 14(flat), Figure 14(atom), and Figure 14(δ) axiomatizes environment-dependent parallel substitution.

We derive several notions from parallel reduction: an extension for environments needed in the confluence theorem, and some transitive closures. In this setting we agree that a "computation" is a reduction sequence consisting of zero or more steps.

Definition 2.15 (*parallel reduction for environments*). The relation $L_1 \vdash\rightrightarrows L_2$ defined in Figure 15 indicates one step of parallel reduction from L_1 to L_2. ▲

Definition 2.16 (*parallel computation and conversion*). The relation $L \vdash T_1 \rightrightarrows^* T_2$ (computation) is the transitive closure of $L \vdash T_1 \rightrightarrows T_2$, while $L \vdash T_1 \leftrightarrows^* T_2$ (conversion) is the symmetric and transitive closure of $L \vdash T_1 \rightrightarrows T_2$. Moreover $L_1 \vdash\rightrightarrows^* L_2$ (computation) is the transitive closure of $L_1 \vdash\rightrightarrows L_2$. Figure 16 defines $L \vdash T_1 \rightrightarrows^* T_2$ for reference. The other notions are defined in the same manner. ▲

The transitive closures we just defined are reflexive, because so is $L \vdash T_1 \rightrightarrows T_2$. Therefore the symbol * in their notation is justified as a Kleene star meaning "zero or more".

A characteristic feature of $\lambda\delta$ is the use of reflexive relations for environments termed here "refinements", invoked when proving that reduction preserves some property. Specifically, they are invoked in the case of Figure 14(β) given that a backward application of Figure 14(bind) moves part of the β-redex and part of the β-reductum

$$\frac{}{\star \approxeq_{\langle l,m\rangle} \star}\text{atom} \qquad \frac{L_1 \approxeq_{\langle 0,m\rangle} L_2}{L_1.\delta/\lambda W \approxeq_{\langle 0,m+1\rangle} L_2.\delta/\lambda W}\text{pair}$$

$$\frac{L_1 \approxeq_{\langle 0,0\rangle} L_2}{L_1.\delta_1/\lambda_1 W_1 \approxeq_{\langle 0,0\rangle} L_2.\delta_2/\lambda_2 W_2}\text{zero} \qquad \frac{L_1 \approxeq_{\langle l,m\rangle} L_2}{L_1.\delta_1/\lambda_1 W_1 \approxeq_{\langle l+1,m\rangle} L_2.\delta_2/\lambda_2 W_2}\text{succ}$$

Fig. 12. Ranged equivalence.

$$\frac{}{L \vdash @V.\lambda W.T \to \delta(©W.V).T}\beta \qquad \frac{\downarrow_{\langle 0,i\rangle} L = K.\delta V_1 \quad \uparrow^{\langle 0,i+1\rangle} V_1 = V_2}{L \vdash \#i \to V_2}\delta$$

$$\frac{}{L \vdash ©U.T \to T}\epsilon \qquad \frac{\uparrow^{\langle 0,1\rangle} T_2 = T_1}{L \vdash \delta V.T_1 \to T_2}\zeta \qquad \frac{\uparrow^{\langle 0,1\rangle} V_1 = V_2}{L \vdash @V_1.\delta W.T \to \delta W.@V_2.T}\theta$$

Fig. 13. Transitions.

$$\frac{L \vdash W_1 \rightrightarrows W_2 \quad L.\delta/\lambda W_1 \vdash T_1 \rightrightarrows T_2}{L \vdash \delta/\lambda W_1.T_1 \rightrightarrows \delta/\lambda W_2.T_2}\text{bind} \qquad \frac{L \vdash V_1 \rightrightarrows V_2 \quad L \vdash T_1 \rightrightarrows T_2}{L \vdash ©/@V_1.T_1 \rightrightarrows ©/@V_2.T_2}\text{flat}$$

$$\frac{}{L \vdash \star/\#i \rightrightarrows \star/\#i}\text{atom} \qquad \frac{\downarrow_{\langle 0,i\rangle} L = K.\delta V_1 \quad K \vdash V_1 \rightrightarrows V_2 \quad \uparrow^{\langle 0,i+1\rangle} V_2 = W_2}{L \vdash \#i \rightrightarrows W_2}\delta$$

$$\frac{L \vdash V_1 \rightrightarrows V_2 \quad L \vdash W_1 \rightrightarrows W_2 \quad L.\lambda W_1 \vdash T_1 \rightrightarrows T_2}{L \vdash @V_1.\lambda W_1.T_1 \rightrightarrows \delta(©W_2.V_2).T_2}\beta \qquad \frac{L.\delta V \vdash U_1 \rightrightarrows U_2 \quad \uparrow^{\langle 0,1\rangle} T_2 = U_2}{L \vdash \delta V.U_1 \rightrightarrows T_2}\zeta$$

$$\frac{L \vdash T_1 \rightrightarrows T_2}{L \vdash ©U.T_1 \rightrightarrows T_2}\epsilon \qquad \frac{L \vdash V_1 \rightrightarrows V_2 \quad \uparrow^{\langle 0,1\rangle} V_2 = W_2 \quad L \vdash U_1 \rightrightarrows U_2 \quad L.\delta U_1 \vdash T_1 \rightrightarrows T_2}{L \vdash @V_1.\delta U_1.T_1 \rightrightarrows \delta U_2.@W_2.T_2}\theta$$

Fig. 14. Parallel reduction for terms (single step).

in the environment. The basic refinement is given next and occurs in the proof of the confluence theorem. The other refinements imply this one. See Definition 2.20, Definition 2.23, Definition 2.31, Definition 2.36.

Definition 2.17 (*refinement for preservation of reduction*). Figure 17 defines the relation $L_1 \dot{\subseteq} L_2$ stating that L_1 refines L_2 for preservation of reduction. ▲

The main results on reduction, conversion, and refinement are in Section 3.1.

2.4. Iterated Static Type Assignment

The "static" type assignment defined in this section is our counterpart of the so-called "de Bruijn" type assignment of the Automath tradition [van Daalen 1994]. As such, it plays a central role in our definition of validity. Its name recalls that we can compute it without the help of $\beta\zeta\theta$-reductions.

Intuitively, the term T has a static type U in the environment L iff the head variable occurrence of T is hereditarily closed in L. In that case, U is just a candidate type for T. However, when T is valid, its static type serves as the "canonical" type [Kamareddine and Nederpelt 1996a], or as the "inferred" type [Coscoy 1996].

The "static type iterated n times" is related to the notion of validity implied by Rule (1) and It will be convenient to define it as a primitive notion (denoted by $L \vdash T \bullet^{*(n)} U$), that will not be the reflexive and transitive closure of the "static type iterated one time". In fact we are not interested in full reflexivity (*i.e.*, $L \vdash T \bullet^{*(0)} T$ for each T). On the contrary, we wish to ensure that $L \vdash T \bullet^{*(n)} U$ holds iff the head

$$\frac{}{\star \vdash\Rrightarrow \star}\text{atom} \qquad \frac{L_1 \vdash\Rrightarrow L_2 \quad L_1 \vdash W_1 \Rrightarrow W_2}{L_1.\delta/\lambda W_1 \vdash\Rrightarrow L_2.\delta/\lambda W_2}\text{pair}$$

Fig. 15. Parallel reduction for environments (single step).

$$\frac{L \vdash T_1 \Rrightarrow T_2}{L \vdash T_1 \Rrightarrow^* T_2}\text{inj} \qquad \frac{L \vdash T_1 \Rrightarrow^* T \quad L \vdash T \Rrightarrow T_2}{L \vdash T_1 \Rrightarrow^* T_2}\text{step}$$

Fig. 16. Parallel computation for terms (multi-step).

$$\frac{}{L \,\dot{\subseteq}\, \star}\text{atom} \qquad \frac{L_1 \,\dot{\subseteq}\, L_2}{L_1.\delta/\lambda W \,\dot{\subseteq}\, L_2.\delta/\lambda W}\text{pair} \qquad \frac{L_1 \,\dot{\subseteq}\, L_2}{L_1.\delta(©W.V) \,\dot{\subseteq}\, L_2.\lambda W}\text{beta}$$

Fig. 17. Refinement for preservation of reduction.

variable occurrence of T is hereditarily closed in L regardless of n, hence even for $n = 0$. As a matter of fact, differentiating the case $n = 0$ for the sake of reflexivity, yields a less elegant definition of $L \vdash T \bullet^{*(n)} U$.

According to our type policy, the sort of index k is typed by the sort of index $h(k)$ where h is function chosen at will as long as a monotonicity condition is satisfied.

Definition 2.18 (*iterated static type assignment*). A "sort hierarchy parameter" is any function h satisfying the strict monotonicity condition: $k < h(k)$. Moreover h^n will denote h composed n times. For a natural number n, the relation $L \vdash T \bullet_h^{*(n)} U$ defined in Figure 18, indicates that U is the n-iterated "static" type of T in L according to h. ▲

This definition allows to say that U is the static type of T in L when $L \vdash T \bullet_h^{*(1)} U$, which differs in Figure 18(cast) from the notion $L \vdash T \bullet_h U$ defined by [Guidi 2009] with the name st. For example we have $L.\lambda(©\star k_1.\star k_2) \vdash \#0 \bullet_h (©\star k_1.\star k_2)$ but $L.\lambda(©\star k_1.\star k_2) \vdash \#0 \bullet_h^{*(1)} \star k_2$. Although $L \vdash T \bullet_h^{*(0)} T$ does not hold in general, we can prove that $L \vdash T_1 \bullet_h^{*(0)} T_2$ implies $L \vdash T_1 \Rrightarrow T_2$ by δ-expansion and ϵ-contraction. We remark that the rules of Figure 18 are syntax-oriented, so the n-iterated static type of T in L, if it exists, is unique for any given h and n. See Theorem 3.4(1).

2.5. Degree Assignment

The "degree" of a term T is a number d indicating the position of T in a type hierarchy. A well-established definition assigns degree 1 to the first sort (for instance τ in Λ_∞ or $\star$ in the λ-Cube [Barendregt 1993]) and degree $d + 1$ to T such that $\Gamma \vdash T : U$ when U has degree d. In $\lambda\delta$, as in ECC [Luo 1990], there is no top sort and the degree is an integer. So this definition prevents from reasoning by induction on the degree.

According to our policy, the degree of a sort is a natural number given by a function g that can be chosen at will as long as a compatibility condition is satisfied.

Once sorts are assigned a degree, the assignment extends to terms accordingly.

Definition 2.19 (*degree assignment*). Given a sort hierarchy parameter h, a "sort degree parameter" is any function g_h satisfying the compatibility condition: if $g_h(k) = d$ then $g_h(h(k)) = d - 1$. The relation $L \vdash T \blacksquare_{h,g} d$ defined in Figure 19, indicates that T has degree d in L according to h and g_h. ▲

As we see, the term T has a degree in L iff the head variable occurrence of T is hereditarily closed in L. So having a degree, is equivalent to having a static type.

The refinement given next occurs in the proof of the preservation theorem and is needed to prove that the reduction of valid terms preserves their degree.

$$\text{natural number} \quad n \quad \text{starting at } 0$$

$$\frac{}{L \vdash \star k \bullet_h^{*(n)} \star(h^n(k))}\text{sort} \qquad \frac{\downarrow_{\langle 0,i\rangle} L = K.\lambda W \quad K \vdash W \bullet_h^{*(0)} V}{L \vdash \#i \bullet_h^{*(0)} \#i}\text{zero}$$

$$\frac{L.\delta/\lambda W \vdash T \bullet_h^{*(n)} U}{L \vdash \delta/\lambda W.T \bullet_h^{*(n)} \delta/\lambda W.U}\text{bind}$$

$$\frac{\downarrow_{\langle 0,i\rangle} L = K.\lambda W_1 \quad K \vdash W_1 \bullet_h^{*(n)} V_1 \quad \uparrow^{\langle 0,i+1\rangle} V_1 = V_2}{L \vdash \#i \bullet_h^{*(n+1)} V_2}\text{succ} \qquad \frac{L \vdash T \bullet_h^{*(n)} U}{L \vdash @V.T \bullet_h^{*(n)} @V.U}\text{appl}$$

$$\frac{\downarrow_{\langle 0,i\rangle} L = K.\delta V \quad K \vdash V \bullet_h^{*(n)} W_1 \quad \uparrow^{\langle 0,i+1\rangle} W_1 = W_2}{L \vdash \#i \bullet_h^{*(n)} W_2}\text{ldef} \qquad \frac{L \vdash T \bullet_h^{*(n)} U}{L \vdash \copyright W.T \bullet_h^{*(n)} U}\text{cast}$$

Fig. 18. Iterated tatic type assignment.

$$\text{natural number} \quad d \quad \text{starting at } 0$$

$$\frac{g_h(k) = d}{L \vdash \star k \blacksquare_{h,g}\, d}\text{sort} \qquad \frac{\downarrow_{\langle 0,i\rangle} L = K.\delta V \quad K \vdash V \blacksquare_{h,g}\, d}{L \vdash \#i \blacksquare_{h,g}\, d}\text{ldef} \qquad \frac{\downarrow_{\langle 0,i\rangle} L = K.\lambda W \quad K \vdash W \blacksquare_{h,g}\, d}{L \vdash \#i \blacksquare_{h,g}\, d+1}\text{ldec}$$

$$\frac{L.\delta/\lambda W \vdash T \blacksquare_{h,g}\, d}{L \vdash \delta/\lambda W.T \blacksquare_{h,g}\, d}\text{bind} \qquad \frac{L \vdash T \blacksquare_{h,g}\, d}{L \vdash \copyright/@V.T \blacksquare_{h,g}\, d}\text{flat}$$

Fig. 19. Degree assignment.

$$\frac{}{\star \mathrel{\dot\subseteq}_{\blacksquare h,g} \star}\text{atom} \qquad \frac{L_1 \mathrel{\dot\subseteq}_{\blacksquare h,g} L_2}{L_1.\delta/\lambda W \mathrel{\dot\subseteq}_{\blacksquare h,g} L_2.\delta/\lambda W}\text{pair}$$

$$\frac{L_1 \mathrel{\dot\subseteq}_{\blacksquare h,g} L_2 \quad L_1 \vdash V \blacksquare_{h,g}\, d+1 \quad L_2 \vdash W \blacksquare_{h,g}\, d}{L_1.\delta(\copyright W.V) \mathrel{\dot\subseteq}_{\blacksquare h,g} L_2.\lambda W}\text{beta}$$

Fig. 20. Refinement for preservation of degree.

Definition 2.20 (*refinement for preservation of degree*). Figure 20 defines the relation $L_1 \mathrel{\dot\subseteq}_{\blacksquare h,g} L_2$ stating that L_1 refines L_2 for preservation of degree. ▲

The main results on degree assignment and on its refinement are in Section 3.2.

2.6. Stratified Validity

Our validity rules for a term X in an environment L, are designed to ensure that:

(1) every variable occurrence in X is closed in X or in L; the expected type of every declared variable occurrence in X is valid in its environment; the expansion of every defined variable occurrence in X is valid in its environment;
(2) every subterm of X is valid in its environment;
(3) for every type annotation $\copyright W.V$ in X, the inferred type of V converts to W in L;
(4) for every application $@V.T$ in X, the inferred type of T iterated enough times converts to the form $\lambda W.U$, and the inferred type of V converts to W in L.

Clause (4) is our extension of the "applicability condition", which in a PTS is:

— for every application $@V.T$ in X, the inferred type of T iterated one time converts to the form $\Pi W.U$, and the inferred type of V converts to W in L.

$$\frac{n \le d \quad L \vdash T_1 \blacksquare_{h,g}\, d \quad L \vdash T_1 \bullet_h^{*(n)} T \quad L \vdash T \Rrightarrow^* T_2}{L \vdash T_1 \bullet^*\!\!\Rrightarrow_{h,g}^{*(n)} T_2}\,\text{scpds}$$

$$\frac{L \vdash T_1 \bullet^*\!\!\Rrightarrow_{h,g}^{*(n_1)} T \quad L \vdash T_2 \bullet^*\!\!\Rrightarrow_{h,g}^{*(n_2)} T}{L \vdash T_1 \bullet^*\!\!\Leftrightarrow_{h,g}^{*(n_1,n_2)} T_2}\,\text{scpes}$$

Fig. 21. Stratified decomposed computation and conversion.

$$\frac{}{L \vdash \star k\ !_{h,g}}\,\text{sort} \qquad \frac{\downarrow_{\langle 0,i\rangle} L = K.\delta/\lambda W \quad K \vdash W\ !_{h,g}}{L \vdash \#i\ !_{h,g}}\,\text{lref} \qquad \frac{L \vdash W\ !_{h,g} \quad L.\delta/\lambda W \vdash T\ !_{h,g}}{L \vdash \delta/\lambda W.T\ !_{h,g}}\,\text{bind}$$

$$\frac{L \vdash U\ !_{h,g} \quad L \vdash T\ !_{h,g} \quad L \vdash U \bullet^*\!\!\Rrightarrow_{h,g}^{*(0)} U_0 \quad L \vdash T \bullet^*\!\!\Rrightarrow_{h,g}^{*(1)} U_0}{L \vdash \copyright U.T\ !_{h,g}}\,\text{cast}$$

$$\frac{L \vdash V\ !_{h,g} \quad L \vdash T\ !_{h,g} \quad L \vdash V \bullet^*\!\!\Rrightarrow_{h,g}^{*(1)} W_0 \quad L \vdash T \bullet^*\!\!\Rrightarrow_{h,g}^{*(n)} \lambda W_0.U_0}{L \vdash @V.T\ !_{h,g}}\,\text{appl}$$

Fig. 22. Stratified validity.

In [Guidi 2009] we took by mistake the latter condition replacing Π with λ, rather than Clause (4). The idea of Clause (3) and Clause (4) is that a valid term is typable and its types are the valid terms that convert to its inferred type. Notice that this property holds for the calculus of Guidi [2009]. As for Λ_∞ [van Daalen 1994], the preservation theorem for $\lambda\delta$ (stating that validity is preserved by reduction) requires an induction on the degree motivated by its extended applicability condition.

So we define a "stratified" validity depending on a degree assignment in that we require a positive degree for V in Clause (3) and Clause (4), and in that the inferred type of T is not iterated more times than the degree of T in Clause (4). Intuitively, this is validity up to a degree. The next ancillary relations are needed in the formal statement of Clause (3) and Clause (4).

Definition 2.21 (*decomposed computation and conversion*). Figure 21 defines the relation $L \vdash T_1 \bullet^*\!\!\Rrightarrow_{h,g}^{*(n)} T_2$, concatenating a degree-guarded iterated static type assignment and a computation, and the corresponding conversion $L \vdash T_1 \bullet^*\!\!\Leftrightarrow_{h,g}^{*(n_1,n_2)} T_2$. ▲

Definition 2.22 (*stratified validity*). The relation $L \vdash T\ !_{h,g}$ defined in Figure 22 states that the term T is valid in L with respect to the parameters h and g_h. ▲

The refinement given next is needed to prove the preservation Theorem 3.15.

Definition 2.23 (*refinement for preservation of validity*). Figure 23 defines the relation $L_1 \dot\subseteq\!!_{h,g} L_2$ stating that L_1 refines L_2 for preservation of stratified validity. ▲

The main results on stratified validity and on its refinement are in Section 3.8.

2.7. Closures

Most properties of $\lambda\delta$ are proved by structural induction, but this proof method fails for some important results like the confluence theorem. In most cases a proof by induction on the "proper subclosures" provides for a good alternative. The main exception is the preservation theorem. Hereafter, a "closure" is an ordered pair $\langle L, T\rangle$ where T is a term

$$\frac{L \vdash U\ !\ _{h,g} \quad L \vdash T\ !\ _{h,g} \quad \forall n.\ n \le d \Rightarrow L \vdash U \bullet^{*} \Leftrightarrow^{*(n,n+1)}_{h,g} T}{L \vdash ©U.T\ !\ _{h,g}{}^{(d)}}\text{hcast}$$

$$\frac{}{\star\ \dot{\sqsubseteq}!_{h,g}\ \star}\text{atom} \qquad \frac{L_1\ \dot{\sqsubseteq}!_{h,g}\ L_2}{L_1.\delta/\lambda W\ \dot{\sqsubseteq}!_{h,g}\ L_2.\delta/\lambda W}\text{pair}$$

$$\frac{L_1\ \dot{\sqsubseteq}!_{h,g}\ L_2 \quad L_1 \vdash ©W.V\ !\ _{h,g}{}^{(d)} \quad L_2 \vdash W\ !\ _{h,g} \quad L_1 \vdash V\ \blacksquare_{h,g}\ d+1 \quad L_2 \vdash W\ \blacksquare_{h,g}\ d}{L_1.\delta(©W.V)\ \dot{\sqsubseteq}!_{h,g}\ L_2.\lambda W}\text{beta}$$

Fig. 23. Refinement for preservation of stratified validity.

$$\frac{}{\langle L, \delta/\lambda/©/@V.T\rangle \sqsupset \langle L, V\rangle}\text{pair} \qquad \frac{}{\langle L, ©/@V.T\rangle \sqsupset \langle L, T\rangle}\text{flat} \qquad \frac{}{\langle L, \delta/\lambda W.T\rangle \sqsupset \langle L.\delta/\lambda W, T\rangle}\text{bind}$$

$$\frac{}{\langle K.\delta/\lambda W, \#0\rangle \sqsupset \langle K, W\rangle}\text{lref} \qquad \frac{\downarrow_{\langle 0,m+1\rangle} L = K \quad \uparrow^{\langle 0,m+1\rangle} T = U}{\langle L, U\rangle \sqsupset \langle K, T\rangle}\text{drop}$$

Fig. 24. Direct subclosure.

$$\frac{\langle L_1, T_1\rangle \sqsupset^? \langle L_2, T_2\rangle}{\langle L_1, T_1\rangle \sqsupset^* \langle L_2, T_2\rangle}\text{inj} \qquad \frac{\langle L_1, T_1\rangle \sqsupset^* \langle L, T\rangle \quad \langle L, T\rangle \sqsupset^? \langle L_2, T_2\rangle}{\langle L_1, T_1\rangle \sqsupset^* \langle L_2, T_2\rangle}\text{step}$$

Fig. 25. Subclosure.

closed in an environment L. Intuitively, a subclosure of $\langle L, T\rangle$ contains a subterm of T and a subenvironment of L.

The "direct" and "transitive" subclosures of $\langle L, T\rangle$ are defined next.

Definition 2.24 (*direct subclosure*). The relation $\langle L_1, T_1\rangle \sqsupset \langle L_2, T_2\rangle$ defined in Figure 24, states that $\langle L_2, T_2\rangle$ is a "direct subclosure" of $\langle L_1, T_1\rangle$.
The relation $\langle L_1, T_1\rangle \sqsupset^? \langle L_2, T_2\rangle$ is its reflexive closure. ▲

The symbol $^?$ in $\langle L_1, T_1\rangle \sqsupset^? \langle L_2, T_2\rangle$ means "one or none" as for regular expressions.

Definition 2.25 (*subclosure and proper subclosure*). Figure 25 defines the relation $\langle L_1, T_1\rangle \sqsupset^* \langle L_2, T_2\rangle$ (subclosure) as the (reflexive and) transitive closure of $\langle L_1, T_1\rangle \sqsupset^? \langle L_2, T_2\rangle$. While the proper subclosure is the transitive closure of $\langle L_1, T_1\rangle \sqsupset \langle L_2, T_2\rangle$. ▲

We want to remark that generalizing the constant 0 in Figure 24(drop), invalidates the commutation property between the direct subclosure and the parallel reduction, which is crucial for the preservation theorem. Moreover the proper subclosure is well founded, as we see by observing that each step of direct subclosure decreases the sum of the term constructors in the closure.

2.8. Extended Reduction

Having introduced subclosures, we can take a glance at the strong normalization of "rst-reduction" [de Vrijer 1994], informally known as the "big tree" theorem.

Ideally, given a closure $\langle L_1, T_1\rangle$ we define a step $\to_r$ along the axis of reducts, a step $\to_s$ along the axis of subclosures, and a step $\to_t$ along the axis of iterated static types:

$$\frac{L_1 \vdash T_1 \Rrightarrow T_2 \quad T_1 \neq T_2}{\langle L_1, T_1\rangle \to_r \langle L_1, T_2\rangle} \qquad \frac{\langle L_1, T_1\rangle \sqsupset \langle L_2, T_2\rangle}{\langle L_1, T_1\rangle \to_s \langle L_2, T_2\rangle} \qquad \frac{L_1 \vdash T_1 \bullet^{*(1)}_{h} T_2 \quad L_1 \vdash T_1\ \blacksquare_{h,g}\ d+1}{\langle L_1, T_1\rangle \to_t \langle L_1, T_2\rangle} \tag{4}$$

and we are interested in proving that any sequence of such steps staring from $\langle L, T\rangle$, is finite when $L \vdash T\ !\ _{h,g}$. This is the strong normalization of a relation $\to_{rst}$ comprising

$$\frac{g_h(k) = d + 1}{L \vdash \star k \to_{h,g} \star(h(k))}s \qquad \frac{\downarrow_{\langle 0,i\rangle} L = K.\lambda W_1 \quad \uparrow^{\langle 0,i+1\rangle} W_1 = W_2}{L \vdash \#i \to_{h,g} W_2}l \qquad \frac{}{L \vdash ©U.T \to_{h,g} U}e$$

Fig. 26. Extended transitions.

the steps in (4). We remark that the interest in this result lies on the very powerful induction principle it provides for proving properties of valid terms. We shall need this power for the preservation theorem. Notice the side condition $T_1 \neq T_2$ ensuring that $\to_r$ is not reflexive (we can prove that Definition 2.14 forbids single-step reduction cycles), and the side condition $L_1 \vdash T_1 \,\blacksquare_{h,g}\, d+1$ ensuring that $\to_t$ cannot be applied indefinitely (otherwise, $\langle L, \star k\rangle \to_t \langle L, \star(h(k))\rangle$ is always possible).

As to the proof of the "big tree" theorem, we take a sequence of steps starting from a valid closure and we would like to commute adjacent steps until the steps of the same kind are clustered. At that point, an infinite sequence would lead to an infinite cluster, contradicting either strong normalization of reduction (steps of kind $\to_r$), or well-foundedness of subclosures (steps of kind $\to_s$), or else finiteness of degree in the given system of reference g_h (steps of kind $\to_t$).

Unfortunately, it is a matter of fact that a step $\to_r$ and a step $\to_t$ may not commute. Consider the β-redex $T_1 = @V.\lambda W_1.\#0$ and its β-reductum $T_2 = \delta(©W_1.V).\#0$. Then the static type of T_1 is $U_1 = @V.\lambda W_1.(\uparrow^{\langle 0,1\rangle} W_1)$, and its β-reductum is $U_0 = \delta(©W_1.V).(\uparrow^{\langle 0,1\rangle} W_1)$. Moreover, let W_2 be the static type of V, then the static type of T_2 is $U_2 = \delta(©W_1.V).(\uparrow^{\langle 0,1\rangle} W_2)$. Now compare U_0 and U_2, that is: W_1 and W_2 (respectively, the "expected" and the "inferred" type of V). Even assuming that T_1 is valid, these terms are the same one just up to conversion. It is an even simpler matter of fact that a step $\to_s$ and a step $\to_t$ may not commute. Consider the term T_1 and its static type U_1, take V as a subterm of T_1 and its static type W_2. Yet W_2 is not a subterm of U_1 and may just be related to W_1 by conversion when T_1 is valid.

Anyway, a step $\to_r$ and a step $\to_s$ commute with the help of reduction for environments. In fact, we can prove the "pentagon" (*i.e.*, a proposition on five closures connected by five relations) of Rule (5), in which the reduction for environments emerges in the case $L_1 = K.\lambda V_1$ and $T_1 = \#0$.

$$\frac{\langle L_1, T_1\rangle \sqsupset \langle K, V_1\rangle \quad K \vdash V_1 \rightrightarrows V_2}{\exists L_2, T_2.\ L_1 \vdash\rightrightarrows L_2 \ \&\ L_2 \vdash T_1 \rightrightarrows T_2 \ \&\ \langle L_2, T_2\rangle \sqsupset \langle K, V_2\rangle} \tag{5}$$

These considerations lead us to define the "extended reduction" such that:

(1) it extends ordinary reduction (*i.e.*, a $\to_r$ step) by supporting a $\to_t$ step;
(2) it preserves strong normalization "smoothly" in that little effort is expected in updating the proof that works for ordinary reduction [Guidi 2009];
(3) it preserves the commutation with subclosures in the form of Rule (5).

Extended reduction is our counterpart of "rt-reduction" [de Vrijer 1994]. It comprises the transitions of Definition 2.13 and the ones listed next.

Definition 2.26 (*extended transitions*). Figure 26 defines the extended redexes and their associated transitions t, l, and e, which depend on a sort degree parameter g_h and on an environment L. The transitions t, l and e respectively replace a sort, a declared variable, and a type annotation with their expected type. ▲

The transitions t and l provide the support for the t-step of (4), while the transition e allows the "smooth" update of the strong normalization proof advocated by Clause (2), as we shall see. We present extended reduction in its parallel form to extend Definition 2.14, with respect to which we add the rules for the transitions t and e. Rule δ

$$\frac{L \vdash W_1 \rightrightarrows_{h,g} W_2 \quad L.\delta/\lambda W_1 \vdash T_1 \rightrightarrows_{h,g} T_2}{L \vdash \delta/\lambda W_1.T_1 \rightrightarrows_{h,g} \delta/\lambda W_2.T_2}\,\mathrm{bind} \qquad \frac{L \vdash V_1 \rightrightarrows_{h,g} V_2 \quad L \vdash T_1 \rightrightarrows_{h,g} T_2}{L \vdash \copyright/@V_1.T_1 \rightrightarrows_{h,g} \copyright/@V_2.T_2}\,\mathrm{flat}$$

$$\frac{}{L \vdash \star/\#i \rightrightarrows_{h,g} \star/\#i}\,\mathrm{atom} \qquad \frac{\downarrow_{\langle 0,i\rangle} L = K.\delta/\lambda W_1 \quad K \vdash W_1 \rightrightarrows_{h,g} W_2 \quad \uparrow^{\langle 0,i+1\rangle} W_2 = V_2}{L \vdash \#i \rightrightarrows_{h,g} V_2}\,\delta$$

$$\frac{L \vdash V_1 \rightrightarrows_{h,g} V_2 \quad L \vdash W_1 \rightrightarrows_{h,g} W_2 \quad L.\lambda W_1 \vdash T_1 \rightrightarrows_{h,g} T_2}{L \vdash @V_1.\lambda W_1.T_1 \rightrightarrows_{h,g} \delta(\copyright W_2.V_2).T_2}\,\beta \qquad \frac{g_h(k) = d + 1}{L \vdash \star k \rightrightarrows_{h,g} \star(h(k))}\,s$$

$$\frac{L \vdash V_1 \rightrightarrows_{h,g} V_2 \quad \uparrow^{\langle 0,1\rangle} V_2 = W_2 \quad L \vdash U_1 \rightrightarrows_{h,g} U_2 \quad L.\delta U_1 \vdash T_1 \rightrightarrows_{h,g} T_2}{L \vdash @V_1.\delta U_1.T_1 \rightrightarrows_{h,g} \delta U_2.@W_2.T_2}\,\theta$$

$$\frac{L.\delta V \vdash U_1 \rightrightarrows_{h,g} U_2 \quad \uparrow^{\langle 0,1\rangle} T_2 = U_2}{L \vdash \delta V.U_1 \rightrightarrows_{h,g} T_2}\,\zeta \qquad \frac{L \vdash T_1 \rightrightarrows_{h,g} T_2}{L \vdash \copyright U.T_1 \rightrightarrows_{h,g} T_2}\,\epsilon \qquad \frac{L \vdash U_1 \rightrightarrows_{h,g} U_2}{L \vdash \copyright U_1.T \rightrightarrows_{h,g} U_2}\,e$$

Fig. 27. Extended parallel reduction for terms (single step).

$$\frac{}{\star \vdash\rightrightarrows_{h,g} \star}\,\mathrm{atom} \qquad \frac{L_1 \vdash\rightrightarrows_{h,g} L_2 \quad L_1 \vdash W_1 \rightrightarrows_{h,g} W_2}{L_1.\delta/\lambda W_1 \vdash\rightrightarrows_{h,g} L_2.\delta/\lambda W_2}\,\mathrm{pair}$$

Fig. 28. Extended parallel reduction for environments (single step).

$$\frac{L \vdash T_1 \rightrightarrows T_2}{L \vdash T_1 \rightrightarrows^*_{h,g} T_2}\,\mathrm{inj} \qquad \frac{L \vdash T_1 \rightrightarrows^*_{h,g} T \quad L \vdash T \rightrightarrows_{h,g} T_2}{L \vdash T_1 \rightrightarrows^*_{h,g} T_2}\,\mathrm{step}$$

Fig. 29. Extended parallel computation for terms (multi-step).

is modified as well to include the support for transition l. Definition 2.15 and Definition 2.16 are extended accordingly. The point of extended reduction compared to static type assignment, is that its context rules allow to compute the static type in every subterm and not just along the “spine”.

Definition 2.27 (*extended parallel reduction for terms*). The relation $L \vdash T_1 \rightrightarrows_{h,g} T_2$ of Figure 27 indicates one step of extended parallel reduction from T_1 to T_2 in L. ▲

Definition 2.28 (*extended parallel reduction for environments*). Figure 28 defines $L_1 \vdash\rightrightarrows_{h,g} L_2$ indicating one step of extended parallel reduction from L_1 to L_2. ▲

Definition 2.29 (*extended parallel computation*). The relation $L \vdash T_1 \rightrightarrows^*_{h,g} T_2$ is the transitive closure of $L \vdash T_1 \rightrightarrows_{h,g} T_2$, while $L_1 \vdash\rightrightarrows^*_{h,g} L_2$ is the transitive closure of $L_1 \vdash\rightrightarrows_{h,g} L_2$. Figure 29 defines $L \vdash T_1 \rightrightarrows^*_{h,g} T_2$ for reference. $L_1 \vdash\rightrightarrows^*_{h,g} L_2$ is defined in the same manner. ▲

The main results on extended reduction are in Section 3.3.

2.9. Atomic Arity Assignment

Atomic arities are simple types representing the abstract syntax of our reducibility candidates, introduced in the next Section 2.10, and replace in this role the more complex “binary arities” used by Guidi [2009]. Such arities are assigned to terms according to well-established rules. The term “atomic” indicates that the base constructor of these arities is not structured.

$$\text{atomic arity} \quad A, B \quad ::= \star \mid B \supset A$$

$$\frac{}{L \vdash \star i : \star}\text{sort} \quad \frac{\downarrow_{\langle 0,i \rangle} L = K.\delta/\lambda W \quad K \vdash W : B}{L \vdash \#i : B}\text{lref} \quad \frac{L \vdash V : B \quad L.\delta V \vdash T : A}{L \vdash \delta V.T : A}\text{abbr}$$

$$\frac{L \vdash W : B \quad L.\lambda W \vdash T : A}{L \vdash \lambda W.T : B \supset A}\text{abst} \quad \frac{L \vdash V : B \quad L \vdash T : B \supset A}{L \vdash @V.T : A}\text{appl} \quad \frac{L \vdash U : A \quad L \vdash T : A}{L \vdash ©U.T : A}\text{cast}$$

Fig. 30. Atomic arities and their assignment.

$$\frac{}{\star \mathrel{\dot\subseteq:} \star}\text{atom} \quad \frac{L_1 \mathrel{\dot\subseteq:} L_2}{L_1.\delta/\lambda W \mathrel{\dot\subseteq:} L_2.\delta/\lambda W}\text{pair} \quad \frac{L_1 \mathrel{\dot\subseteq:} L_2 \quad L_1 \vdash ©W.V : B \quad L_2 \vdash W : B}{L_1.\delta(©W.V) \mathrel{\dot\subseteq:} L_2.\lambda W}\text{beta}$$

Fig. 31. Refinement for preservation of atomic arity.

$$\frac{\forall T_2.\ (L \vdash T_1 \rightrightarrows_{h,g} T_2) \Rightarrow (T_1 = T_2)}{L \vdash\rightrightarrows_{h,g} \mathbb{N}(T_1)}\text{cnx}$$

$$\frac{\forall T_2.\ (L \vdash T_1 \rightrightarrows_{h,g} T_2) \Rightarrow (T_1 \neq T_2) \Rightarrow (L \vdash \searrow^*_{h,g} T_2)}{L \vdash \searrow^*_{h,g} T_1}\text{csx}$$

Fig. 32. Normal terms and strongly normalizing terms for extended reduction.

Definition 2.30 (*atomic arities and their assignment*). Atomic arities are the simple types defined in Figure 30. $\star$ is the base type, and $B \supset A$ is the arrow type. Moreover the relation $L \vdash T : A$, defined in Figure 30 as well, assigns the arity A to T in L. ▲

As a type assignment, $L \vdash T : A$ has two interpretations: either A is the simple type of the object T, or A is the simple type associated to the type T. In this respect, consider the map $T \mapsto T^*$ that turns a term of $\lambda\delta$ into a term of $\lambda{\to}$ by operating the necessary $\delta\epsilon\zeta$-reductions on T and by replacing every abstraction in T, say λW in the environment K, with the abstraction λB such that $K \vdash W : B$. Moreover, extend this map to environment entries. Then the rules of Figure 30 clearly show that $L \vdash T : A$ implies $L^* \vdash T^* : A$ in $\lambda{\to}$ (we did not prove this fact formally yet).

We need the next refinement in order to prove the preservation of atomic arity.

Definition 2.31 (*refinement for preservation of atomic arity*). Figure 31 defines the relation $L_1 \mathrel{\dot\subseteq:} L_2$ stating that L_1 refines L_2 for preservation of atomic arity. ▲

Our results on atomic arity assignment and on its refinement are in Section 3.4.

2.10. Reducibility Candidates

The "reducibility candidates" are subsets of λ-terms satisfying certain "saturation" conditions used to establish properties of some typed λ-calculi. In this article we use subsets of closures, closed under the next seven conditions, to prove that every term having an atomic arity in an environment, is strongly normalizing with respect to extended reduction. We start by defining the normal terms and the strongly normalizing terms. These definitions take into account the fact that extended reduction is reflexive and forbids single-step cycles.

Definition 2.32 (*normal terms and strongly normalizing terms*). Figure 32 defines $L \vdash\rightrightarrows_{h,g} \mathbb{N}(T)$ and $L \vdash \searrow^*_{h,g} T$, stating respectively that T in L is normal, and that T in L is strongly normalizing, for extended reduction with respect to h and g_h. ▲

$$\frac{\langle L, @{\star}k.T\rangle \in \mathcal{R}}{\langle L, T\rangle \in \mathcal{R}}\text{S} \quad \frac{\downarrow_{\langle l,m\rangle} L = K \quad \langle K, T\rangle \in \mathcal{R} \quad \uparrow^{\langle l,m\rangle} T = U}{\langle L, U\rangle \in \mathcal{R}}\text{S0}$$

$$\frac{\langle L, T\rangle \in \mathcal{C}}{\langle L, T\rangle \in \mathcal{R}}\text{S1} \quad \frac{\langle L, \overline{V}\rangle \in \mathcal{R} \quad \mathbb{S}(T) \quad L \vdash\rightrightarrows_{h,g} \mathbb{N}(T)}{\langle L, @\overline{V}.T\rangle \in \mathcal{C}}\text{S2} \quad \frac{\langle L, \overline{V}\rangle \in \mathcal{R}}{\langle L, @\overline{V}.{\star}k\rangle \in \mathcal{C}}\text{S4}$$

$$\frac{\langle L, @\overline{V}.\delta(©W.V).T\rangle \in \mathcal{C}}{\langle L, @\overline{V}.@V.\lambda W.T\rangle \in \mathcal{C}}\text{S3} \quad \frac{\downarrow_{\langle 0,i\rangle} L = K.\delta/\lambda W_1 \quad \uparrow^{\langle 0,i+1\rangle} W_1 = W_2 \quad \langle L, @\overline{V}.W_2\rangle \in \mathcal{C}}{\langle L, @\overline{V}.\#i\rangle \in \mathcal{C}}\text{S5}$$

$$\frac{\langle L, V\rangle \in \mathcal{R} \quad \uparrow^{\langle 0,1\rangle} \overline{V_1} = \overline{V_2} \quad \langle L.\delta V, @\overline{V_2}.T\rangle \in \mathcal{C}}{\langle L, @\overline{V_1}.\delta V.T\rangle \in \mathcal{C}}\text{S6} \quad \frac{\langle L, @\overline{V}.U\rangle \in \mathcal{C} \quad \langle L, @\overline{V}.T\rangle \in \mathcal{C}}{\langle L, @\overline{V}.©U.T\rangle \in \mathcal{C}}\text{S7}$$

Fig. 33. Reducibility candidate.

$$\frac{\forall L, W, U, \overline{c}.\ \downarrow_{\overline{c}} L = K \Rightarrow \uparrow^{\overline{c}} T = U \Rightarrow \langle L, W\rangle \in \mathcal{C}_1 \Rightarrow \langle L, @W.U\rangle \in \mathcal{C}_2}{\langle K, T\rangle \in \mathcal{C}_1 \supset \mathcal{C}_2}\text{cfun}$$

Fig. 34. Function subset.

$$[\![\star]\!]_{\mathcal{R}} = \mathcal{R} \qquad [\![B \supset A]\!]_{\mathcal{R}} = [\![B]\!]_{\mathcal{R}} \supset [\![A]\!]_{\mathcal{R}}$$

Fig. 35. Interpretation of an atomic arity as a subset of closures.

Notice that $L \vdash \searrow^{*}_{h,g} T_1$ is inductively defined with base case $L \vdash\rightrightarrows_{h,g} \mathbb{N}(T_1)$. In fact, $L \vdash\rightrightarrows_{h,g} \mathbb{N}(T_1)$ implies $L \vdash \searrow^{*}_{h,g} T_1$ since $L \vdash \searrow^{*}_{h,g} T_2$ holds by "ex falso quodlibet".

Given a property $\mathcal{R}$ on closures, a reducibility candidate $\mathcal{C}$ for $\mathcal{R}$ is a subset of closures satisfying $\mathcal{R}$, that we describe constructively as a relation. So we may write $\mathcal{C}(T, L)$ for $\langle L, T\rangle \in \mathcal{C}$. Our reducibility theorem states that if $\mathcal{R}$ is a reducibility candidate, then every closure with an atomic arity belongs to some $\mathcal{C}$ and therefore, satisfies $\mathcal{R}$. In formal words we can prove that $L \vdash T : A$ implies $\mathcal{R}(T, L)$. Strong normalization follows from choosing $L \vdash \searrow^{*}_{h,g} T$ as $\mathcal{R}(T, L)$.

We are going to present Tait-style reducibility candidates [Tait 1975], which differ from the Girard-style reducibility candidates [Girard et al. 1989] used by Guidi [2009], in that condition "CR2" is not required (*i.e.*, $\langle L, T_1\rangle \in \mathcal{C}$ and $L \vdash T_1 \rightrightarrows_{h,g} T_2$ imply $\langle L, T_2\rangle \in \mathcal{C}$), and notably, in that closures without an arity are allowed in $\mathcal{C}$. This simplification gives us more freedom for constructing elements of $\mathcal{C}$.

Definition 2.33 (*reducibility candidate*). Given a subset $\mathcal{R}$ of closures satisfying Rule (S) and Rule (S0) of Figure 33, a reducibility candidate $\mathcal{C}$ for $\mathcal{R}$ is a subset of closures satisfying Rule (S1) to Rule (S7) of Figure 33. The notation "$\langle L, \overline{V}\rangle \in \mathcal{R}$" means "$\langle L, V\rangle \in \mathcal{R}$ for each component V of $\overline{V}$". ▲

Compound reducibility candidated are built through well-established constructions. For now we are interested just in the "functional" construction introduced next.

Definition 2.34 (*function subset*). If $\mathcal{C}_1$ and $\mathcal{C}_2$ are subsets of closures, then the subset $\mathcal{C}_1 \supset \mathcal{C}_2$ is defined in Figure 34. ▲

Notice that the environment L of W possibly extends the environment K of T. as is required to prove Figure 33(S6), in which L and $L.\delta V$ have diffrent length.

Definition 2.35 (*interpretation of an atomic arity*). For a subset of closures $\mathcal{R}$, the subset of closures $[\![A]\!]_{\mathcal{R}}$ associated to the atomic arity A, is defined in Figure 35. ▲

$$\frac{}{\star \mathrel{\dot{\subseteq}_{\mathcal{R}}} \star}\text{atom} \qquad \frac{L_1 \mathrel{\dot{\subseteq}_{\mathcal{R}}} L_2}{L_1.\delta/\lambda W \mathrel{\dot{\subseteq}_{\mathcal{R}}} L_2.\delta/\lambda W}\text{pair}$$

$$\frac{L_1 \mathrel{\dot{\subseteq}_{\mathcal{R}}} L_2 \quad \langle L_1, W\rangle \in [\![B]\!]_{\mathcal{R}} \quad \langle L_1, V\rangle \in [\![B]\!]_{\mathcal{R}} \quad L_2 \vdash W : B}{L_1.\delta(©W.V) \mathrel{\dot{\subseteq}_{\mathcal{R}}} L_2.\lambda W}\text{beta}$$

Fig. 36. Refinement for reducibility.

$$\frac{|L_1| = |L_2|}{L_1 \equiv_l^{\star k} L_2}\text{sort} \qquad \frac{|L_1| = |L_2| \quad i < l}{L_1 \equiv_l^{\# i} L_2}\text{skip} \qquad \frac{|L_1| = |L_2| \quad |L_1| \le i \quad |L_2| \le i}{L_1 \equiv_l^{\# i} L_2}\text{free}$$

$$\frac{l \le i \quad \downarrow_{\langle 0,i\rangle} L_1 = K_1.\delta/\lambda W \quad \downarrow_{\langle 0,i\rangle} L_2 = K_2.\delta/\lambda W \quad K_1 \equiv_0^W K_2}{L_1 \equiv_l^{\# i} L_2}\text{lref}$$

$$\frac{L_1 \equiv_l^W L_2 \quad L_1.\delta/\lambda W \equiv_{l+1}^T L_2.\delta/\lambda W}{L_1 \equiv_l^{\delta/\lambda W.T} L_2}\text{bind} \qquad \frac{L_1 \equiv_l^V L_2 \quad L_1 \equiv_l^T L_2}{L_1 \equiv_l^{©/@V.T} L_2}\text{flat}$$

Fig. 37. Lazy equivalence for environments.

The refinement given next is needed to state the general form of the reducibility theorem. In particular it expresses in $\lambda\delta$ a simultaneus substitution like the one occurring in the reducibility theorem for System F, which is stated using the "parametric" reducibility of Girard et al. [1989].

Definition 2.36 (*refinement for reducibility*). The relation $L_1 \mathrel{\dot{\subseteq}_{\mathcal{R}}} L_2$ defined in Figure 36, states that L_1 refines L_2 for reducibility. ▲

The main results on candidates and on their refinement are in Section 3.5.

2.11. Lazy Equivalence

In Section 2.10 we defined the normalization of a term T in the environment L that, by Theorem 3.7(5), is implied by $L \vdash T : A$. Now we would like to define the normalization of an environment L in such a way that $L \vdash T : A$ implies it as well. However, we notice from Figure 30(lref) that $L \vdash T : A$ constrains just the entries of L hereditarily referred by T. Thus, following the paradigm of Figure 32(csx), we need to replace $T_1 \neq T_2$ with the negated equivalence $L_1 \not\equiv^T L_2$ stating that L_1 and L_2 differ in one entry hereditarily referred by T. The corresponding equivalence is defined next. Working under the assumption that every entry of L has an arity, simplifies the development significantly, but we aim at showing that this assumption is redundant.

Definition 2.37 (*lazy equivalence for environments*). The relation $L_1 \equiv_l^T L_2$ defined in Figure 37, states that the environments L_1 and L_2 are equal in the entries hereditarily referred by the term T at level l. ▲

This relation is an equivalence that we term "lazy" since we check for equality just the entries of L_1 and L_2 hereditarily referred by T. Its nonrecursive definition (8) uses "hereditarily free" variables. We say that a variable is "hereditarily free" in $\langle L, T\rangle$ when it is free in T or in an entry of L hereditarily referred by T. This idea is expressed formally by the next definition. Alternatively, we can say that a variable is hereditarily free in $\langle L, T\rangle$ when it is free in a δl-reduct of T in L (see Definition 2.13 and Definition 2.26 for δ-reducts and l-reducts respectively).

Definition 2.38 (*hereditarily free variables*). Figure 38 defines $i \in \mathbb{F}_l^*\langle L, T\rangle$, stating that the variable introduced at depth i is hereditarily free at level l in $\langle L, T\rangle$. ▲

$$\frac{\forall T.\uparrow^{\langle i,1\rangle} T \neq U}{i \in \mathbb{F}^*_l\langle L, U\rangle}\text{eq}$$

$$\frac{l \le j \quad j < i \quad (\forall T.\uparrow^{\langle j,1\rangle} T \neq U) \quad \downarrow_{\langle 0,j\rangle} L = K.\delta/\lambda W \quad i-j-1 \in \mathbb{F}^*_0\langle K, W\rangle}{i \in \mathbb{F}^*_l\langle L, U\rangle}\text{be}$$

Fig. 38. Hereditarily free variables.

$$\frac{}{\star \Cup^U_l \star = \star}\text{atom}$$

$$\frac{L_1 \Cup^U_l L_2 = L \quad |L_1| \notin \mathbb{F}^*_l\langle \delta_1/\lambda_1 W_1.L_1, U\rangle}{\delta_1/\lambda_1 W_1.L_1 \Cup^U_l \delta_2/\lambda_2 W_2.L_2 = \delta_1/\lambda_1 W_1.L}\text{sn}$$

$$\frac{L_1 \Cup^U_l L_2 = L \quad l \le |L_1| \quad |L_1| \in \mathbb{F}^*_l\langle \delta_1/\lambda_1 W_1.L_1, U\rangle}{\delta_1/\lambda_1 W_1.L_1 \Cup^U_l \delta_2/\lambda_2 W_2.L_2 = \delta_2/\lambda_2 W_2.L}\text{dx}$$

Fig. 39. Pointwise union of environments.

$$\frac{\forall L_2.\ (L_1 \vdash\!\rightrightarrows_{h,g} L_2) \Rightarrow (L_1 \not\equiv^T_l L_2) \Rightarrow (\searrow\!\!\searrow^{*\,T}_{h,g,l} L_2)}{\searrow\!\!\searrow^{*\,T}_{h,g,l} L_1}\text{lsx}$$

Fig. 40. Strongly normalizing environments for extended reduction.

We need the level l to reason about hereditarily free variables in the scope of binders. For example we can prove that $i+1 \in \mathbb{F}^*_{l+1}\langle L.\delta/\lambda W, U\rangle$ implies $i \in \mathbb{F}^*_l\langle L, \delta/\lambda W.U\rangle$.

An ancillary operation that we term "pointwise union" at level l of L_1 and L_2 with respect to T (notation: $L_1 \Cup^T_l L_2$), leads to important properties connecting lazy equivalence and parallel reduction for environments such as Theorem 3.9(6). The environment $L_1 \Cup^T_l L_2$ is defined when $|L_1| = |L_2|$ and its i-th entry is taken from L_2 if $l \le i$ and $i \in \mathbb{F}^*_l\langle L_1, T\rangle$, or else it is taken from L_1.

Definition 2.39 (*pointwise union*). The partial operation $L_1 \Cup^T_l L_2$ defined in Figure 39, constructs the "pointwise union" at level l of L_1 and L_2 with respect to T. ▲

Lazy equivalence yields environments L normalizing with respect to T (notation $\searrow\!\!\searrow^{*\,T}_{h,g,l} L$) such that $L \vdash \searrow\!\!\searrow^*_{h,g} T$ implies $\searrow\!\!\searrow^{*\,T}_{h,g,l} L$ for every level l. See Theorem 3.10(3).

Definition 2.40 (*strongly normalizing environments*). Figure 40 defines the relation $\searrow\!\!\searrow^{*\,T}_{h,g,l} L$, stating that L is strongly normalizing at level l for extended reduction with respect to the parameters h and g_h, and with respect to T. ▲

Notice the common structure of Figure 40(lsx) and Figure 32(csx).

An ancillary predicate on environments $\sim\searrow\!\!\searrow^*_{h,g,l} L$ is needed in Theorem 3.10(1). It serves $\searrow\!\!\searrow^{*\,T}_{h,g,l} L$ as, for instance, $L_1 \vdash\!\rightrightarrows L_2$ serves $L \vdash T_1 \rightrightarrows T_2$ in Theorem 3.1(3).

Definition 2.41 (*strongly co-normalizing environments*). The predicate $\sim\searrow\!\!\searrow^*_{h,g,l} L$ defined in Figure 41, states that the L is "co-normalizing" at level l with respect to h and g_h. This means that every i-th entry of L such that $i < l$, is strongly normalizing according to Definition 2.40. "Co-normalizing" refers to "$i < l$" as opposed to "$l \le i$". ▲

The main results on lazy equivalence, pointwise union, and strongly normalizing environments are in Section 3.6. Comparing Section 2.2 with Section 2.11, the reader

$$\frac{}{\sim\Searrow^{*}_{h,g,l}\,\star}\,\text{atom}\qquad\frac{\sim\Searrow^{*}_{h,g,0}\,L}{\sim\Searrow^{*}_{h,g,0}(L.\delta/\lambda W)}\,\text{skip}\qquad\frac{\sim\Searrow^{*}_{h,g,l}\,L\qquad\Searrow^{*\,W}_{h,g,l}\,L}{\sim\Searrow^{*}_{h,g,l+1}(L.\delta/\lambda W)}\,\text{pair}$$

Fig. 41. Strongly co-normalizing environments for extended reduction.

$$\frac{L\vdash T_1\rightrightarrows_{h,g} T_2}{\langle L,T_1\rangle\geq_{h,g}\langle L,T_2\rangle}\,\text{cpx}\qquad\frac{L_1\vdash\rightrightarrows_{h,g} L_2}{\langle L_1,T\rangle\geq_{h,g}\langle L_2,T\rangle}\,\text{lpx}$$

$$\frac{\langle L_1,T_1\rangle\sqsupset^{?}\langle L_2,T_2\rangle}{\langle L_1,T_1\rangle\geq_{h,g}\langle L_2,T_2\rangle}\,\text{fquq}\qquad\frac{L_1\equiv^{T}_{0} L_2}{\langle L_1,T\rangle\geq_{h,g}\langle L_2,T\rangle}\,\text{lleq}$$

$$\frac{\langle L_1,T_1\rangle\geq_{h,g}\langle L_2,T_2\rangle}{\langle L_1,T_1\rangle\geq_{h,g}\langle L_2,T_2\rangle}\,\text{inj}\qquad\frac{\langle L_1,T_1\rangle\geq_{h,g}\langle L,T\rangle\qquad\langle L,T\rangle\geq_{h,g}\langle L_2,T_2\rangle}{\langle L_1,T_1\rangle\geq_{h,g}\langle L_2,T_2\rangle}\,\text{step}$$

Fig. 42. $qrst$-reduction and $qrst$-computation.

should notice that the notions defined here depend just on the component l of the relocation pair $\langle l,m\rangle$. In this perspective, the given definitions are the general ones instantiated for $m=\infty$. We present them in this form because the parameter m turns out to be unnecessary for now.

2.12. Very Big Trees

With the help of lazy equivalence, we can finally define our counterpart of "rst-reduction" [de Vrijer 1994], which we informally introduced in Section 2.8. This counterpart is actually an extension that operates on closures. We term it "$qrst$-reduction" because we add a "q-step" of lazy equivalence.

Definition 2.42 (*$qrst$-reduction and $qrst$-computation*). The relation $\langle L_1,T_1\rangle\geq_{h,g}\langle L_2,T_2\rangle$ defined in Figure 42, denotes one step of $qrst$-reduction from the closure $\langle L_1,T_1\rangle$ to the closure $\langle L_2,T_2\rangle$ with respect to the parameters h and g_h. The relation $\langle L_1,T_1\rangle\geq_{h,g}\langle L_2,T_2\rangle$ ($qrst$-computation), defined in Figure 42 as well, is the is the (reflexive and) transitive closure of $\langle L_1,T_1\rangle\geq_{h,g}\langle L_2,T_2\rangle$. ▲

Figure 42(fquq) is the "s-step", Figure 42(cpx) is the "rt-step" for terms, Figure 42(lpx) is the "rt-step" for environments, and Figure 42(lleq) is our new "q-step". Because of it, our "big" trees are actually "very big" with respect to de Vrijer [1994]. Formally, the "very big" tree rooted at $\langle L,T\rangle$ comprises the $qrst$-computations starting at $\langle L,T\rangle$. Our "very big tree" theorem states that if T has an atomic arity in L (Section 2.9), then the nonreflexive rst-steps in this tree are finite.

In order to state the theorem, the next definition highlights the proper (*i.e.*, nonreflexive) rst-steps and the $qrst$-computations containing them.

Definition 2.43 (*proper rst-reduction and proper $qrst$-computation*). Figure 43 defines the relation $\langle L_1,T_1\rangle>_{h,g}\langle L_2,T_2\rangle$, denoting one step of proper rst-reduction from $\langle L_1,T_1\rangle$ to $\langle L_2,T_2\rangle$ with respect to the parameters h and g_h, and the relation $\langle L_1,T_1\rangle>\equiv_{h,g}\langle L_2,T_2\rangle$, denoting a proper $qrst$-computation. ▲

Theorem 3.11(2) shows that a step of proper rst-reduction is never reflexive, but a proper $qrst$-computation may be. Consider the term $@\Delta_{k,T}.\Delta_{k,T}$ where $\Delta_{k,T}=\lambda T.@\star k.@\#0.\#0$. Following the example of $@\Delta_T.\Delta_T$ in Section 2.3, we can prove $L\vdash @\Delta_{k,T}.\Delta_{k,T}\rightrightarrows @\star k.@\Delta_{k,T}.\Delta_{k,T}$ (proper r-step), and then $\langle L,@\star k.@\Delta_{k,T}.\Delta_{k,T}\rangle\sqsupset\langle L,@\Delta_{k,T}.\Delta_{k,T}\rangle$ (s-step) by Figure 24(flat). Moreover by Theorem 3.11(4), starting a proper $qrst$-computation with a proper step, is not restrictive.

$$\frac{\langle L_1, T_1\rangle \sqsupset \langle L_2, T_2\rangle}{\langle L_1, T_1\rangle \succ_{h,g} \langle L_2, T_2\rangle}\,\mathrm{fqu} \qquad \frac{L \vdash T_1 \rightrightarrows_{h,g} T_2 \quad T_1 \neq T_2}{\langle L, T_1\rangle \succ_{h,g} \langle L, T_2\rangle}\,\mathrm{cpx} \qquad \frac{L_1 \vdash\rightrightarrows_{h,g} L_2 \quad L_1 \not\equiv_0^T L_2}{\langle L_1, T\rangle \succ_{h,g} \langle L_2, T\rangle}\,\mathrm{lpx}$$

$$\frac{\langle L_1, T_1\rangle \succ_{h,g} \langle L, T\rangle \quad \langle L, T\rangle \geq_{h,g} \langle L_2, T_2\rangle}{\langle L_1, T_1\rangle >\equiv_{h,g} \langle L_2, T_2\rangle}\,\mathrm{fpbg}$$

Fig. 43. Proper rst-reduction and proper $qrst$-computation.

$$\frac{L_1 \equiv_l^T L_2}{\langle L_1, T\rangle \equiv_l \langle L_2, T\rangle}\,\mathrm{fleq} \qquad \frac{\forall L_2, T_2.\ \langle L_1, T_1\rangle \succ_{h,g} \langle L_2, T_2\rangle \Rightarrow \trianglerighteq_{h,g} \langle L_2, T_2\rangle}{\trianglerighteq_{h,g} \langle L_1, T_1\rangle}\,\mathrm{fsb}$$

Fig. 44. q-equivalence and strongly rst-normalizing closures.

Now we can define the closures whose "very big" tree contains a finite number of nonreflexive rst-steps. This is achieved by standard means with the next definition.

Definition 2.44 (*q-equivalence and strongly rst-normalizing closures*). Figure 43 defines the relation $\langle L_1, T_1\rangle \equiv_l \langle L_2, T_2\rangle$ (q-equivalence) that extends lazy equivalence to closures, and the predicate $\trianglerighteq_{h,g} \langle L, T\rangle$ stating that $\langle L, T\rangle$ is strongly normalizing for $qrst$-reduction with respect to the parameters h and g_h. ▲

Theorem 3.11(2) and Theorem 3.11(3) show that $\langle L_1, T_1\rangle \succ_{h,g} \langle L_2, T_2\rangle$ is equivalent to $\langle L_1, T_1\rangle \succeq_{h,g} \langle L_2, T_2\rangle$ & $\langle L_1, T_1\rangle \not\equiv_0 \langle L_2, T_2\rangle$, so we can rephrase Figure 44(fsb) following the pattern of Figure 32(csx) and Figure 40(lsx). Moreover $\trianglerighteq_{h,g} \langle L, T\rangle$ can be generated by Rule (11), which is Figure 44(fsb) with $\langle L_1, T_1\rangle >\equiv_{h,g} \langle L_2, T_2\rangle$ in place of $\langle L_1, T_1\rangle \succ_{h,g} \langle L_2, T_2\rangle$. So $\langle L, T\rangle$ is strongly rst-normalizing iff it is strongly $qrst$-normalizing.

Our results on $qrst$-computations and $qrst$-normalization are in Section 3.7.

3. PROPOSITIONS ON $\lambda\delta$

In this section we present the main properties of reduction (Section 3.1), of degree assignment (Section 3.2), of rt-reduction (Section 3.3), of atomic arity assignment (Section 3.4), of reducibility candidates (Section 3.5), of lazy equivalence (Section 3.6), of $qrst$-reduction (Section 3.7), and finally of stratified validity (Section 3.8) respecting the dependences between these properties.

We aim at reaching our versions of the "three problems" [Nederpelt et al. 1994]: Theorem 3.2(1) (confluence of computation), Theorem 3.12(2) (strong $qrst$-normalization of valid terms), and Theorem 3.15(6) (subject reduction of stratified validity).

The detailed theory of $\lambda\delta$ (1416 proofs) exists only in the digital form of Guidi [2014]. In this article we just outline the proofs of the presented statements by reporting on the proof strategy and on the main dependences of each proof. Most proofs are by induction on the height of a derivation or by cases on the last step of a derivation. Very often both techniques are applied together.

Appendix B lists the pointers to the digital proofs outlined in the article.

3.1. Results on Reduction

The relevant properties of reduction, conversion, and their refinement are listed next.

THEOREM 3.1 (REDUCTION AND ITS REFINEMENT).

(*1*) *(transitivity of refinement)*
If $L_1 \dot{\subseteq} L$ *and* $L \dot{\subseteq} L_2$ *then* $L_1 \dot{\subseteq} L_2$.

(*2*) *(transitivity of reduction for terms through refinement)*
If $L_1 \dot{\subseteq} L_2$ *and* $L_2 \vdash T_1 \rightrightarrows T_2$ *then* $L_1 \vdash T_1 \rightrightarrows T_2$.

(*3*) *(confluence of reduction for terms with itself, diamond property, general form)*
If $L_0 \vdash T_0 \rightrightarrows T_1$ *and* $L_0 \vdash T_0 \rightrightarrows T_2$ *and* $L_0 \vdash\rightrightarrows L_1$ *and* $L_0 \vdash\rightrightarrows L_2$ *then there exists* T *such that* $L_1 \vdash T_1 \rightrightarrows T$ *and* $L_2 \vdash T_2 \rightrightarrows T$.
(*4*) *(confluence of reduction for environments with itself, diamond property)*
If $L_0 \vdash\rightrightarrows L_1$ *and* $L_0 \vdash\rightrightarrows L_2$ *then there exists* L *such that* $L_1 \vdash\rightrightarrows L$ *and* $L_2 \vdash\rightrightarrows L$.

PROOF. Clause (1) is proved by induction on its first premise and by cases on its second premise. Clause (2) is proved by induction on its second premise. Clause (3) is proved by induction on the proper subclosures of $\langle L_0, T_0\rangle$ (Section 2.7) and by cases on its four premises. Reduction for environments emerges when considering Figure 14(δ) and when a binder in the "spine" of T_0 is pushed into L_0 in the cases of Figure 14(bind), Figure 14(β), and Figure 14(θ). Moreover, Clause (2) and Figure 17(beta) are invoked when Figure 14(β) is considered. Clause (4) is proved by induction on $|L_0|$ and by cases on its two premises with the help of Clause (3). ▲

THEOREM 3.2 (COMPUTATION AND CONVERSION).

(*1*) *(confluence of computation for terms with itself, Church-Rosser property)*
If $L \vdash T_0 \rightrightarrows^* T_1$ *and* $L \vdash T_0 \rightrightarrows^* T_2$ *then there exists* T *such that* $L \vdash T_1 \rightrightarrows^* T$ *and* $L \vdash T_2 \rightrightarrows^* T$.
(*2*) *(confluence of computation for environments with itself, Church-Rosser property)*
If $L_0 \vdash\rightrightarrows^* L_1$ *and* $L_0 \vdash\rightrightarrows^* L_2$ *then there exists* L *such that* $L_1 \vdash\rightrightarrows^* L$ *and* $L_2 \vdash\rightrightarrows^* L$.
(*3*) *(formulation of conversion as a pair of confluent computations)*
If $L \vdash T_1 \leftrightarrows^* T_2$ *then there exists* T *such that* $L \vdash T_1 \rightrightarrows^* T$ *and* $L \vdash T_2 \rightrightarrows^* T$.

PROOF. Clause (1) and Clause (2) are proved by induction on their first premise by invoking the corresponding "strip" lemmas [Barendregt 1993] from Theorem 3.1(3) and Theorem 3.1(4) respectively. Clause (3) is proved by induction on its premise with the help of the "strip" lemma from Theorem 3.1(3). ▲

The main result on reduction is Church-Rosser property, also known as the confluence theorem and one of the so-called "three problems" in the Automath tradition. The main result on conversion is its formulation as a pair of confluent computations: one direction is Theorem 3.2(3), the reverse is straightforward. Using this formulation, Theorem 3.1(3) and Theorem 3.2(1), give the generation lemma on abstraction, a desired property mentioned by van Daalen [1994]. This lemma states that $L \vdash \lambda W_1.T_1 \leftrightarrows^* \lambda W_2.T_2$ implies $L \vdash W_1 \leftrightarrows^* W_2$ and $L.\lambda W_1 \vdash T_1 \leftrightarrows^* T_2$.

3.2. Results on Degree Assignment

The relevant properties of degree assignment and of its refinement are listed next.

THEOREM 3.3 (DEGREE ASSIGNMENT AND ITS REFINEMENT).

(*1*) *(equivalence of degree assignment and iterated static type assignment, left to right)*
If $L \vdash T \blacksquare_{h,g}\, d$ *then for each* n *there exists* U *such that* $L \vdash T \bullet_h^{*(n)} U$ *and* $L \vdash U \blacksquare_{h,g}\, d-n$.
(*2*) *(equivalence of degree assignment and iterated static type assignment, right to left)*
If $L \vdash T \bullet_h^{*(n)} U$ *then for each* g_h *there exists* d *such that* $L \vdash T \blacksquare_{h,g}\, d$ *then* $L \vdash U \blacksquare_{h,g}\, d-n$.
(*3*) *(equivalence of degree assignment and iterated static type assignment, variant)*
If $L \vdash T \bullet_h^{*(n)} U$ *then for every* n_0 *there exist* g_h *and* $d \geq n_0$ *such that* $L \vdash T \blacksquare_{h,g}\, d$ *and* $L \vdash U \blacksquare_{h,g}\, d-n$.
(*4*) *(inclusion of refinement)*
If $L_1 \mathrel{\dot{\subseteq}}_{\blacksquare_{h,g}} L_2$ *then* $L_1 \mathrel{\dot{\subseteq}} L_2$.

(5) *(transitivity of degree assignment through refinement)*
If $L_1 \dot{\subseteq}_{\blacksquare h,g} L_2$ *and* $L_2 \vdash T \blacksquare_{h,g} d$ *then* $L_1 \vdash T \blacksquare_{h,g} d$.
(6) *(confluence of refinement and degree assignment)*
If $L_1 \dot{\subseteq}_{\blacksquare h,g} L_2$ *and* $L_1 \vdash T \blacksquare_{h,g} d$ *then* $L_2 \vdash T \blacksquare_{h,g} d$.
(7) *(transitivity of refinement)*
If $L_1 \dot{\subseteq}_{\blacksquare h,g} L$ *and* $L \dot{\subseteq}_{\blacksquare h,g} L_2$ *then* $L_1 \dot{\subseteq}_{\blacksquare h,g} L_2$.

PROOF. Clause (1), Clause (2), Clause (3), and Clause (4) are proved by induction on the premise. Clause (5) and Clause (6) are proved by induction on the second premise and by cases on the first premise. Clause (7) is proved by induction on its first premise and by cases on its second premise by invoking Clause (5) and Clause (6). ▲

Theorem 3.3(1) and Theorem 3.3(3) are the main properties of degree assignment, from which we derive the next Theorem 3.4(2) (notice that in [Guidi 2009] we were able to prove it just for $n = 0$).

THEOREM 3.4 (ITERATED STATIC TYPE ASSIGNMENT).

(1) *(uniqueness of iterated static type assignment)*
If $L \vdash T \bullet_h^{*(n)} U_1$ *and* $L \vdash T \bullet_h^{*(n)} U_2$ *then* $U_1 = U_2$.
(2) *(irreflexivity of static type assignment iterated at least once)*
$L \vdash T \bullet_h^{*(n+1)} T$ *is contradictory.*

PROOF. Clause (1) is proved by induction on its first premise and by cases on its second premise. Clause (2) is proved directly with the help of Theorem 3.3(3). ▲

3.3. Results on Extended Reduction

The relevant properties of extended reduction are listed next.

THEOREM 3.5 (EXTENDED REDUCTION).

(1) *(transitivity of extended reduction for terms through refinement)*
If $L_1 \dot{\subseteq} L_2$ *and* $L_2 \vdash T_1 \rightrightarrows_{h,g} T_2$ *then* $L_1 \vdash T_1 \rightrightarrows_{h,g} T_2$.
(2) *(inclusion of reduction, "r-step")*
If $L \vdash T_1 \rightrightarrows T_2$ *then* $L \vdash T_1 \rightrightarrows_{h,g} T_2$.
(3) *(inclusion of static type assignment, "t-step")*
If $L \vdash T_1 \bullet_h^{*(1)} T_2$ *and* $L \vdash T_1 \blacksquare_{h,g} d+1$ *then* $L \vdash T_1 \rightrightarrows_{h,g} T_2$.
(4) *(commutation of direct subclosure with extended reduction for terms)*
If $\langle L, T_1 \rangle \sqsupset \langle K, V_1 \rangle$ *and* $K \vdash V_1 \rightrightarrows_{h,g} V_2$ *then there exists* T_2 *such that* $L \vdash T_1 \rightrightarrows_{h,g} T_2$ *and* $\langle L, T_2 \rangle \sqsupset \langle K, V_2 \rangle$.
(5) *(commutation of extended reduction for environments with direct subclosure)*
If $L_1 \vdash\rightrightarrows_{h,g} L_2$ *and* $\langle L_2, T_2 \rangle \sqsupset \langle K_2, V \rangle$ *then there exist* K_1 *and* T *such that* $L_1 \vdash T_2 \rightrightarrows_{h,g} T$ *and* $\langle L_1, T \rangle \sqsupset \langle K_1, V \rangle$ *and* $K_1 \vdash\rightrightarrows_{h,g} K_2$.
(6) *(absorption of extended reduction for environments)*
If $L_1 \vdash\rightrightarrows_{h,g} L_2$ *and* $L_2 \vdash T_1 \rightrightarrows_{h,g} T_2$ *then* $L_1 \vdash T_1 \rightrightarrows^*_{h,g} T_2$.
(7) *(extended computation from a β-redex)*
If $L \vdash @V.\lambda W.T_1 \rightrightarrows^*_{h,g} T_2$ *then either* $@V.\lambda W.T_1 \approx T_2$ *or* $L \vdash \delta(©W.V).T_1 \rightrightarrows^*_{h,g} T_2$.

PROOF. Clause (1) is proved by induction on its second premise and by cases on its first premise. For the reference to a declaration, Figure 27(δ), we have $T_1 = \#i$, and $\downarrow_{\langle 0,i \rangle} L_2 = K_2.\lambda W_1$, and $K_2 \vdash W_1 \rightrightarrows_{h,g} W_2$, and $\uparrow^{\langle 0,i+1 \rangle} W_2 = T_2$. It may be the case, not occurring with ordinary reduction, that $\downarrow_{\langle 0,i \rangle} L_1 = K_1.\delta(©W_1.V_1)$ and $K_1 \dot{\subseteq} K_2$ for some K_1 and V_1 by Figure 17(beta). In that event the induction hypothesis yields $K_1 \vdash W_1 \rightrightarrows_{h,g} W_2$ and Figure 27(e) gives $K_1 \vdash ©W_1.V_1 \rightrightarrows_{h,g} W_2$ so Figure 27(δ) concludes

$L_1 \vdash \#i \rightrightarrows_{h,g} T_2$. Here we see the purpose of e-reduction and of the expected type W_1 in the β-reduced item $\delta(©W_1.V_1)$. The untyped β-reduced item δV_1 of Guidi [2009] shows here its weakness causing Clause (7) to fail. Clause (2) is proved by induction on its premise. Clause (3) is proved by induction on its first premise and by cases on its second premise after replacing $L \vdash T_1 \bullet_h^{*(1)} T_2$ with $L \vdash T_1 \bullet_h^{*(n)} T_2$ and $n = 1$. Clause (4) is proved by cases on its first premise. Clause (5) is proved by cases on its second premise and then by cases on its first premise. Clause (6) is proved by induction on its second premise and by cases on its first premise. Clause (7) is proved directly with the help of Clause (1) and Figure 17(beta). ▲

The "transitivity through refinement", Theorem 3.1(2) and Theorem 3.5(1), is the crucial property that holds for ordinary reduction and that extended reduction must preserve in order to guarantee the "smooth" update of the strong normalization proof advocated in Section 2.8. In particular, extended reduction preserves Theorem 3.5(7), and thus preserves the saturation condition of Figure 33(S3) for the subset of strongly normalizing closures. Another interesting property of extended reduction is the "square" of Theorem 3.5(4), which improves the "pentagon" of Rule (5). Notice that a transition l makes the fifth "side" disappear.

Unfortunately, the "pentagon" remains in Theorem 3.5(5), where the extended reduction for terms is needed in the case $L_2 = K_2.\delta/\lambda V$ and $T_2 = \#0$.

Theorem 3.5(6) shows that extended computation for environments is generated by the next rules that resemble Figure 28. The same holds for ordinary computation.

$$\frac{}{\star \vdash\rightrightarrows^*_{h,g} \star}\,\mathrm{atom} \qquad \frac{L_1 \vdash\rightrightarrows^*_{h,g} L_2 \quad L_1 \vdash W_1 \rightrightarrows^*_{h,g} W_2}{L_1.\delta/\lambda W_1 \vdash\rightrightarrows^*_{h,g} L_2.\delta/\lambda W_2}\,\mathrm{pair} \tag{6}$$

3.4. Results on Atomic Arity Assignment

The properties of atomic arity assignment and of its refinement are listed next.

THEOREM 3.6 (ARITY ASSIGNMENT AND ITS REFINEMENT).

(*1*) *(inclusion of refinement)*
If $L_1 \dot{\subseteq}\!: L_2$ *then* $L_1 \dot{\subseteq} L_2$.
(*2*) *(transitivity of assignment through refinement)*
If $L_1 \dot{\subseteq}\!: L_2$ *and* $L_2 \vdash T : A$ *then* $L_1 \vdash T : A$.
(*3*) *(confluence of refinement and assignment)*
If $L_1 \dot{\subseteq}\!: L_2$ *and* $L_1 \vdash T : A$ *then* $L_2 \vdash T : A$.
(*4*) *(transitivity of refinement)*
If $L_1 \dot{\subseteq}\!: L$ *and* $L \dot{\subseteq}\!: L_2$ *then* $L_1 \dot{\subseteq}\!: L_2$.
(*5*) *(uniqueness of atomic arities)*
If $L \vdash T : A_1$ *and* $L \vdash T : A_2$ *then* $A_1 = A_2$.
(*6*) *(inclusion of assignment)*
If $L \vdash T : A$ *then for each* h *and* n *then there exists* U *such that* $L \vdash T \bullet_h^{*(n)} U$ *and* $L \vdash U : A$.
(*7*) *(preservation of atomic arity through extended reduction, general form)*
If $L_1 \vdash T_1 : A$ *and* $L_1 \vdash T_1 \rightrightarrows_{h,g} T_2$ *and* $L_1 \vdash\rightrightarrows_{h,g} L_2$ *then* $L_2 \vdash T_2 : A$.

PROOF. Clause (1) and Clause (6) are proved by induction on their premise. Clause (2) and Clause (3) are proved by induction on their second premise and by cases on their first premise. Clause (4) is proved by induction on its first premise and by cases on its second premise with the help of Clause (2) and Clause (3). Clause (5) is proved by induction on its first premise and by cases on its second premise. Clause (7) is proved by induction on its first premise by cases on its second premise and then by cases on its third premise. As for Theorem 3.1(3), reduction for environments emerges when considering Figure 27(δ) and when a binder in the "spine" of T_1 is pushed into L_1 in the cases of Figure 27(bind), Figure 27(β), and Figure 27(θ). Moreover, Clause (2) and Figure 31(beta) are invoked when Figure 27(β) is considered. ▲

Theorem 3.6(7) (proposition 500 of Guidi [2014]) states the "subject reduction" property of the arity assignment, a prerequisite for the preservation Theorem 3.15.

3.5. Results on Reducibility Candidates

The properties of reducibility candidates and of their refinement are listed next.

THEOREM 3.7 (REDUCIBILITY CANDIDATES AND THEIR REFINEMENT).

(1) *(the candidate of strongly normalizing closures for extended reduction)*
For any h and g_h, the subset $\{\langle L, T\rangle \mid L \vdash \searrow^*_{h,g} T\}$ is a reducibility candidate for itself.
(2) *(the candidate associated to an atomic arity)*
If $\mathcal{R}$ is a reducibility candidate for itself then $[\![A]\!]_{\mathcal{R}}$ is a reducibility candidate for $\mathcal{R}$.
(3) *(reducibility theorem for extended reduction, general form)*
If $\mathcal{R}$ is a reducibility candidate for itself then $L_1 \mathrel{\dot{\subseteq}_{\mathcal{R}}} L_2$ and $\downarrow_{\overline{c}} L_2 = K_2$ and $K_2 \vdash T : A$ and $\uparrow^{\overline{c}} T = U$ imply $\langle L_1, U\rangle \in [\![A]\!]_{\mathcal{R}}$.
(4) *(reducibility theorem for extended reduction)*
If $\mathcal{R}$ is a reducibility candidate for itself then $L \vdash T : A$ implies $\langle L, T\rangle \in \mathcal{R}$.
(5) *(strong normalization theorem for extended reduction)*
If $L \vdash T : A$ then $L \vdash \searrow^*_{h,g} T$.
(6) *(inclusion of refinement)*
If $L_1 \mathrel{\dot{\subseteq}_{\mathcal{R}}} L_2$ then $L_1 \mathrel{\dot{\subseteq}} L_2$.
(7) *(inverse inclusion of refinement)*
If $\mathcal{R}$ is a reducibility candidate for itself then $L_1 \mathrel{\dot{\subseteq}\vdots} L_2$ implies $L_1 \mathrel{\dot{\subseteq}_{\mathcal{R}}} L_2$.

PROOF. Clause (1) is proved directly by invoking Theorem 3.5(7) and similar propositions (one for each extended redex). Clause (2) is proved by induction on A. Clause (3) is proved by induction on $K_2 \vdash T : A$ and by cases on the other premises by invoking Clause (2). Multiple relocation emerges from Figure 34(cfun), while the refinement emerges since Figure 36(beta) is needed when T is a λ-abstraction. Theorem 3.6(5) is invoked when T is a reference to a declaration in the case of Figure 36(beta). Clause (4) is a corollary of Clause (3) and of Figure 33(S1). Clause (5) is a corollary of Clause (4) and of Clause (1). Clause (6) is proved by induction on its premise. Clause (7) is proved by induction on its premise with the help of Clause (3). ▲

Theorem 3.7(1) is the most relevant property of strongly normalizing terms. Moreover the relation $L \vdash \searrow^*_{h,g} T$ is generated by the next rule resembling Figure 32(csx).

$$\frac{\forall T_2.\ (L \vdash T_1 \rightrightarrows^*_{h,g} T_2) \Rightarrow (T_1 \neq T_2) \Rightarrow (L \vdash \searrow^*_{h,g} T_2)}{L \vdash \searrow^*_{h,g} T_1}\,\text{csx} \tag{7}$$

3.6. Results on Lazy Equivalence

The relevant properties of pointwise union and lazy equivalence are listed next.

We give alternative definitions of lazy equivalence. The nonrecursive definition (8) is more appropriate for the proofs we shall present. A nonrecursive definition of pointwise union in the style of (8) is available as well. It is not easy to read, though.

$$\frac{|L_1| = |L_2| \quad \begin{pmatrix}\forall K_1, K_2, W_1, W_2, i.\\ l \le i \Rightarrow i \in \mathbb{F}^*_l\langle L_1, T\rangle \Rightarrow {\downarrow_{\langle 0,i\rangle}} L_1 = K_1.\delta_1/\lambda_1 W_1 \Rightarrow {\downarrow_{\langle 0,i\rangle}} L_2 = K_2.\delta_2/\lambda_2 W_2 \Rightarrow\\ \delta_1/\lambda_1 = \delta_2/\lambda_2 \ \&\ W_1 = W_2\end{pmatrix}}{L_1 \equiv^T_l L_2}\,\text{lleq} \tag{8}$$

$$\frac{|L_1| = |L_2| \quad \begin{pmatrix}\forall K_1, K_2, W_1, W_2, i.\\ l \le i \Rightarrow (\forall U.\ {\uparrow^{\langle i,1\rangle}} U \ne T) \Rightarrow {\downarrow_{\langle 0,i\rangle}} L_1 = K_1.\delta_1/\lambda_1 W_1 \Rightarrow {\downarrow_{\langle 0,i\rangle}} L_2 = K_2.\delta_2/\lambda_2 W_2 \Rightarrow\\ \delta_1/\lambda_1 = \delta_2/\lambda_2 \ \&\ W_1 = W_2 \ \&\ K_1 \equiv^{W_1}_0 K_2\end{pmatrix}}{L_1 \equiv^T_l L_2}\,\text{lleq} \tag{9}$$

THEOREM 3.8 (POINTWISE UNION).

(1) *(construction lemma for tail binder, positive case)*
If $|L_1| \in \mathbb{F}^*_l\langle \delta_1/\lambda_1 W_1.L_1, U\rangle$ *and* $l \le |L_1|$ *then*
$L_1 \Cup^U_l L_2 = L$ *implies* $(\delta_1/\lambda_1 W_1.L_1) \Cup^U_l (\delta_2/\lambda_2 W_2.L_2) = \delta_2/\lambda_2 W_2.L$.

(2) *(construction lemma for tail binder, negative case)*
If $|L_1| \notin \mathbb{F}^*_l\langle \delta_1/\lambda_1 W_1.L_1, U\rangle$ *then*
$L_1 \Cup^U_l L_2 = L$ *implies* $(\delta_1/\lambda_1 W_1.L_1) \Cup^U_l (\delta_2/\lambda_2 W_2.L_2) = \delta_1/\lambda_1 W_1.L$.

(3) *(existence lemma)*
If $|L_1| = |L_2|$ *then for each* T *and* l *then there exists* L *such that* $L_1 \Cup^T_l L_2 = L$.

PROOF. Clause (1) and Clause (2) are proved by cases on their last premise. Clause (3) is proved by induction on $|L_1|$ with the help of Clause (1) and Clause (2). ▲

Theorem 3.8(3) (proposition 1400 of Guidi [2014]) needs tail binders (Definition 2.4).

THEOREM 3.9 (LAZY EQUIVALENCE).

(1) *(left operand lemma)*
If $L_1 \equiv^T_l L_2$ *and* $L_2 \vdash\rightrightarrows_{h,g} K_2$ *and* $L_1 \Cup^T_l K_2 = K_1$ *then* $L_1 \vdash\rightrightarrows_{h,g} K_1$.

(2) *(right operand lemma)*
If $L_1 \equiv^T_l L_2$ *and* $L_2 \vdash\rightrightarrows_{h,g} K_2$ *and* $L_1 \Cup^T_l K_2 = K_1$ *then* $K_2 \equiv^T_l K_1$.

(3) *(transitivity with ranged equivalence)*
If $L_1 \equiv^T_l L$ *and* $(\forall m.\ L \approxeq_{\langle l,m\rangle} L_2)$ *then* $L_1 \equiv^T_l L_2$.

(4) *(transitivity with direct subclosure)*
If $L_1 \equiv^T_0 L_2$ *and* $\langle L_2, T\rangle \sqsupset \langle K_2, U\rangle$ *then there exists* K_1
such that $\langle L_1, T\rangle \sqsupset \langle K_1, U\rangle$ *and* $K_1 \equiv^U_0 K_2$.

(5) *(transitivity with extended reduction for terms)*
If $L_1 \equiv^{T_1}_0 L_2$ *and* $L_2 \vdash T_1 \rightrightarrows_{h,g} T_2$ *then* $L_1 \vdash T_1 \rightrightarrows_{h,g} T_2$.

(6) *(transitivity with extended reduction for environments)*
If $L_1 \equiv^T_l L_2$ *and* $L_2 \vdash\rightrightarrows_{h,g} K_2$ *then there exists* K_1
such that $L_1 \vdash\rightrightarrows_{h,g} K_1$ *and* $K_1 \equiv^T_l K_2$.

(7) *(confluence with extended reduction for terms)*
If $L_1 \vdash T_1 \rightrightarrows_{h,g} T_2$ *then* $L_1 \equiv^{T_1}_0 L_2$ *implies* $L_1 \equiv^{T_2}_0 L_2$.

PROOF. Clause (1) and Clause (2) are proved directly by accessing to lazy equivalence through Rule (8). Clause (3) is proved by induction on its first premise. Clause (4) and Clause (5) are proved by induction on their second premise and by cases on their first premise. Clause (6) follows from Clause (1) and Clause (2) by taking $K_1 = L_1 \Cup^T_l K_2$, which results from Theorem 3.8(3). Here we see the purpose of pointwise union. Clause (7) is proved by induction on its first premise and by cases on its second premise with the help of Clause (3) when Figure 27(bind), Figure 27(β), and Figure 27(θ) are considered. Here we see the purpose of ranged equivalence. ▲

The shape of the second premise in Theorem 3.9(3) is due the implicit instantiation of m with ∞ in Definition 2.37 (lazy equivalence) as noted at the end of Section 2.11. Theorem 3.9(6) and Theorem 3.9(7) (proposition 1000 of Guidi [2014]) are the most interesting properties of lazy equivalence with respect to extended reduction. Their proofs were the most demanding of this set.

THEOREM 3.10 (STRONGLY NORMALIZING ENVIRONMENTS).

(*1*) *(transitivity of strong normalization for environments through extended reduction)*
If $\sim\searrow^{*}_{h,g,l} L$ *and* $L \vdash T_1 \Rrightarrow_{h,g} T_2$*, then* $\searrow^{*\,T_1}_{g,g,l} L$ *implies* $\searrow^{*\,T_2}_{g,g,l} L$.

(*2*) *(construction lemma for variable reference, general form)*
If $l \leq i$ *and* $K_1 \vdash \searrow^{*}_{h,g} W$ *and* $K_1 \vdash\Rrightarrow^{*}_{h,g} K_2$*,*
then $\downarrow_{\langle 0,i\rangle} L_2 = K_2.\delta/\lambda W$ *and* $\searrow^{*\,W}_{h,g,0} K_2$ *imply* $\searrow^{*\,\#i}_{h,g,l} L_2$.

(*3*) *(strong normalization for terms implies strong normalization for environments)*
If $L \vdash \searrow^{*}_{h,g} T$ *then* $\searrow^{*\,T}_{h,g,l} L$ *for every* l.

PROOF. Clause (1) is proved by induction on its second premise and by cases on its third premise. Strongly co-normalizing environments (Definition 2.41) emerge when $T_1 = \#i$ with $i < l$ and Figure 27(δ) is considered. Every construction lemma is needed except for Clause (2), which is proved by induction on $K_1 \vdash \searrow^{*}_{h,g} W$ using Rule (7) and then by induction on $\searrow^{*\,W}_{h,g,0} K_2$ with the help of Clause (1) and of Theorem 3.5(6). Clause (3) is proved by induction on the proper subclosures of $\langle L, T\rangle$ with the help of every construction lemma including Clause (2). ▲

Theorem 3.10(3) is the most relevant property of strongly normalizing environments. Notice that $\searrow^{*\,T}_{h,g,l} L$ is generated by the next rule resembling Figure 40(lsx).

$$\frac{\forall L_2.\ (L_1 \vdash\Rrightarrow^{*}_{h,g} L_2) \Rightarrow (L_1 \not\equiv^{T}_{l} L_2) \Rightarrow (\searrow^{*\,T}_{h,g,l} L_2)}{\searrow^{*\,T}_{h,g,l} L_1}\,\mathrm{lsx} \tag{10}$$

3.7. Results on Very Big Trees

The properties of $qrst$-computations and strong $qrst$-normalization are listed next.

THEOREM 3.11 ($qrst$-COMPUTATIONS).

(*1*) *(decomposition property for* $qrst$*-computation)*
If $\langle L_1, T_1\rangle \succeq_{h,g} \langle L_2, T_2\rangle$ *then there exist* L_0*,* L*, and* T *such that*
$L_1 \vdash T_1 \Rrightarrow^{*}_{h,g} T$ *and* $\langle L_1, T\rangle \sqsupset^{*} \langle L_0, T_2\rangle$ *and* $L_0 \vdash\Rrightarrow^{*}_{h,g} L$ *and* $L \equiv^{T_2}_{0} L_2$.

(*2*) *(formulation of proper* rst*-reduction with* q*-equivalence, left to right)*
If $\langle L_1, T_1\rangle \succ_{h,g} \langle L_2, T_2\rangle$ *then* $\langle L_1, T_1\rangle \succeq_{h,g} \langle L_2, T_2\rangle$ *and* $\langle L_1, T_1\rangle \not\equiv_{0} \langle L_2, T_2\rangle$.

(*3*) *(formulation of proper* rst*-reduction with* q*-equivalence, right to left)*
If $\langle L_1, T_1\rangle \succeq_{h,g} \langle L_2, T_2\rangle$ *and* $\langle L_1, T_1\rangle \not\equiv_{0} \langle L_2, T_2\rangle$ *then* $\langle L_1, T_1\rangle \succ_{h,g} \langle L_2, T_2\rangle$.

(*4*) *(transitivity of proper* rst*-reduction through lazy equivalence)*
If $K_1 \equiv^{T}_{0} K_2$ *and* $\langle K_2, T\rangle \succ_{h,g} \langle L_2, U\rangle$ *then*
there exists L_1 *such that* $\langle K_1, T\rangle \succ_{h,g} \langle L_1, U\rangle$ *and* $L_1 \equiv^{U}_{0} L_2$.

(*5*) *(transitivity of proper* $qrst$*-computation through* $qrst$*-reduction, left case)*
If $\langle L_1, T_1\rangle \succeq_{h,g} \langle L, T\rangle$ *and* $\langle L, T\rangle >\!\equiv_{h,g} \langle L_2, T_2\rangle$ *then* $\langle L_1, T_1\rangle >\!\equiv_{h,g} \langle L_2, T_2\rangle$.

(*6*) *(transitivity of proper* $qrst$*-computation through* $qrst$*-computation, left case)*
If $\langle L_1, T_1\rangle \succeq_{h,g} \langle L, T\rangle$ *and* $\langle L, T\rangle >\!\equiv_{h,g} \langle L_2, T_2\rangle$ *then* $\langle L_1, T_1\rangle >\!\equiv_{h,g} \langle L_2, T_2\rangle$.

PROOF. Clause (1) is proved by induction on its premise rearranging the $qrst$-steps with Theorem 3.5(4), Theorem 3.5(5), Theorem 3.5(6), Theorem 3.9(4), Theorem 3.9(5), and Theorem 3.9(6). Clause (2) is proved by cases on its premise. Clause (3) is proved by cases on its first premise. Clause (4) is proved cases on its second premise with the help of Theorem 3.9(4), Theorem 3.9(5), Theorem 3.9(6), and Theorem 3.9(7). Clause (5) is a corollary of Clause (3) and Clause (4). Clause (6) is proved by induction on its first premise with the help of Clause (5). ▲

Notice that the reverse of Theorem 3.11(1) is straightforward. Also notice that Theorem 3.11(6) implies the transitivity of proper $qrst$-computation. The "right case" of the transitivity, that is: $\langle L_1, T_1\rangle >\!\equiv_{h,g} \langle L, T\rangle \;\&\; \langle L, T\rangle \geq_{h,g} \langle L_2, T_2\rangle \Rightarrow \langle L_1, T_1\rangle >\!\equiv_{h,g} \langle L_2, T_2\rangle$, comes immediately from the transitivity of $qrst$-computation (defined as a transitive closure in Section 2.12). Another important corollary of Theorem 3.11(6) is that the relation $\trianglerighteq_{h,g} \langle L, T\rangle$ is generated by the next rule:

$$\frac{\forall L_2, T_2.\ \langle L_1, T_1\rangle >\!\equiv_{h,g} \langle L_2, T_2\rangle \Rightarrow \trianglerighteq_{h,g} \langle L_2, T_2\rangle}{\trianglerighteq_{h,g} \langle L_1, T_1\rangle}\,\mathrm{fsb} \tag{11}$$

The induction principle for $\trianglerighteq_{h,g} \langle L, T\rangle$ derived from this rule, gives a very strong induction hypothesis that takes advantage of the generality of proper $qrst$-computation (Definition 2.43). We need such a strength to prove the preservation Theorem 3.15.

THEOREM 3.12 (STRONGLY $qrst$-NORMALIZING CLOSURES).

(1) *(strong normalization implies strong $qrst$-normalization, general form)*
If $L_1 \vdash \searrow^*_{h,g} T_1$ *and* $\langle L_1, T_1\rangle \geq_{h,g} \langle L_2, T_2\rangle$ *then* $\trianglerighteq_{h,g} \langle L_2, T_2\rangle$.
(2) *(very big tree theorem)*
If $L \vdash T : A$ *then* $\trianglerighteq_{h,g} \langle L, T\rangle$ *for each* h *and* g_h.

PROOF. Clause (1) is proved by induction on its first premise and then by induction on the proper subclosures of $\langle L_2, T_2\rangle$ by invoking Theorem 3.10(3) and the the reverse of Theorem 3.11(1). Clause (2) is a corollary of Clause (1) and Theorem 3.7(5). ▲

3.8. Results on Stratified Validity

The relevant properties of stratified validity and of its refinement are listed next.

THEOREM 3.13 (STRATIFIED VALIDITY AND ITS REFINEMENT).

(1) *(inclusion of validity)*
If $L \vdash T\ !\ _{h,g}$ *then there exists* A *such that* $L \vdash T : A$.
(2) *(validity implies strong qrst-normalization)*
If $L \vdash T\ !\ _{h,g}$ *then* $\trianglerighteq_{h,g} \langle L, T\rangle$.
(3) *(first inclusion of refinement)*
If $L_1 \doteq\!!_{h,g} L_2$ *then* $L_1 \doteq\!\blacksquare_{h,g} L_2$.
(4) *(second inclusion of refinement)*
If $L_1 \doteq\!!_{h,g} L_2$ *then* $L_1 \doteq\!: L_2$.
(5) *(transitivity of degree-guarded iterated static type assignment through refinement)*
If $n \leq d$ *and* $L_2 \vdash T \blacksquare_{h,g} d$ *then* $L_1 \doteq\!!_{h,g} L_2$ *and* $L_2 \vdash T \bullet^{*(n)}_{h} U_2$
imply $L_1 \vdash T \bullet^{*(n)}_{h} U_1$ *and* $L_1 \vdash U_1 \leftrightarrows^* U_2$ *for some* U_1.
(6) *(transitivity of stratified decomposed computation through refinement)*
If $L_1 \doteq\!!_{h,g} L_2$ *and* $L_2 \vdash T_1 \bullet^* \!\rightrightarrows^{*(n)}_{h,g} T_2$
then $L_1 \vdash T_1 \bullet^* \!\rightrightarrows^{*(n)}_{h,g} T$ *and* $L_1 \vdash T_2 \rightrightarrows^* T$ *for some* T.
(7) *(transitivity of validity through refinement)*
If $L_1 \doteq\!!_{h,g} L_2$ *and* $L_2 \vdash T\ !\ _{h,g}$ *then* $L_1 \vdash T\ !\ _{h,g}$.

$$\begin{array}{lll}
\mathrm{PD}_{h,g}\,\langle L_1, T_1\rangle & \text{is} & (L_1 \vdash T_1 \,!\,_{h,g})\ \&\forall L_2, T_2, d.\\
& & (L_1 \vdash T_1 \blacksquare_{h,g}\ d)\ \&\ (L_1 \vdash T_1 \rightrightarrows T_2)\ \&\ (L_1 \vdash\rightrightarrows L_2) \Rightarrow (L_1 \vdash T_1 \blacksquare_{h,g}\ d)\\
\mathrm{PVR}_{h,g}\,\langle L_1, T_1\rangle & \text{is} & (L_1 \vdash T_1 \,!\,_{h,g})\ \&\forall L_2, T_2.\\
& & (L_1 \vdash T_1 \rightrightarrows T_2)\ \&\ (L_1 \vdash\rightrightarrows L_2) \Rightarrow (L_2 \vdash T_2 \,!\,_{h,g})\\
\mathrm{PVT}_{h,g}\,\langle L_1, T_1\rangle & \text{is} & (L_1 \vdash T_1 \,!\,_{h,g})\ \&\forall U_1, d, n.\\
& & n \le d\ \&\ (L_1 \vdash T_1 \blacksquare_{h,g}\ d)\ \&\ L_1 \vdash T_1 \bullet_h^{*(n)} U_1 \Rightarrow (L_1 \vdash U_1 \,!\,_{h,g})\\
\mathrm{PT}_{h,g}\,\langle L_1, T_1\rangle & \text{is} & (L_1 \vdash T_1 \,!\,_{h,g})\ \&\forall L_2, T_2, U_1, d, n.\\
& & n \le d\ \&\ (L_1 \vdash T_1 \blacksquare_{h,g}\ d)\ \&\ L_1 \vdash T_1 \bullet_h^{*(n)} U_1\ \&\ (L_1 \vdash T_1 \rightrightarrows T_2)\ \&\\
& & (L_1 \vdash\rightrightarrows L_2) \Rightarrow \exists U_2.\ L_2 \vdash T_2 \bullet_h^{*(n)} U_2\ \&\ L_2 \vdash U_1 \leftrightarrows^{*} U_2
\end{array}$$

Fig. 45. Preservation properties.

PROOF. Clause (1) is proved by induction on its premise by invoking Theorem 3.6(5) and Theorem 3.6(7). Here we see that preservation of validity requires preservation of atomic arity. Clause (2) is a corollary of Clause (1) and of Theorem 3.12(2). Clause (3) is proved by induction on its premise. Clause (4) is proved by induction on its premise by invoking Clause (1), Theorem 3.6(3), Theorem 3.6(5), and Theorem 3.6(7) when Figure 23(beta) is considered. Clause (5) is proved by induction on last premise, by cases on its second premise, and then by cases on its third premise. Theorem 3.3(1) is invoked among other propositions when Figure 18(zero) and Figure 18(succ) are considered in the case of Figure 23(beta). Clause (6) is a corollary of Clause (3), Clause (5), Theorem 3.1(2), Theorem 3.2(3), and Theorem 3.3(5). Clause (7) is proved by induction on its second premise and by cases on its first premise, by invoking Clause (6) and Theorem 3.2(1) when Figure 22(appl) and Figure 22(cast) are considered. ▲

We introduce some abbreviations in the style of van Daalen [1994] to state the preservation theorem. With respect to van Daalen [1994], our PVR is connected to his CL, and our PT is connected to his $\mathrm{P}^*\mathrm{T}$.

Definition 3.14 (*preservation properties*). Figure 45 defines four properties of the closure $\langle L_1, T_1\rangle$ with respect to h and g_h. They are: preservation of degree by reduction (PD), preservation of validity by reduction (PVR), preservation of validity by static type (PVT), and preservation of static type by reduction (PT). ▲

THEOREM 3.15 (PRESERVATION PROPERTIES).

(*1*) *(conditional preservation of degree by reduction)*
$(\forall L_1, T_1.\ \langle L, T\rangle >\!\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PD}_{h,g}\,\langle L_1, T_1\rangle)$ *and*
$(\forall L_1, T_1.\ \langle L, T\rangle >\!\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PVR}_{h,g}\,\langle L_1, T_1\rangle)$ *and*
$(\forall L_1, T_1.\ \langle L, T\rangle >\!\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PVT}_{h,g}\,\langle L_1, T_1\rangle)$ *imply* $\mathrm{PD}_{h,g}\,\langle L, T\rangle$.
(*2*) *(conditional preservation of validity by reduction)*
$(\forall L_1, T_1.\ \langle L, T\rangle >\!\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PD}_{h,g}\,\langle L_1, T_1\rangle)$ *and*
$(\forall L_1, T_1.\ \langle L, T\rangle >\!\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PVR}_{h,g}\,\langle L_1, T_1\rangle)$ *and*
$(\forall L_1, T_1.\ \langle L, T\rangle >\!\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PVT}_{h,g}\,\langle L_1, T_1\rangle)$ *and*
$(\forall L_1, T_1.\ \langle L, T\rangle >\!\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PT}_{h,g}\,\langle L_1, T_1\rangle)$ *imply* $\mathrm{PVR}_{h,g}\,\langle L, T\rangle$.
(*3*) *(conditional preservation of validity by static type)*
$(\forall L_1, T_1.\ \langle L, T\rangle >\!\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PD}_{h,g}\,\langle L_1, T_1\rangle)$ *and*
$(\forall L_1, T_1.\ \langle L, T\rangle >\!\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PVR}_{h,g}\,\langle L_1, T_1\rangle)$ *and*
$(\forall L_1, T_1.\ \langle L, T\rangle >\!\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PVT}_{h,g}\,\langle L_1, T_1\rangle)$ *and*
$(\forall L_1, T_1.\ \langle L, T\rangle >\!\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PT}_{h,g}\,\langle L_1, T_1\rangle)$ *imply* $\mathrm{PVT}_{h,g}\,\langle L, T\rangle$.

(4) *(conditional preservation of static type by reduction)*
$(\forall L_1, T_1.\ \langle L, T\rangle >\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PD}_{h,g}\langle L_1, T_1\rangle)$ *and*
$(\forall L_1, T_1.\ \langle L, T\rangle >\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PVR}_{h,g}\langle L_1, T_1\rangle)$ *and*
$(\forall L_1, T_1.\ \langle L, T\rangle >\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PVT}_{h,g}\langle L_1, T_1\rangle)$ *and*
$(\forall L_1, T_1.\ \langle L, T\rangle >\equiv_{h,g} \langle L_1, T_1\rangle \Rightarrow \mathrm{PT}_{h,g}\langle L_1, T_1\rangle)$ *imply* $\mathrm{PT}_{h,g}\langle L, T\rangle$.

(5) *(preservation theorem, general form)*
If $L \vdash T\ !_{h,g}$ *then* $\mathrm{PD}_{h,g}\langle L, T\rangle$ *and* $\mathrm{PVR}_{h,g}\langle L, T\rangle$ *and* $\mathrm{PVT}_{h,g}\langle L, T\rangle$ *and* $\mathrm{PT}_{h,g}\langle L, T\rangle$.

(6) *(preservation of validity by computation)*
If $L \vdash T_1 \rightrightarrows^* T_2$ *then* $L \vdash T_1\ !_{h,g}$ *implies* $L \vdash T_2\ !_{h,g}$.

(7) *(preservation of conversion by static type)*
If $L \vdash T_1\ !_{h,g}$ *and* $L \vdash T_2\ !_{h,g}$ *and* $n \leq d_1$ *and* $n \leq d_2$ *and* $L \vdash T_1 \blacksquare_{h,g} d_1$ *and* $L \vdash T_2 \blacksquare_{h,g} d_2$ *and* $L \vdash T_1 \bullet_h^{*(n)} U_1$ *and* $L \vdash T_2 \bullet_h^{*(n)} U_2$ *then* $L \vdash T_1 \leftrightarrows^* T_2$ *implies* $L \vdash U_1 \leftrightarrows^* U_2$.

PROOF. Clause (1), Clause (2), Clause (3), and Clause (4) are proved by cases on T, and then by cases on the other premises. When Figure 14(β) is considered, Clause (1) invokes Theorem 3.3(5) and Figure 20(beta), Clause (2) invokes Theorem 3.13(7) and Figure 23(beta), while Clause (4) invokes Theorem 3.13(5) and Figure 23(beta). Moreover Clause (2) needs Theorem 3.2(1) in the cases of Figure 14(flat) (already noted by van Daalen [1994]) and of Figure 14(θ), while Clause (4) needs Theorem 3.3(1) in the case of Figure 14(δ). Clause (5) is proved by induction on the proper rst-reducts of $\langle L, T\rangle$ by invoking Clause (1), Clause (2), Clause (3), and Clause (4). The induction is assured by Theorem 3.13(2) and by Rule (11). Clause (6) is proved by induction on its first premise by invoking PVR from Clause (5). Clause (7) is a corollary of Clause (6), Theorem 3.2(3), and Theorem 3.4(1), given PVT and PD from Clause (5). ▲

Theorem 3.15(5) sums up the most significant propositions discussed in this article.

4. CONCLUSION AND FUTURE WORK

We presented in Section 2 a revised version of the formal system $\lambda\delta$ to be termed "$\lambda\delta$ version 2A", and we proved in Section 3 that this calculus enjoys three relevant desired properties: confluence of reduction (Theorem 3.2), strong normalization along $qrst$-computations (Theorem 3.7), and preservation of validity by reduction (Theorem 3.15).

Notably, the matter of this article was entirely developed by the author with the unavoidable help of the proof management system Matita of Asperti et al. [2011], which mechanically validated the resulting formalization of Guidi [2014] in full. The development took 42 months, producing 143 definitions and 1416 propositions. More data is available at $\lambda\delta$ Web site <http://lambdadelta.info/>.

We wish to stress that, to our knowledge, we are presenting as Theorem 3.12(2) the first fully machine-checked proof of the so-called "big tree" theorem [de Vrijer 1994] for a calculus that includes Λ_∞. It is also important to point out that the proof of this theorem is harder in $\lambda\delta$ than in Λ_∞ since the latter system does not have environments.

The long time we needed to take $\lambda\delta$ to this stage, played in favor of presenting the development as is, while the revision of the calculus is far from being complete. In particular the present treatment lacks the type assignment judgment $L \vdash T :_h U$ and its desired properties found in Guidi [2009]. Anyway, it is a design feature of $\lambda\delta$, the fact that a term is typed iff it is valid, so the preservation theorem presented here is the crux of the "subject reduction" property of this judgment.

Moreover, we are interested in relating the present notion of validity, based on an extended (*i.e.*, Λ_∞-like) applicability condition, with the one implied by Guidi [2009], which is based on a restricted (*i.e.*, PTS-like) applicability condition (see Section 2.6). It might happen that every valid closure in the extended sense has an η-equivalent formulation that is valid in the restricted sense. We support this conjecture by noting that a typical case in which we need extended validity, is the next:

$$L.\lambda_z{\star}k.\lambda_y(\lambda{\star}k.{\star}k).\lambda_x y \vdash @z.x\ !\ _{h,g} \tag{12}$$

where named variables improve the readability. If we η-expand y (*i.e.*, the expected type of x) to $\lambda_w{\star}k.@w.y$, restricted validity suffices.

It is important to stress that the above transformation looks like an η-expansion because of the notation, but it might have a different logical meaning. We see such a case considering Landau's "Grundlagen der Analysis" (GdA) formalized in the system $\mathrm{Aut{-}QE}$ [van Benthem Jutting 1994a], where Automath's unified binder `[x:W]` stands either for $\lambda_x W$, or for $\Pi_x W$. The GdA validates just in the extended sense because a situation like (12) occurs in the definition of the constant `ande2"l-r"`, but four formal η-expansions assure its validity in the restricted sense as well. Each one takes an expected type `b`, that is the y of (12), and turns it into `[x:a]<x>b` (`<x>` is our applicator $@x$). We must note that the expected type of `b` is `[x:a]'prop'`, whereas the expected type of `[x:a]<x>b` is `'prop'`. So this expansion is not type-preserving, especially if we accept the statement of Brown [2011] on the GdA that every unified binder of degree one stands for a Π. This means that the expansion is indeed a Π-introduction. Interestingly, Brown [2011] states that formal η-expansions, whose logical meaning should be investigated, solve all incompatibilities preventing the GdA to validate in a PTS.

Theorem 3.13(1) shows that a valid closure can by typed by a simple type. Using $\lambda\delta$ as a logical framework is not a priority, but if we wish to do so (say, for validating the GdA), we need the additional expressive power given by universes (say, $\star$ in the λ-Cube, or `'type'` and `'prop'` in the GdA). However, adding universes to $\lambda\delta$ while preserving its properties is challenging because the naive extension of Λ_∞ with "type inclusion" (the device with which universes are built in the Automath tradition) is not conservative, since either confluence or uniqueness of types is lost.

Other additions to $\lambda\delta$ we shall consider, include: "global" variables referred by level (while the current variables referred by depth would be "local"), term-like environments with projections as we advocated in [Guidi 2009], and metavariables. Furthermore, we are interested in improving multiple relocation (Definition 2.9), which we introduced since the set of the functions $\uparrow^{\langle l,m\rangle}$ is not closed for composition, by considering the functions $\uparrow^{\overline{c}}$ as primitive, and by representing a multiple relocation more conveniently than with a list of pairs. As the reader can see, our long-term aim is to make $\lambda\delta$ a fully fledged and elegant type system suitable for many purposes.

ACKNOWLEDGMENTS

I am grateful to A. Asperti, C. Sacerdoti Coen, and S. Solmi for their constant support and for their valuable advices on the contents of this text. I wish to dedicate this work to A.D. Bonanno and R. Prazzoli for the joyful moments we shared in these years during the development of $\lambda\delta$.

REFERENCES

A. Asperti, L. Padovani, C. Sacerdoti Coen, F. Guidi, and I. Schena. 2003. Mathematical Knowledge Management in HELM. *Annals of Mathematics and Artificial Intelligence* 38, 1 (May 2003), 27–46.

A. Asperti, W. Ricciotti, C. Sacerdoti Coen, and E. Tassi. 2011. The Matita Interactive Theorem Prover. In *Proceedings of the 23rd International Conference on Automated Deduction (CADE-2011) (Lecture Notes in Computer Science)*, N. Bjørner and V. Sofronie-Stokkermans (Eds.), Vol. 6803. Springer, Berlin, Germany, 64–69.

H.P. Barendregt. 1993. Lambda Calculi with Types. *Osborne Handbooks of Logic in Computer Science* 2 (1993), 117–309.

C.E. Brown. 2011. Faithful Reproductions of the Automath Landau Formalization. Typescript note. (2011).

Coq development team. 2002. *The Coq Proof Assistant Reference Manual: release 7.3.1*. INRIA, Orsay, France.

Y. Coscoy. 1996. A Natural Language Explanation for Formal Proofs. In *Int. Conf. on Logical Aspects of Computational Linguistics (LACL) (Lecture Notes in Artificial Intelligence)*, C. Retoré (Ed.), Vol. 1328. Springer, Berlin, Germany, 149–167.

P.L. Curien and H. Herbelin. 2000. The duality of computation. In *5th ACM SIGPLAN int. conf. on Functional programming (ICFP '00)* (revised (sept 2005) ed.) *(ACM SIGPLAN Notices)*, Vol. 35/9. ACM Press, New York, USA, 233–243.

N.G. de Bruijn. 1991. A plea for weaker frameworks. In *Logical Frameworks*. Cambridge University Press, Cambridge, UK, 40–67.

N.G. de Bruijn. 1994a. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. In *Selected Papers on Automath [Nederpelt et al. 1994]*. North-Holland Pub. Co., Amsterdam, The Netherlands, 375–388.

N.G. de Bruijn. 1994b. Some extensions of AUTOMATH: the AUT-4 family. In *Selected Papers on Automath [Nederpelt et al. 1994]*. North-Holland Pub. Co., Amsterdam, The Netherlands, 283–288.

R.C. de Vrijer. 1994. Big trees in a λ-calculus with λ-expressions as types. In *Selected Papers on Automath [Nederpelt et al. 1994]*. North-Holland Pub. Co., Amsterdam, The Netherlands, 469–492.

J.-Y. Girard, P. Taylor, and Y. Lafont. 1989. *Proofs and types*. Cambridge Tracts in Theoretical Computer Science, Vol. 7. Cambridge University Press, Cambridge, UK.

F. Guidi. 2006. lambdadelta_1. Formal specification for the proof assistant Coq 7.3.1. (November 2006). Available at the $\lambda\delta$ Web site: <http://lambdadelta.info/>.

F. Guidi. 2009. The Formal System $\lambda\delta$. *Transactions on Computational Logic* 11, 1 (November 2009), 5:1–5:37.

F. Guidi. 2014. lambdadelta_2. Formal specification for the proof assistant Matita 0.99.2. (October 2014). Available at the $\lambda\delta$ Web site: <http://lambdadelta.info/>.

F. Kamareddine and R.P. Nederpelt. 1996a. Canonical Typing and π-conversion in the Barendregt Cube. *J. Funct. Program.* 6, 2 (1996), 245–267.

F. Kamareddine and R.P. Nederpelt. 1996b. A useful λ-notation. *Theoretical Computer Science* 155, 1 (1996), 85–109.

Z. Luo. 1990. *An Extended Calculus of Constructions*. Ph.D. Dissertation. University of Edinburgh.

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

R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer (Eds.). 1994. *Selected Papers on Automath*. Studies in Logic and the Foundations of Mathematics, Vol. 133. North-Holland Pub. Co., Amsterdam, The Netherlands.

W. W. Tait. 1975. A realizability interpretation of the theory of species. In *Logic Colloquium, Symposium on Logic Held at Boston, 1972-73 (Lecture Notes in Mathematics)*, R. Parikh (Ed.), Vol. 453. Springer, Berlin, Germany, 240–251.

L.S. van Benthem Jutting. 1994a. Checking Landau's Grundlagen in the Automath System. In *Selected Papers on Automath [Nederpelt et al. 1994]*. North-Holland Pub. Co., Amsterdam, The Netherlands, 299–301,701–720,721–732,763–799,805–808.

L.S. van Benthem Jutting. 1994b. The language theory of Λ_∞, a typed λ-calculus where terms are types. In *Selected Papers on Automath [Nederpelt et al. 1994]*. North-Holland Pub. Co., Amsterdam, The Netherlands, 655–683.

D.T. van Daalen. 1994. The language theory of Automath. In *Selected Papers on Automath [Nederpelt et al. 1994]*. North-Holland Pub. Co., Amsterdam, The Netherlands, 163–200 and 303–312 and 493–653.

Received ; revised ; accepted

A. SUMMARY OF NOTATION

The ongoing revision of $\lambda\delta$ includes an update of the notational conventions of Guidi [2009]. This Appendix summarizes the revised notation we introduced in Section 2.

A, B	atomic arity	(Definition 2.30)
$\mathcal{C}$	reducibility candidate	(Definition 2.33)
K, L	environment	(Definition 2.1)
$\mathcal{R}$	generic property on closures	(Definition 2.33)
T, U, V, W	term	(Definition 2.1)
$\overline{V}$	list of arguments	(Definition 2.2)
c	relocation pair	(Definition 2.7)
$\overline{c}$	list of relocation pairs	(Definition 2.9)
d	degree	(Definition 2.19)
g	sort degree parameter	(Definition 2.19)
h	sort hierarchy parameter	(Definition 2.18)
i, j	variable reference depth	(Definition 2.1)
k	sort index	(Definition 2.1)
l	relocation level	(Definition 2.7)
m	relocation depth	(Definition 2.7)
n	number of iterations	(Definition 2.18)
$B \supset A$	functional atomic arity	(Definition 2.30)
$\mathcal{C}_1 \supset \mathcal{C}_2$	function subset	(Definition 2.34)
$K.L$	concatenation	(Definition 2.4)
$L.\delta V$	definition	(Definition 2.1)
$L.\lambda W$	declaration	(Definition 2.1)
$L_1 \approxeq_{\langle l,m \rangle} L_2$	ranged equivalence	(Definition 2.12)
$L_1 \vdash\rightrightarrows L_2$	parallel reduction for environments	(Definition 2.15)
$L_1 \vdash\rightrightarrows_{h,g} L_2$	extended parallel reduction for env.'s	(Definition 2.28)
$L \vdash\rightrightarrows_{h,g} \mathbb{N}(T)$	normal term for extended reduction	(Definition 2.32)
$L_1 \vdash\rightrightarrows^* L_2$	parallel computation for environments	(Definition 2.16)
$L_1 \vdash\rightrightarrows^*_{h,g} L_2$	extended parallel computation for env.'s	(Definition 2.29)
$L \vdash \Searrow^*_{h,g} T$	strongly norm. term for ext. reduction	(Definition 2.32)
$L \vdash T \,!\,_{h,g}$	stratified validity	(Definition 2.22)
$L \vdash T \,!\,_{h,g}{}^{(d)}$	stratified higher validity	(Definition 2.23)
$L \vdash T \,\vdots\, A$	atomic arity assignment	(Definition 2.30)
$L \vdash T_1 \rightarrow T_2$	sequential reduction	(Definition 2.13)
$L \vdash T_1 \rightarrow_{h,g} T_2$	extended sequential reduction	(Definition 2.26)
$L \vdash T_1 \rightrightarrows T_2$	parallel reduction for terms	(Definition 2.14)
$L \vdash T_1 \rightrightarrows_{h,g} T_2$	extended parallel reduction for terms	(Definition 2.27)
$L \vdash T_1 \rightrightarrows^* T_2$	parallel computation for terms	(Definition 2.16)
$L \vdash T_1 \rightrightarrows^*_{h,g} T_2$	extended parallel computation for terms	(Definition 2.29)
$L \vdash T_1 \overset{\leftrightarrow}{\leftrightarrow}{}^* T_2$	parallel conversion for terms	(Definition 2.16)
$L \vdash T \bullet_h^{*(n)} U$	iterated static type assignment	(Definition 2.18)
$L \vdash T_1 \bullet^*\!\rightrightarrows^{*(n)}_{h,g} T_2$	stratified decomposed computation	(Definition 2.21)
$L \vdash T_1 \bullet^*\!\overset{\leftrightarrow}{\leftrightarrow}{}^{*(n_1,n_2)}_{h,g} T_2$	stratified decomposed conversion	(Definition 2.21)
$L \vdash T \blacksquare_{h,g} d$	degree assignment	(Definition 2.19)
$L_1 \dot{\subseteq} L_2$	refinement for preservation of reduction	(Definition 2.17)
$L_1 \dot{\subseteq}_{\mathcal{R}} L_2$	refinement for reducibility	(Definition 2.36)

$L_1 \mathrel{\dot{\subseteq}\!:} L_2$	refinement for preserv. of atomic arity	(Definition 2.31)
$L_1 \mathrel{\dot{\subseteq}\blacksquare_{h,g}} L_2$	refinement for preservation of degree	(Definition 2.20)
$L_1 \mathrel{\dot{\subseteq}!_{h,g}} L_2$	refinement for preserv. of strat. validity	(Definition 2.23)
$L_1 \equiv_l^T L_2$	lazy equivalence for environments	(Definition 2.37)
$L_1 \Cup_l^T L_2 = L$	pointwise union	(Definition 2.39)
$T_1 \mathrel{\bar{\sim}} T_2$	same top structure	(Definition 2.6)
h^n	iterated composition	(Definition 2.18)
$i \in \mathbb{F}_l^* \langle L, U\rangle$	hereditarily free variable	(Definition 2.38)
$\overline{\circ}$	empty list	(Section 2)
$\star$	empty environment	(Definition 2.1)
$\star$	base atomic arity	(Definition 2.30)
$\star k$	sort	(Definition 2.1)
$\#i$	variable reference	(Definition 2.1)
$\lvert L \rvert$	length	(Definition 2.3)
$\delta V.L$	tail definition	(Definition 2.4)
$\delta V.T$	abbreviation	(Definition 2.1)
$\lambda W.L$	tail declaration	(Definition 2.4)
$\lambda W.T$	abstraction	(Definition 2.1)
$@V.T$	application	(Definition 2.1)
$@\overline{V}.T$	multiple application	(Definition 2.2)
©$W.T$	type annotation	(Definition 2.1)
$\uparrow^{\overline{c}} T_1 = T_2$	multiple relocation	(Definition 2.9)
$\uparrow^{\langle l,m\rangle} T_1 = T_2$	relocation	(Definition 2.7)
$\uparrow^{\langle l,m\rangle} \overline{T}_1 = \overline{T}_2$	vector relocation	(Definition 2.8)
$\downarrow_{\overline{c}} L_1 = L_2$	multiple drop	(Definition 2.11)
$\downarrow_{\langle l,m\rangle} L_1 = L_2$	drop	(Definition 2.10)
$\Downarrow_{h,g,l}^{*\ T} L$	strongly norm. env. for ext. reduction	(Definition 2.40)
$\sim\Downarrow_{h,g,l}^{*} L$	strongly co-norm. env. for ext. reduction	(Definition 2.41)
$\trianglerighteq_{h,g} \langle L, T\rangle$	strongly norm. closure for rst-reduction	(Definition 2.44)
$\langle L, T\rangle$	closure	(Section 2.7)
$\langle L, \overline{V}\rangle \in \mathcal{R}$	multiple habitation	(Definition 2.33)
$\langle L_1, T_1\rangle \equiv_l \langle L_2, T_2\rangle$	lazy equivalence for closures	(Definition 2.44)
$\langle L_1, T_1\rangle \sqsupset \langle L_2, T_2\rangle$	direct subclosure	(Definition 2.24)
$\langle L_1, T_1\rangle \sqsupset^? \langle L_2, T_2\rangle$	reflexive direct subclosure	(Definition 2.24)
$\langle L_1, T_1\rangle \sqsupset^* \langle L_2, T_2\rangle$	subclosure	(Definition 2.25)
$\langle L_1, T_1\rangle \succ_{h,g} \langle L_2, T_2\rangle$	proper rst-reduction	(Definition 2.43)
$\langle L_1, T_1\rangle \succeq_{h,g} \langle L_2, T_2\rangle$	$qrst$-reduction	(Definition 2.42)
$\langle L_1, T_1\rangle >\equiv_{h,g} \langle L_2, T_2\rangle$	proper $qrst$-computation	(Definition 2.43)
$\langle L_1, T_1\rangle \geq_{h,g} \langle L_2, T_2\rangle$	$qrst$-computation	(Definition 2.42)
$\langle l, m\rangle$	relocation pair	(Definition 2.7)
$[\![A]\!]_{\mathcal{R}}$	interpretation of the atomic arity	(Definition 2.35)
$\mathbb{S}(T)$	simple (or neutral) term	(Definition 2.5)
$\mathrm{PD}_{h,g} \langle L_1, T_1\rangle$	preservation of degree by reduction	(Definition 3.14)
$\mathrm{PT}_{h,g} \langle L_1, T_1\rangle$	preservation of static type by reduction	(Definition 3.14)
$\mathrm{PVR}_{h,g} \langle L_1, T_1\rangle$	preservation of validity by reduction	(Definition 3.14)
$\mathrm{PVT}_{h,g} \langle L_1, T_1\rangle$	preservation of validity by static type	(Definition 3.14)
$\overline{;}$	list concatenation	(Section 2)
$\forall, \exists, \Rightarrow, \&$	metalinguistic logical constants	(Section 2)
$/$	shared notation	(Definition 2.1)
$\blacktriangle$	end of definition, end of proof	(Section 1)

B. POINTERS TO THE CERTIFIED PROOFS

At the the moment of writing this article, the certified specification of the revised $\lambda\delta$ is available just as a bundle of script files for the latest version of the proof management system Matita. The bundle, `lambdadelta_2.tgz`, is available at the Web site <http://lambdadelta.info/>. For each proposition stated in the article, we give a pointer consisting of a path with three components: a directory inside the directory `basic_2` of the bundle, a file name inside this directory, and a proved statement inside this file. Notice that the notation in the files may differ from Appendix A because of incompatibilities between the characters available for LaTeX and for Matita.

Moreover, the given pointers might be modified in the forthcoming revisions of $\lambda\delta$.

(1) Path for Theorem 3.1(1): `static/lsubr_lsubr/lsubr_trans`
(2) Path for Theorem 3.1(2): `reduction/cpr/lsubr_cpr_trans`
(3) Path for Theorem 3.1(3): `reduction/lpr_lpr/cpr_conf_lpr`
(4) Path for Theorem 3.1(4): `reduction/lpr_lpr/lpr_conf`
(5) Path for Theorem 3.2(1): `computation/cprs_cprs/cprs_conf`
(6) Path for Theorem 3.2(2): `computation/lprs_lprs/lprs_conf`
(7) Path for Theorem 3.2(3): `equivalence/cpcs_cpcs/cpcs_inv_cprs`
(8) Path for Theorem 3.3(1): `unfold/lstas_da/da_lstas`
(9) Path for Theorem 3.3(2): `unfold/lstas_da/lstas_inv_da`
(10) Path for Theorem 3.3(3): `unfold/lstas_da/lstas_inv_da_ge`
(11) Path for Theorem 3.3(4): `static/lsubd/lsubd_fwd_lsubr`
(12) Path for Theorem 3.3(5): `static/lsubd_da/lsubd_da_trans`
(13) Path for Theorem 3.3(6): `static/lsubd_da/lsubd_da_conf`
(14) Path for Theorem 3.3(7): `static/lsubd_lsubd/lsubd_trans`
(15) Path for Theorem 3.4(1): `unfold/lstas_lstas/lstas_mono`
(16) Path for Theorem 3.4(2): `unfold/lstas_da/lstas_inv_refl_pos`
(17) Path for Theorem 3.5(1): `reduction/cpx/lsubr_cpx_trans`
(18) Path for Theorem 3.5(2): `reduction/cpx/cpr_cpx`
(19) Path for Theorem 3.5(3): `reduction/cpx_lift/sta_cpx`
(20) Path for Theorem 3.5(4): `reduction/cpx_lift/fqu_cpx_trans`
(21) Path for Theorem 3.5(5): `reduction/lpx_drop/lpx_fqu_trans`
(22) Path for Theorem 3.5(6): `computation/lpxs_lpxs/lpx_cpx_trans`
(23) Path for Theorem 3.5(7): `computation/cpxs_tsts/cpxs_fwd_beta`
(24) Path for Theorem 3.6(1): `static/lsuba/lsuba_fwd_lsubr`
(25) Path for Theorem 3.6(2): `static/lsuba_aaa/lsuba_aaa_trans`
(26) Path for Theorem 3.6(3): `static/lsuba_aaa/lsuba_aaa_conf`
(27) Path for Theorem 3.6(4): `static/lsuba_lsuba/lsuba_trans`
(28) Path for Theorem 3.6(5): `static/aaa_aaa/aaa_mono`
(29) Path for Theorem 3.6(6): `unfold/lstas_aaa/aaa_lstas`
(30) Path for Theorem 3.6(7): `reduction/lpx_aaa/cpx_lpx_aaa_conf`
(31) Path for Theorem 3.7(1): `computation/csx_tsts_vector/csx_gcr`
(32) Path for Theorem 3.7(2): `computation/gcp_cr/acr_gcr`
(33) Path for Theorem 3.7(3): `computation/gcp_aaa/acr_aaa_csubc_lifts`
(34) Path for Theorem 3.7(4): `computation/gcp_aaa/gcr_aaa`
(35) Path for Theorem 3.7(5): `computation/csx_aaa/aaa_csx`
(36) Path for Theorem 3.7(6): `computation/lsubc/lsubc_fwd_lsubr`
(37) Path for Theorem 3.7(7): `computation/lsubc_lsuba/lsuba_lsubc`
(38) Path for Theorem 3.8(1): `multiple/llor_alt/llor_tail_frees`
(39) Path for Theorem 3.8(2): `multiple/llor_alt/llor_tail_cofrees`
(40) Path for Theorem 3.8(3): `multiple/llor_drop/llor_total`
(41) Path for Theorem 3.9(1): `multiple/llpx_sn_llor/llpx_sn_llor_fwd_sn`

(42) Path for Theorem 3.9(2): `multiple/lleq_llor/llpx_sn_llor_dx`
(43) Path for Theorem 3.9(3): `multiple/lleq_lreq/lleq_lreq_trans`
(44) Path for Theorem 3.9(4): `multiple/lleq_fqus/lleq_fqu_trans`
(45) Path for Theorem 3.9(5): `reduction/cpx_lleq/lleq_cpx_trans`
(46) Path for Theorem 3.9(6): `reduction/lpx_lleq/lleq_lpx_trans`
(47) Path for Theorem 3.9(7): `reduction/cpx_lleq/cpx_lleq_conf_sn`
(48) Path for Theorem 3.10(1): `computation/lcosx_cpx/lsx_cpx_trans_lcosx`
(49) Path for Theorem 3.10(2): `computation/lsx_csx/lsx_lref_be_lpxs`
(50) Path for Theorem 3.10(3): `computation/lsx_csx/csx_lsx`
(51) Path for Theorem 3.11(1): `computation/fpbs_alt/fpbs_inv_alt`
(52) Path for Theorem 3.11(2): `reduction/fpbq_alt/fpb_fpbq_alt`
(53) Path for Theorem 3.11(3): `reduction/fpbq_alt/fpbq_inv_fpb_alt`
(54) Path for Theorem 3.11(4): `reduction/fpb_lleq/lleq_fpb_trans`
(55) Path for Theorem 3.11(5): `computation/fpbg_fpbs/fpbq_fpbg_trans`
(56) Path for Theorem 3.11(6): `computation/fpbg_fpbs/fpbs_fpbg_trans`
(57) Path for Theorem 3.12(1): `computation/fsb_csx/csx_fsb_fpbs`
(58) Path for Theorem 3.12(2): `computation/fsb_aaa/aaa_fsb`
(59) Path for Theorem 3.13(1): `dynamic/snv_aaa/snv_fwd_aaa`
(60) Path for Theorem 3.13(2): `dynamic/snv_fsb/snv_fwd_fsb`
(61) Path for Theorem 3.13(3): `dynamic/lsubsv_lsubd/lsubsv_fwd_lsubd`
(62) Path for Theorem 3.13(4): `dynamic/lsubsv_lsuba/lsubsv_fwd_lsuba`
(63) Path for Theorem 3.13(5): `dynamic/lsubsv_lstas/lsubsv_lstas_trans`
(64) Path for Theorem 3.13(6): `dynamic/lsubsv_scpds/lsubsv_scpds_trans`
(65) Path for Theorem 3.13(7): `dynamic/lsubsv/lsubsv_snv_trans`
(66) Path for Theorem 3.15(1): `dynamic/snv_da_lpr/da_cpr_lpr_aux`
(67) Path for Theorem 3.15(2): `dynamic/snv_lpr/snv_cpr_lpr_aux`
(68) Path for Theorem 3.15(3): `dynamic/snv_lstas/snv_lstas_aux`
(69) Path for Theorem 3.15(4): `dynamic/snv_lstas_lpr/lstas_cpr_lpr_aux`
(70) Path for Theorem 3.15(5): `dynamic/snv_preserve/snv_preserve`
(71) Path for Theorem 3.15(6): `dynamic/snv_preserve/snv_cprs_lpr`
(72) Path for Theorem 3.15(7): `dynamic/snv_preserve/lstas_cpcs_lpr`