A locally nameless representation for a natural semantics for lazy evaluation

Technical Report 01/12
Lidia Sánchez-Gil[1], Mercedes Hidalgo-Herrero[2], and Yolanda Ortega-Mallén[3]

[1] Dpto. Sistemas Informáticos y Computación, Facultad de CC. Matemáticas, Universidad Complutense de Madrid, Spain
[2] Dpto. Didáctica de las Matemáticas, Facultad de Educación, Universidad Complutense de Madrid, Spain
[3] Dpto. Sistemas Informáticos y Computación, Facultad de CC. Matemáticas, Universidad Complutense de Madrid, Spain

Abstract. We propose a locally nameless representation for Launchbury's natural semantics for lazy evaluation. Names are reserved for free variables, while bound variable names are replaced by indices. This avoids the use of α-conversion and facilitates the identification of equivalent values in reduction proofs. We use cofinite quantification to express the semantic rules that introduce fresh names, but we prove that existential rules are admissible too. Moreover, we prove that the choice of names during the evaluation of a term is irrelevant as long as they are fresh enough.

1 Introduction

In the usual representation of the lambda-calculus, i.e., with variable names for free and bound variables, terms are identified up to α-conversion. This notation is suitable for explaining new concepts and for giving examples, while α-substitution together with Barendregt's variable convention [2] are freely used in informal reasoning. But the variable convention may lead to prove false (see [8]), and α-substitution is hard to implement in an automatic proof assistant. Therefore, other representations have been proposed to avoid names and α-conversion. For instance, the de Bruijn notation [5], where variable names are replaced by indices. However, this nameless notation is much less intuitive and quite cumbersome to use, as small modifications of a term may imply multiple shiftings of the indices. A compromise between the named representation and the de Bruijn notation is the locally nameless representation as presented in [4]. In this case, bound variable names are replaced by indices, while free variables keep their names. This mixed notation combines the advantages of both named and nameless representations. On the one hand, α-conversion is no longer needed and variable substitution is easily defined because there is no danger of name capture. On the other hand, terms are still readable and easy to manipulate.

We use a locally nameless representation to express Launchbury's natural semantics for lazy evaluation [6]. Our final purpose is to implement this natural

$$x \in \mathit{Var}$$
$$e \in \mathit{Exp} ::= \lambda x.e \mid (e\ x) \mid x \mid \mathtt{let}\ \{x_i = e_i\}_{i=1}^{n}\ \mathtt{in}\ e.$$

Fig. 1. Restricted *named* syntax of the extended λ-calculus

semantics in some proof assistant like Coq [3], and then to prove formally several properties of the semantics. The reduction rule for local declarations implies the introduction of fresh names. We use neither an existential nor a universal rule for this case. Instead, we opt for a cofinite rule as introduced by Aydemir et al. in [1]. Nevertheless, an *introduction lemma* is stated (and proved) which expresses that an existential rule is admissible too. Our locally nameless semantics is completed with a *regularity lemma* which ensures that every term and heap involved in a reduction proof are well-formed, and with a *renaming lemma* which indicates that the choice of names (free variables) is irrelevant as long as they are fresh enough. We have experienced the advantages of using cofinite rules when demonstrating these results.

In summary, the contributions of this paper are:

1. A locally nameless representation of the λ-calculus extended with recursive local declarations;
2. A locally nameless version of the inductive rules of Launchbury's natural semantics for lazy evaluation;
3. A new version of cofinite rules where the variables quantified in the premises do appear in the conclusion too; and
4. A formal proof of several properties of our reduction system like the regularity, the introduction and the renaming lemmas.

The paper is structured as follows: In Section 2 we present the locally nameless representation of the lambda calculus extended with recursive local declarations. The locally nameless translation of the natural semantics for lazy evaluation given in [6] is described in Section 3, together with the regularity, the introduction and the renaming lemmas. The proofs of these lemmas and other auxiliary results are detailed in the Appendix. In Section 4 we draw conclusions and outline our future work.

2 The locally nameless representation

The language described in [6] is a normalized lambda calculus extended with recursive local declarations. We reproduce the restricted syntax in Figure 1. Normalization is achieved in two steps. First an α-conversion is performed so that all bound variables have distinct names. In a second phase, it is ensured that arguments for applications are restricted to be variables. These static transformations make more explicit the sharing of closures and, thus, simplify the definition of the reduction rules.

We give the corresponding locally nameless representation by following the methodology summarized in [4]:

$$
\begin{array}{lll}
x \in \mathit{Id} & & i, j \in \mathbb{N} \\
v \in \mathit{Var} & ::= & \texttt{bvar}\ i\ j \mid \texttt{fvar}\ x \\
t \in \mathit{LNExp} & ::= & v \mid \texttt{abs}\ t \mid \texttt{app}\ t\ v \mid \texttt{let}\ \{t_i\}_{i=1}^{n}\ \texttt{in}\ t
\end{array}
$$

Fig. 2. Locally nameless syntax

1. Define the syntax of the extended λ-calculus in the locally nameless style.
2. Define the variable opening and variable closing operations.
3. Define the free variables and substitution functions, as well as the local closure predicate.
4. State and prove the properties of the operations on terms that are needed in the development to be carried out.

2.1 Locally nameless syntax

The locally nameless (restricted) syntax is shown in Figure 2. *Var* stands now for the set of *variables*, where it is distinguished between *bound variables* and *free variables*. The calculus includes two variable binders: λ-abstraction and `let`-expression. Since `let` declarations are multibinders, bound variables are represented with two natural numbers: the first number indicates to which binder of the term (either abstraction or `let`) the variable is bound, while the second refers to the position of the variable inside the binder (in the case of an abstraction this second number should be 0). In the following, we will represent a list like $\{t_i\}_{i=1}^{n}$ as $\bar{t}$, with length $|\bar{t}| = n$.

Example 1. Let $e \in \mathit{Exp}$ be the λ-expression given in the named representation

$$e \equiv \lambda z.\texttt{let}\ \{x_1 = \lambda y_1.y_1, x_2 = \lambda y_2.y_2\}\ \texttt{in}\ (z\ x_2).$$

The corresponding locally nameless term $t \in \mathit{LNExp}$ is:

$$t \equiv \texttt{abs}\ (\texttt{let}\ \{\texttt{abs}\ (\texttt{bvar}\ 0\ 0), \texttt{abs}\ (\texttt{bvar}\ 0\ 0)\}\ \texttt{in}\ \texttt{app}\ (\texttt{bvar}\ 1\ 0)\ (\texttt{bvar}\ 0\ 1)).$$

Notice that x_1 and x_2 denote α-equivalent expressions in e. This is more clearly seen in t, where both expressions are represented with syntactically equal terms. □

Application arguments are still restricted to variables, but the first phase of the normalization (described at the beginning of the section) is no longer needed.

2.2 Variable opening and variable closing

Variable opening and *variable closing* are the main operations to manipulate locally nameless terms. We extend the definitions given in [4] to the `let`-expression defined in Figure 2.[4]

[4] Multiple binders are defined in [4]. One corresponds to non recursive local declarations, and the other to mutually recursive expressions. Both constructions are treated as extensions, so that they are not completely developed.

$$
\begin{array}{ll}
\{k \to \overline{x}\}(\texttt{bvar}\ i\ j) & = \begin{cases} \texttt{fvar}\ (\texttt{List.nth}\ j\ \overline{x}) & \text{if } i = k \wedge j < |\overline{x}| \\ \texttt{bvar}\ i\ j & \text{otherwise} \end{cases} \\
\{k \to \overline{x}\}(\texttt{fvar}\ x) & = \texttt{fvar}\ x \\
\{k \to \overline{x}\}(\texttt{abs}\ t) & = \texttt{abs}\ (\{k+1 \to \overline{x}\}\ t) \\
\{k \to \overline{x}\}(\texttt{app}\ t\ v) & = \texttt{app}\ (\{k \to \overline{x}\}\ t)\ (\{k \to \overline{x}\}\ v) \\
\{k \to \overline{x}\}(\texttt{let}\ \overline{t}\ \texttt{in}\ t) & = \texttt{let}\ (\{k+1 \to \overline{x}\}\ \overline{t})\ \texttt{in}\ (\{k+1 \to \overline{x}\}\ t)
\end{array}
$$

where $\{k \to \overline{x}\}\ \overline{t} = \texttt{List.map}\ (\{k \to \overline{x}\}\ \cdot)\ \overline{t}$.

Fig. 3. Variable opening

In order to be able to explore the body of a binder construction (abstraction or `let`), one needs to replace the corresponding bound variables by fresh names. In the case of an abstraction `abs` t the *variable opening operation* provides a (fresh) name to replace in t the bound variables referring to the outermost abstraction. Analogously, the opening of a `let`-term `let` $\overline{t}$ `in` t provides a list of distinct fresh names (as many as local declarations in $\overline{t}$) to replace the bound variables occurring in $\overline{t}$ and in the body t that refer to this particular declaration.

Variable opening is defined in terms of a recursive function $\{k \to \overline{x}\}t$ (Figure 3), where the number k represents the nesting level of the binder of interest, and $\overline{x}$ is a list of pairwise-distinct identifiers in Id. Since the level of the outermost binder is 0, variable opening is defined as:

$$t^{\overline{x}} = \{0 \to \overline{x}\}t.$$

Sometimes we are interested in applying the opening operation to a list of terms: $\overline{t}^{\overline{x}} = \texttt{List.map}\ (\cdot^{\overline{x}})\ \overline{t}$.

The last definition and those in Figure 3 include some operations on lists. We use an ML-like notation. For instance, `List.nth` $j\ \overline{x}$ represents the $(j+1)^{th}$ element of $\overline{x}$,[5] and `List.map` $f\ \overline{t}$ indicates that the function f is applied to every term in the list $\overline{t}$. In the rest of definitions we will use similar list operations.

Inversely to variable opening, there is an operation to transform free names into bound variables. The *variable closing* of a term is represented by ${}^{\backslash\overline{x}}t$, where $\overline{x}$ is the list of names to be bound (recall that all names in $\overline{x}$ are different). The definition of variable closing is based on a recursive function $\{k \leftarrow \overline{x}\}t$ (Figure 4), where k indicates again the level of nesting of binders.Whenever a free variable `fvar` x is encountered, x is looked up in $\overline{x}$. If x occurs in position j, then the free variable is replaced by the bound variable (`bvar` $k\ j$), otherwise it is left unchanged.Variable closing is then defined as follows:

$${}^{\backslash\overline{x}}t = \{0 \leftarrow \overline{x}\}t.$$

Variable closing of a list of terms is: ${}^{\backslash\overline{x}}\overline{t} = \texttt{List.map}\ ({}^{\backslash\overline{x}}\cdot)\ \overline{t}$.

[5] In order to better accommodate to bound variables indices, elements in a list are numbered starting with 0.

$$
\begin{aligned}
\{k \leftarrow \overline{x}\}(\texttt{bvar}\ i\ j) &= \texttt{bvar}\ i\ j \\
\{k \leftarrow \overline{x}\}(\texttt{fvar}\ x) &= \begin{cases} \texttt{bvar}\ k\ j & \text{if } \exists j : 0 \leq j < |\overline{x}|.x = \texttt{List.nth}\ j\ \overline{x} \\ \texttt{fvar}\ x & \text{otherwise} \end{cases} \\
\{k \leftarrow \overline{x}\}(\texttt{abs}\ t) &= \texttt{abs}\ (\{k+1 \leftarrow \overline{x}\}\ t) \\
\{k \leftarrow \overline{x}\}(\texttt{app}\ t\ v) &= \texttt{app}\ (\{k \leftarrow \overline{x}\}\ t)\ (\{k \leftarrow \overline{x}\}\ v) \\
\{k \leftarrow \overline{x}\}(\texttt{let}\ \bar{t}\ \texttt{in}\ t) &= \texttt{let}\ (\{k+1 \leftarrow \overline{x}\}\ \bar{t})\ \texttt{in}\ (\{k+1 \leftarrow \overline{x}\}\ t)
\end{aligned}
$$

where $\{k \leftarrow \overline{x}\}\ \bar{t} = \texttt{List.map}\ (\{k \leftarrow \overline{x}\} \cdot)\ \bar{t}$.

Fig. 4. Variable closing

$$
\text{LC\_VAR}\quad \frac{}{\texttt{lc}\ (\texttt{fvar}\ x)} \qquad\qquad \text{LC\_ABS}\quad \frac{\forall x \notin L \subseteq Id \quad \texttt{lc}\ t^{[x]}}{\texttt{lc}\ (\texttt{abs}\ t)}
$$

$$
\text{LC\_APP}\quad \frac{\texttt{lc}\ t \quad \texttt{lc}\ v}{\texttt{lc}\ (\texttt{app}\ t\ v)} \qquad\qquad \text{LC\_LET}\quad \frac{\forall \overline{x}^{|\bar{t}|} \notin L \subseteq Id \quad \texttt{lc}\ [t : \bar{t}]^{\overline{x}}}{\texttt{lc}\ (\texttt{let}\ \bar{t}\ \texttt{in}\ t)}
$$

$$
\text{LC\_LIST}\quad \frac{\texttt{List.forall}\ (\texttt{lc}\ \cdot)\ \bar{t}}{\texttt{lc}\ \bar{t}}
$$

Fig. 5. Local closure

2.3 Local closure, free variables and substitution

The locally nameless syntax in Figure 2 allows to build terms that have no corresponding expression in *LNExp* (Figure 1). For instance, the term `abs` (`bvar` 1 5) is an improper syntactic object, since index 1 does not refer to a binder in the term. The well-formed terms, i.e., those that correspond to expressions in *LNExp*, are called *locally closed*.

To determine if a term is locally closed one should check that any bound variable in the term has valid indices, i.e., that they refer to binders in the term. However, this checking is not straightforward, and an easier method is to open with fresh names every abstraction and `let`-expression in the term to be checked, and prove that no bound variable is ever reached. This checking is implemented with the *local closure* predicate `lc` t given in Figure 5.

Observe that cofinite quantification rules [1] are used for the binders, i.e., abstraction and `let`. Cofinite quantification is an elegant alternative to exist-fresh conditions and provides stronger induction and inversion principles. Proofs are simplified, because it is not required to define exactly the set of fresh names (several examples of this are given in [4]). The rule LC-ABS establishes that an abstraction is locally closed if there exists a finite set of names L such that, for any name x not in L, the term $t^{[x]}$ is locally closed. Similarly, the rule LC-LET indicates that a `let`-expression is locally closed if there exists a finite set of names L such that, for any list of distinct names $\overline{x}$ not in L and of length $|\bar{t}|$ ($\overline{x}^{|\bar{t}|} \notin L$), the opening of each term in the list of local declarations, $\bar{t}^{\overline{x}}$, and of the term affected by these declarations, $t^{\overline{x}}$, is locally closed. We use the notation $[t : \bar{t}]$ to represent the list with head t and tail $\bar{t}$. The empty list is represented as

$$\text{LCK-BVAR}\quad \frac{i < k \wedge j < \texttt{List.nth}\ i\ \overline{n}}{\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{bvar}\ i\ j)} \qquad \text{LCK-APP}\quad \frac{\texttt{lc\_at}\ k\ \overline{n}\ t \qquad \texttt{lc\_at}\ k\ \overline{n}\ v}{\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{app}\ t\ v)}$$

$$\text{LCK-FVAR}\quad \frac{}{\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{fvar}\ x)} \qquad \text{LCK-LET}\quad \frac{\texttt{lc\_at}\ (k+1)\ [|\overline{t}| : \overline{n}]\ [t : \overline{t}]}{\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{let}\ \overline{t}\ \texttt{in}\ t)}$$

$$\text{LCK-ABS}\quad \frac{\texttt{lc\_at}\ (k+1)\ [1 : \overline{n}]\ t}{\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{abs}\ t)} \qquad \text{LCK-LIST}\quad \frac{\texttt{List.forall}\ (\texttt{lc\_at}\ k\ \overline{n}\ \cdot)\ \overline{t}}{\texttt{lc\_at}\ k\ \overline{n}\ \overline{t}}$$

Fig. 6. Closed at level k

$[\,]$, a unitary list as $[t]$, and $[\overline{t} : t]$ stands for $\overline{t} +\!\!+ [t]$, where $+\!\!+$ is the concatenation of lists.

Coming back to the first approach to local closure, i.e., checking that indices in bound variables are valid, a new predicate is defined: t *is closed at level* k, written $\texttt{lc\_at}\ k\ \overline{n}\ t$ (Figure 6), where k indicates the current depth, that is, how many binders have been passed by. As binders can be either abstractions or local declarations, we need to keep the size of each binder (1 in case of an abstraction, n for a `let`-expression with n local declarations). These sizes are collected in the list $\overline{n}$, thus $|\overline{n}|$ should be at least k. A bound variable $\texttt{bvar}\ i\ j$ is closed at level k if i is smaller than k and j is smaller than $\texttt{List.nth}\ i\ \overline{n}$. The list $\overline{n}$ is new with respect to [4] because there the predicate `lc_at` is not defined for multiple binders.

We can define an order between lists of natural numbers as follows:

$$[\,] \geq [\,] \qquad\qquad m \geq n \wedge \overline{m} \geq \overline{n} \Rightarrow [m : \overline{m}] \geq [n : \overline{n}]$$

If a term t is locally closed at level k for a given list of numbers $\overline{n}$, then it is also locally closed at level k for any list of numbers greater than $\overline{n}$.

Lemma 1.
LC_AT_M_FROM_N $\quad \texttt{lc\_at}\ k\ \overline{n}\ t \Rightarrow \forall \overline{m} \geq \overline{n}\,.\,\texttt{lc\_at}\ k\ \overline{m}\ t$

The two approaches are equivalent, so that it can be proved that a term is locally closed if and only if it is closed at level 0.

Lemma 2.
LC_IIF_LC_AT $\quad \texttt{lc}\ t \Leftrightarrow \texttt{lc\_at}\ 0\ [\,]\ t$

Computing the *free variables* of a term t is very easy in the locally nameless representation, since bound and free variables are syntactically different. The set of free variables of $t \in \mathit{LNExp}$ is denoted as $\texttt{fv}(t)$, and it is defined in Figure 7.

A name x is said to be *fresh for a term* t, written $\texttt{fresh}\ x\ \texttt{in}\ t$, if x does not belong to the set of free variables of t:

$$\frac{x \notin \texttt{fv}(t)}{\texttt{fresh}\ x\ \texttt{in}\ t}$$

$$\begin{array}{llll} \mathtt{fv}(\mathtt{bvar}\ i\ j) & = \emptyset & \mathtt{fv}(\mathtt{fvar}\ x) & = \{x\} \\ \mathtt{fv}(\mathtt{app}\ t\ v) & = \mathtt{fv}(t) \cup \mathtt{fv}(v) & \mathtt{fv}(\mathtt{abs}\ t) & = \mathtt{fv}(t) \\ \mathtt{fv}(\mathtt{let}\ \overline{t}\ \mathtt{in}\ t) & = \mathtt{fv}(\overline{t}) \cup \mathtt{fv}(t) & & \end{array}$$

where $\mathtt{fv}(\overline{t}) = \mathtt{List.foldright}\ (\cdot\ \cup\ \cdot)\ \emptyset\ (\mathtt{List.map}\ \mathtt{fv}\ \overline{t})$.

Fig. 7. Free variables

$$\begin{array}{llll} (\mathtt{bvar}\ i\ j)[z/y] & = \mathtt{bvar}\ i\ j & (\mathtt{fvar}\ x)[z/y] & = \begin{cases} \mathtt{fvar}\ z & \text{if } x = y \\ \mathtt{fvar}\ x & \text{if } x \neq y \end{cases} \\ (\mathtt{abs}\ t)[z/y] & = \mathtt{abs}\ t[z/y] & (\mathtt{app}\ t\ v)[z/y] & = \mathtt{app}\ t[z/y]\ v[z/y] \\ (\mathtt{let}\ \overline{t}\ \mathtt{in}\ t)[z/y] & = \mathtt{let}\ \overline{t}[z/y]\ \mathtt{in}\ t[z/y] & & \end{array}$$

where $\overline{t}[z/y] = \mathtt{List.map}\ ([z/y]\cdot)\ \overline{t}$.

Fig. 8. Substitution

This definition can be easily extended to a list of distinct names $\overline{x}$:

$$\frac{\overline{x} \notin \mathtt{fv}(t)}{\mathtt{fresh}\ \overline{x}\ \mathtt{in}\ t}$$

A term t is *closed* if it has no free variables at all:

$$\frac{\mathtt{fv}(t) = \emptyset}{\mathtt{closed}\ t}$$

Substitution replaces a variable name by another name in a term. So that for $t \in \mathit{LNExp}$ and $z, y \in \mathit{Id}$, $t[z/y]$ is the term where z substitutes any occurrence of y in t (see Figure 8).

Under some conditions variable closing and variable opening are inverse operations. More precisely, opening a term with fresh names and closing it with the same names, produces the original term. Symmetrically, closing a locally closed term t and then opening it with the same names gives back t.

Lemma 3.

CLOSE_OPEN_VAR	$\mathtt{fresh}\ \overline{x}\ \mathtt{in}\ t \Rightarrow {}^{\backslash\overline{x}}(t^{\overline{x}}) = t$
OPEN_CLOSE_VAR	$\mathtt{lc}\ t \Rightarrow ({}^{\backslash\overline{x}}t)^{\overline{x}} = t$

3 Natural semantics for lazy λ-calculus

The natural semantics defined by Launchbury [6] follows a lazy strategy. Judgements are of the form $\Gamma : e \Downarrow \Delta : w$, that is, the expression $e \in \mathit{Exp}$ in the context of the heap Γ reduces to the value w in the context of the (modified) heap Δ. *Values* ($w \in \mathit{Val}$) are expressions in weak-head-normal-form (*whnf*). *Heaps* are partial functions from variables into expressions. Each pair (variable, expression) is called a *binding*, and it is represented by $x \mapsto e$. During evaluation, new bindings may be added to the heap, and bindings may be updated to their

$$\textsc{Lam} \quad \Gamma : \lambda x.e \Downarrow \Gamma : \lambda x.e \qquad \textsc{App} \quad \frac{\Gamma : e \Downarrow \Theta : \lambda y.e' \quad \Theta : e'[x/y] \Downarrow \Delta : w}{\Gamma : (e\ x) \Downarrow \Delta : w}$$

$$\textsc{Var} \quad \frac{\Gamma : e \Downarrow \Delta : w}{(\Gamma, x \mapsto e) : x \Downarrow (\Delta, x \mapsto w) : \hat{w}} \qquad \textsc{Let} \quad \frac{(\Gamma, \{x_i \mapsto e_i\}_{i=1}^n) : e \Downarrow \Delta : w}{\Gamma : \mathtt{let}\ \{x_i = e_i\}_{i=1}^n\ \mathtt{in}\ e \Downarrow \Delta : w}$$

Fig. 9. Natural semantics

corresponding computed values. The rules of this natural semantics are shown in Figure 9. The normalization of the λ-calculus, that has been mentioned in Section 2, simplifies the definition of the operational rules, although a renaming is still needed ($\hat{w}$ in Var) to avoid name clashing. This renaming is justified by the Barendregt's variable convention [2].

3.1 Locally nameless heaps

Before translating the semantic rules in Figure 9 to the locally nameless representation defined in Section 2, we have to establish how *bindings* and *heaps* are represented in this notation.

Recall that bindings associate expressions to free variables, therefore bindings are now pairs $(\mathtt{fvar}\ x, t)$ with $x \in \mathit{Id}$ and $t \in \mathit{LNExp}$. To simplify, we will just write $x \mapsto t$. In the following, we will represent a heap $\{x_i \mapsto t_i\}_{i=1}^n$ as $(\overline{x} \mapsto \overline{t})$, with $|\overline{x}| = |\overline{t}| = n$. The set of the locally-nameless-heaps is denoted as *LNHeap*.

The *domain* of a heap Γ, written $\mathtt{dom}(\Gamma)$, collects the set of names that are bound in the heap.

$$\mathtt{dom}(\emptyset) = \emptyset \qquad \mathtt{dom}(\Gamma, x \mapsto t) = \mathtt{dom}(\Gamma) \cup \{x\}$$

In a well-formed heap names are defined at most once and terms are locally closed. The predicate ok expresses that a heap is well-formed:

$$\text{OK-EMPTY} \ \frac{}{\mathtt{ok}\ \emptyset} \qquad \text{OK-CONS} \ \frac{\mathtt{ok}\ \Gamma \quad x \notin \mathtt{dom}(\Gamma) \quad \mathtt{lc}\ t}{\mathtt{ok}\ (\Gamma, x \mapsto t)}$$

A similar (but related with normalization) predicate *distinctly named* is defined in [6] for heap/term pairs.

The function names returns the set of names that appear in a heap, i.e., the names occurring in the domain or in the right side terms:

$$\mathtt{names}(\emptyset) = \emptyset \qquad \mathtt{names}(\Gamma, x \mapsto t) = \mathtt{names}(\Gamma) \cup \{x\} \cup \mathtt{fv}(t)$$

This definition can be extended to the context of a heap/term pair:

$$\mathtt{names}(\Gamma : t) = \mathtt{names}(\Gamma) \cup \mathtt{fv}(t)$$

We use it to define the freshness predicate of a list of names in a heap/term pair:

$$\text{LNLam}\quad \frac{\{\texttt{ok}\ \Gamma\} \qquad \{\texttt{lc}\ (\texttt{abs}\ t)\}}{\Gamma : \texttt{abs}\ t \Downarrow \Gamma : \texttt{abs}\ t}$$

$$\text{LNVar}\quad \frac{\Gamma : t \Downarrow \Delta : w \qquad \{x \notin \texttt{dom}(\Gamma) \cup \texttt{dom}(\Delta)\}}{(\Gamma, x \mapsto t) : (\texttt{fvar}\ x) \Downarrow (\Delta, x \mapsto w) : w}$$

$$\text{LNApp}\quad \frac{\Gamma : t \Downarrow \Theta : \texttt{abs}\ u \qquad \Theta : u^{[x]} \Downarrow \Delta : w \qquad \{x \notin \texttt{dom}(\Gamma) \Rightarrow x \notin \texttt{dom}(\Delta)\}}{\Gamma : \texttt{app}\ t\ (\texttt{fvar}\ x) \Downarrow \Delta : w}$$

$$\text{LNLet}\quad \frac{\forall \overline{x}^{|\overline{t}|} \notin L \subseteq Id\ \ (\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}} \qquad \{\overline{y}^{|\overline{t}|} \notin L \subseteq Id\}}{\Gamma : \texttt{let}\ \overline{t}\ \texttt{in}\ t \Downarrow (\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) : w^{\overline{y}}}$$

Fig. 10. Locally nameless natural semantics

$$\frac{\overline{x} \notin \texttt{names}(\Gamma : t)}{\texttt{fresh}\ \overline{x}\ \texttt{in}\ (\Gamma : t)}$$

Substitution of variable names is extended to heaps as follows:

$$\emptyset[z/y] = \emptyset \qquad (\Gamma, x \mapsto t)[z/y] = (\Gamma[z/y], x[z/y] \mapsto t[z/y])$$
$$\textit{where}\ x[z/y] = \begin{cases} z & \text{if } x = y \\ x & \text{otherwise} \end{cases}$$

The following property is verified:

Lemma 4.
OK_SUBS_OK $\quad \texttt{ok}\ \Gamma \wedge y \notin \texttt{dom}(\Gamma) \Rightarrow \texttt{ok}\ \Gamma[y/x]$

3.2 Locally nameless semantics

Once the locally nameless syntax and the corresponding operations, functions and predicates have been defined, three steps are sufficient to translate an inductive definition on λ-terms from the named representation into the locally nameless notation (as it is explained in [4]):

1. Replace the named binders, i.e., abstractions and let-constructions, with nameless binders by opening the bodies.
2. Cofinitely quantify the names introduced for variable opening.
3. Add premises to inductive rules in order to ensure that inductive judgements are restricted to locally closed terms.

We apply these steps to the inductive rules for the lazy natural semantics given in Figure 9. These rules produce judgements involving λ-terms as well as heaps. Hence, we also add premises that ensure that inductive judgements are restricted to well-formed heaps. The rules using the locally nameless representation are shown in Figure 10. For clarity, in the rules we put in braces the side-conditions to distinguish them better from the judgements.

The main difference with the rules in Figure 9 is the rule LNLet. To evaluate $\texttt{let}\ \bar{t}\ \texttt{in}\ t$ the local terms $\bar{t}$ have to be introduced in the heap, so that the body t is evaluated in this new context. To this purpose fresh names $\overline{x}$ are needed to open the local terms and the body. The evaluation of $t^{\overline{x}}$ produces a final heap and a value. Both are dependent on the names chosen for the local variables. The domain of the final heap consists of the local names $\overline{x}$ and the rest of names, say $\overline{z}$. The rule LNLet is cofinite quantified. As it is explained in [4], the advantage of the cofinite rules over existential and universal ones is that the freshness side-conditions are not explicit. The freshness condition for $\overline{x}$ is *hidden* in the finite set L, which includes the names that should be avoided during the reduction. The novelty of our cofinite rule, compared with the ones appearing in [1] and [4] (that are similar to the cofinite rules for the predicate $\texttt{lc}$ in Figure 5), is that the names introduced in the (infinite) premises do appear in the conclusion too. Therefore, in the conclusion of the rule LNLet we can replace the names $\overline{x}$ by any list $\overline{y}$ not in L.

The problem with explicit freshness conditions is that they are associated just to rule instances, while they should apply to the whole reduction proof. Take for instance the rule LNVar. In the premise the binding $x \mapsto t$ does no longer belong to the heap. Therefore, a valid reduction for this premise may chose x as fresh. We avoid this situation by requiring that x is undefined in the final heap too.[6] By contrast to the rule Var in Figure 9, no renaming of the final value, that is w, is needed.

The side-condition of rule LNApp deserves an explanation too. Let us suppose that x is undefined in the initial heap Γ. We must avoid that x is chosen as a fresh name during the evaluation of t. For this reason we require that x is defined in the final heap Δ only if x was already defined in Γ. Notice how the body of the abstraction, that is u, is open with the name x. This is equivalent to the substitution of x for y in the body of the abstraction $\lambda y.e'$ (see rule App in Figure 9).

A *regularity* lemma ensures that the judgements produced by this reduction system involve only well-formed heaps and locally closed terms.

Lemma 5.
REGULARITY $\quad \Gamma : t \Downarrow \Delta : w \Rightarrow \texttt{ok}\ \Gamma \wedge \texttt{lc}\ t \wedge \texttt{ok}\ \Delta \wedge \texttt{lc}\ w.$

Similarly, Theorem 1 in [6] ensures that the property of being *distinctly named* is preserved by the rules in Figure 9.

The next lemma states that names defined in a context heap remain defined after the evaluation of any term in that context.

Lemma 6.
DEF_NOT_LOST $\quad \Gamma : t \Downarrow \Delta : w \Rightarrow \texttt{dom}(\Gamma) \subseteq \texttt{dom}(\Delta).$

[6] An alternative is to decorate judgements with a set collecting the names that have been taken out of the heap during a reduction proof, and starting with the empty set. This approach has been adopted by Sestoft in [7].

Moreover, fresh names are only introduced by the rule LNLet and, consequently, they are bound in the final heap/value pair. Therefore, any undefined free variable appearing in the final heap/value pair must occur in the initial heap/term pair too.

Lemma 7.
ADD_VARS $\Gamma : t \Downarrow \Delta : w$
$\Rightarrow (x \in \texttt{names}(\Delta : w) \Rightarrow (x \in \texttt{dom}(\Delta) \vee x \in \texttt{names}(\Gamma : t)))$.

A *renaming* lemma ensures that the evaluation of a term is independent of the fresh names chosen in the reduction process. Moreover, any name in the context can be replaced by a fresh one without changing the meaning of the terms evaluated in that context. In fact, reduction proofs for heap/term pairs are unique up to α-conversion of the names defined in the context heap.

Lemma 8.
RENAMING $\Gamma : t \Downarrow \Delta : w \wedge \texttt{fresh}\ y\ \texttt{in}\ (\Gamma : t) \wedge \texttt{fresh}\ y\ \texttt{in}\ (\Delta : w)$
$\Rightarrow \Gamma[y/x] : t[y/x] \Downarrow \Delta[y/x] : w[y/x]$.

In addition, the renaming lemma permits to prove an *introduction* lemma for the cofinite rule LNLet which establishes that the corresponding existential rule is admissible too.

Lemma 9.
LET_INTRO $(\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}} \wedge \texttt{fresh}\ \overline{x}\ \texttt{in}\ (\Gamma : \texttt{let}\ \overline{t}\ \texttt{in}\ t)$
$\Rightarrow \Gamma : \texttt{let}\ \overline{t}\ \texttt{in}\ t \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}}$.

This result, together with the renaming lemma, justifies that our rule LNLet is equivalent to Launchbury's rule Let used with normalized terms.

4 Conclusions and future work

In the present work we have used a locally nameless representation not only for the pure λ-calculus [4] but also for its extension with mutually recursive local declarations. This notation avoids name clashing between bound and free variables. Afterwards, we have used this representation for redefining Launchbury's natural semantics for lazy evaluation [6]. To this purpose we have adapted the definition of context heaps to the locally nameless notation. A heap may be seen as a multiple binder. Actually, the names defined (bound) in a heap can be replaced by other names, provided that terms keep their meaning in the context represented by the heap. Our renaming lemma ensures that whenever a heap is renamed with fresh names, reduction proofs are preserved.

Launchbury assumes Barendregt's variable convention [2] in [6] when defining his operational semantics only for normalized λ-terms. In order to the avoid this problematic [8] variable convention, we have used cofinite quantification in our locally nameless reduction rules. Freshness conditions are usually considered in each rule individually. Nevertheless, this technique produces name clashing when considering whole reduction proofs. A solution might be to decorate each rule with the set of forbidden names and indicate how to modify this set during the reduction process. However, this could be too restrictive in many occasions. Moreover, existential rules are not easy to deal with because each reduction is obtained just for one specific list of names. If any of the names in this list causes a name clashing with other reduction proofs, then it is cumbersome to demonstrate that an alternative reduction for a fresh list does exist. Cofinite quantification has allowed us to solve this problem because in a single step reductions are guaranteed for an infinite number of lists of names. Moreover, our introduction lemma guarantees that a more conventional exists-fresh rule is correct in our reduction system too.

The cofinite quantification that we have used in our semantic rules is more complex than those in [1] and [4]. Our cofinite rule LNLet in Figure 10 introduces quantified variables in the conclusion as well, as the latter depends on the chosen names.

Our future tasks include the implementation in the proof assistant Coq [3] of the natural semantics redefined in this paper. The final aim is to prove automatically the equivalence of the natural semantics with the alternative version given also in [6]. This alternative version differs from the original one in the introduction of indirections during β-reduction and the elimination of updates. At present we are working on the definition (using the locally nameless representation) of two intermediate semantics, one introducing indirections and the other without updates. Then, we will establish equivalence relations between heaps obtained by each semantics, which allow us to prove the equivalence of the original natural semantics and the alternative semantics through the intermediate semantics.

5 Acknowledgments

This work is partially supported by the projects: TIN2009-14599-C03-01 and S2009/TIC-1465.

References

1. B. E. Aydemir, A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich. Engineering formal metatheory. In *ACM Symposium on Principles of Programming Languages, POPL'08*, pages 3–15. ACM Press, 2008.
2. H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.
3. Y. Bertot. Coq in a hurry. *CoRR*, abs/cs/0603118, 2006.
4. A. Charguéraud. The locally nameless representation. *Journal of Automated Reasoning*, pages 1–46, 2011.
5. N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 75(5):381–392, 1972.

6. J. Launchbury. A natural semantics for lazy evaluation. In *ACM Symposium on Principles of Programming Languages, POPL'93*, pages 144–154. ACM Press, 1993.
7. P. Sestoft. Deriving a lazy abstract machine. *Journal of Functional Programming*, 7(3):231–264, 1997.
8. C. Urban, S. Berghofer, and M. Norrish. Barendregt's variable convention in rule inductions. In *Proceedings of the 21st International Conference on Automated Deduction: Automated Deduction*, pages 35–50. LNCS 4603, Springer-Verlag, 2007.

6 Appendix

6.1 Proof of Lemma 1: LC_AT_M_FROM_N

Lemma 1 :
LC_AT_M_FROM_N $\quad \texttt{lc\_at}\ k\ \overline{n}\ t \Rightarrow \forall \overline{m} \geq \overline{n}\,.\,\texttt{lc\_at}\ k\ \overline{m}\ t$

Proof. The proof is done by structural induction on t.

- $t \equiv \texttt{bvar}\ i\ j$.
  $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{bvar}\ i\ j)$, then $i < k\ \wedge\ j < \texttt{List.nth}\ i\ \overline{n}$.
  If $\overline{m} \geq \overline{n}$, then $\texttt{List.nth}\ i\ \overline{m} \geq \texttt{List.nth}\ i\ \overline{n}$.
  Consequently $i < k\ \wedge\ j < \texttt{List.nth}\ i\ \overline{m}$.
  Applying rule LCK-BVAR, $\texttt{lc\_at}\ k\ \overline{m}\ (\texttt{bvar}\ i\ j)$.

- $t \equiv \texttt{fvar}\ x$.
  Trivial.

- $t \equiv \texttt{abs}\ t'$.
  $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{abs}\ t')$, then $\texttt{lc\_at}\ (k+1)\ [1:\overline{n}]\ t'$.
  Since $\overline{m} \geq \overline{n}$, then $[1:\overline{m}] \geq [1:\overline{n}]$.
  By induction hypothesis, $\texttt{lc\_at}\ (k+1)\ [1:\overline{m}]\ t'$.
  Applying rule LCK-ABS, $\texttt{lc\_at}\ k\ \overline{m}\ (\texttt{abs}\ t)$.

- $t \equiv \texttt{app}\ t'\ v$.
  $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{app}\ t'\ v)$, then $\texttt{lc\_at}\ k\ \overline{n}\ t' \wedge \texttt{lc\_at}\ k\ \overline{n}\ v$.
  Since $\overline{m} \geq \overline{n}$,
  by induction hypothesis, $\texttt{lc\_at}\ k\ \overline{m}\ t' \wedge \texttt{lc\_at}\ k\ \overline{m}\ v$.
  Applying rule LCK-APP, $\texttt{lc\_at}\ k\ \overline{m}\ (\texttt{app}\ t'\ v)$.

- $t \equiv \texttt{let}\ \bar{t}\ \texttt{in}\ t'$.
  $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{let}\ \bar{t}\ \texttt{in}\ t')$, then $\texttt{lc\_at}\ (k+1)\ [|\bar{t}|:\overline{n}]\ \bar{t} \wedge \texttt{lc\_at}\ (k+1)\ [|\bar{t}|:\overline{n}]\ t'$.
  Since $\overline{m} \geq \overline{n}$, then $[|\bar{t}|:\overline{m}] \geq [|\bar{t}|:\overline{n}]$.
  By induction hypothesis, $\texttt{lc\_at}\ (k+1)\ [|\bar{t}|:\overline{m}]\ \bar{t} \wedge \texttt{lc\_at}\ (k+1)\ [|\bar{t}|:\overline{m}]\ t'$.
  Applying rule LCK-LET, $\texttt{lc\_at}\ k\ \overline{m}\ (\texttt{let}\ \bar{t}\ \texttt{in}\ t')$.

□

6.2 Proof of Lemma 2: LC_IIF_LC_AT

To prove Lemma 2, we have to prove two auxiliary results: Lemmas 10 and 11.

If a term t opened with names $\overline{x}$ at level k is locally closed at level k with $\overline{n}$, then the term t is also locally closed at level $k+1$ with $[\overline{n} : |\overline{x}|]$.

Lemma 10.
LC_AT_K+1_FROM_K $\quad k = |\overline{n}| \wedge \texttt{lc\_at}\, k\, \overline{n}\, (\{k \to \overline{x}\} t) \Rightarrow \texttt{lc\_at}\, (k+1)\, [\overline{n} : |\overline{x}|]\, t$

Proof. The proof is done by induction on the structure of t.

- $t \equiv \texttt{bvar}\ i\ j$.
  $\texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}(\texttt{bvar}\ i\ j))$.
  - $i = k \wedge j < |\overline{x}|$
    By hypothesis, $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{fvar}\ (\texttt{List.nth}\ j\ \overline{x}))$
    Thus,
    $i = k < k+1 \wedge j < |\overline{x}| \stackrel{k=|\overline{n}|}{=} \texttt{List.nth}\ k\ [\overline{n} : |\overline{x}|] = \texttt{List.nth}\ i\ [\overline{n} : |\overline{x}|]$.
  - otherwise
    By hypothesis, $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{bvar}\ i\ j)$, then $i < k \wedge j < \texttt{List.nth}\ i\ \overline{n}$.
    Thus, $i < k < k+1 \wedge j < \texttt{List.nth}\ i\ \overline{n} = \texttt{List.nth}\ i\ [\overline{n} : |\overline{x}|]$.

  In both cases, by rule LCK-BVAR, $\texttt{lc\_at}\ (k+1)\ [\overline{n} : |\overline{x}|]\ (\texttt{bvar}\ i\ j)$.

- $t \equiv \texttt{fvar}\ x$.
  Trivial.

- $t \equiv \texttt{abs}\ t'$.
  Since $\texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}(\texttt{abs}\ t'))$, $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{abs}\ (\{k+1 \to \overline{x}\}t'))$.
  Thus, $\texttt{lc\_at}\ (k+1)\ [1 : \overline{n}]\ (\{k+1 \to \overline{x}\}t')$.
  By induction hypothesis, $\texttt{lc\_at}\ (k+2)\ [1 : \overline{n} : |\overline{x}|]\ t'$.
  Applying rule LCK-ABS, $\texttt{lc\_at}\ (k+1)\ [\overline{n} : |\overline{x}|]\ (\texttt{abs}\ t')$.

- $t \equiv \texttt{app}\ t'\ v$.
  Since $\texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}(\texttt{app}\ t'\ v))$,
  $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{app}\ (\{k \to \overline{x}\}t')\ (\{k \to \overline{x}\}v))$.
  Thus, $\texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}t') \wedge \texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}v)$.
  By induction hypothesis, $\texttt{lc\_at}\ (k+1)\ [\overline{n} : |\overline{x}|]\ t' \wedge \texttt{lc\_at}\ (k+1)\ [\overline{n} : |\overline{x}|]\ v$.
  Applying rule LCK-APP, $\texttt{lc\_at}\ (k+1)\ [\overline{n} : |\overline{x}|]\ (\texttt{app}\ t'\ v)$.

- $t \equiv \texttt{let}\ \overline{t}\ \texttt{in}\ t'$.
  Since $\texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}(\texttt{let}\ \overline{t}\ \texttt{in}\ t'))$,
  $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{let}\ (\{k+1 \to \overline{x}\}\overline{t})\ \texttt{in}\ (\{k+1 \to \overline{x}\}t'))$.
  Thus, $\texttt{lc\_at}\ (k+1)\ [|\overline{t}| : \overline{n}]\ (\{k+1 \to \overline{x}\}\overline{t})$
  and $\texttt{lc\_at}\ (k+1)\ [|\overline{t}| : \overline{n}]\ (\{k+1 \to \overline{x}\}t')$.
  By induction hypothesis,
  $\texttt{lc\_at}\ (k+2)\ [|\overline{t}| : \overline{n} : |\overline{x}|]\ \overline{t} \wedge \texttt{lc\_at}\ (k+2)\ [|\overline{t}| : \overline{n} : |\overline{x}|]\ t'$.
  Applying rule LCK-LET, $\texttt{lc\_at}\ (k+1)\ [\overline{n} : |\overline{x}|]\ (\texttt{let}\ \overline{t}\ \texttt{in}\ t')$.

$\square$

Next lemma indicates that if a term is locally closed at level $k+1$ for a given list of natural numbers $[\overline{n} : n]$, then the term open with distinct fresh names $\overline{x}$ (such that $\overline{x} \geq n$) is closed at level k for the list $\overline{n}$.

Lemma 11.
LC_AT_K_FROM_K+1 $\quad k = |\overline{n}| \ \wedge \texttt{lc\_at}\ (k+1)\ [\overline{n} : n]\ t$
$\Rightarrow \forall \overline{x} \subseteq Id, |\overline{x}| \geq n, \texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}t)$

Proof. By structural induction on t.

– $t \equiv \texttt{bvar}\ i\ j$.
Since $\texttt{lc\_at}\ (k+1)\ [\overline{n} : n]\ (\texttt{bvar}\ i\ j)$, $i < k+1 \wedge j < \texttt{List.nth}\ i\ [\overline{n} : n]$
- $i = k \wedge j < \texttt{List.nth}\ k\ [\overline{n} : n] \overset{k=|\overline{n}|}{=} n$
  Let $\overline{x} \subseteq Id$ such that $|\overline{x}| \geq n$.
  Since $\{k \to \overline{x}\}(\texttt{bvar}\ k\ j) = \texttt{fvar}\ (\texttt{List.nth}\ j\ \overline{x})$,
  applying rule LCK-FVAR, $\texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}(\texttt{bvar}\ i\ j))$.
- $i < k \wedge j < \texttt{List.nth}\ i\ [\overline{n} : n] = \texttt{List.nth}\ i\ \overline{n}$
  By rule LCK-BVAR, $\texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}(\texttt{bvar}\ i\ j))$.

– $t \equiv \texttt{fvar}\ x$.
Trivial.

– $t \equiv \texttt{abs}\ t'$.
Since $\texttt{lc\_at}\ (k+1)\ [\overline{n} : n]\ (\texttt{abs}\ t')$, $\texttt{lc\_at}\ (k+2)\ [1 : \overline{n} : n]\ t'$.
By induction hypothesis,
$\forall \overline{x} \subseteq Id, |\overline{x}| \geq n, \texttt{lc\_at}\ (k+1)\ [1 : \overline{n}]\ (\{k+1 \to \overline{x}\}t')$.
Applying rule LCK-ABS,
$\forall \overline{x} \subseteq Id, |\overline{x}| \geq n, \texttt{lc\_at}\ k\ \overline{n}\ (\texttt{abs}\ (\{k+1 \to \overline{x}\}t'))$.
Thus, $\forall \overline{x} \subseteq Id, |\overline{x}| \geq n, \texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}(\texttt{abs}\ t'))$.

– $t \equiv \texttt{app}\ t'\ v$.
Since $\texttt{lc\_at}\ (k+1)\ [\overline{n} : n]\ (\texttt{app}\ t'\ v)$,
$\texttt{lc\_at}\ (k+1)\ [\overline{n} : n]\ t'$ and $\texttt{lc\_at}\ (k+1)\ [\overline{n} : n]\ v$.
By induction hypothesis,
$\forall \overline{x} \subseteq Id, |\overline{x}| \geq n, (\texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}t') \wedge \texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}v))$.
Applying rule LCK-APP,
$\forall \overline{x} \subseteq Id, |\overline{x}| \geq n, \texttt{lc\_at}\ k\ \overline{n}\ (\texttt{app}\ (\{k \to \overline{x}\}t')\ (\{k \to \overline{x}\}v))$.
Thus, $\forall \overline{x} \subseteq Id, |\overline{x}| \geq n, \texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}(\texttt{app}\ t'\ v))$.

– $t \equiv \texttt{let}\ \bar{t}\ \texttt{in}\ t'$.
Since $\texttt{lc\_at}\ (k+1)\ [\overline{n} : n]\ (\texttt{let}\ \bar{t}\ \texttt{in}\ t')$,
$\texttt{lc\_at}\ (k+2)\ [|\bar{t}| : \overline{n} : n]\ \bar{t}$ and $\texttt{lc\_at}\ (k+2)\ [|\bar{t}| : \overline{n} : n]\ t'$.
By induction hypothesis, $\forall \overline{x} \subseteq Id, |\overline{x}| \geq n$,
$\texttt{lc\_at}\ (k+1)\ [|\bar{t}| : \overline{n}]\ (\{k+1 \to \overline{x}\}\bar{t}) \wedge \texttt{lc\_at}\ (k+1)\ [|\bar{t}| : \overline{n}]\ (\{k+1 \to \overline{x}\}t')$.
Applying rule LCK-LET,
$\forall \overline{x} \subseteq Id, |\overline{x}| \geq n, \texttt{lc\_at}\ k\ \overline{n}\ (\texttt{let}\ (\{k+1 \to \overline{x}\}\bar{t})\ \texttt{in}\ (\{k+1 \to \overline{x}\}t'))$.
Thus, $\forall \overline{x} \subseteq Id, |\overline{x}| \geq n, \texttt{lc\_at}\ k\ \overline{n}\ (\{k \to \overline{x}\}(\texttt{let}\ \bar{t}\ \texttt{in}\ t'))$.

□

Now we are ready to prove that a term is locally closed if and only if is closed at level 0.

Lemma 2 :
LC_IIF_LC_AT $\quad \texttt{lc}\ t \Leftrightarrow \texttt{lc\_at}\ 0\ [\,]\ t$

Proof.
$\Rightarrow$) By structural induction on t:

- $t \equiv \texttt{bvar}\ i\ j$.
  Trivial.

- $t \equiv \texttt{fvar}\ x$.
  Trivial.

- $t \equiv \texttt{abs}\ t'$
  $\texttt{lc}\ (\texttt{abs}\ t')$, then $\forall x \notin L \subseteq Id.\texttt{lc}\ {t'}^{[x]}$.
  By induction hypothesis, $\forall x \notin L \subseteq Id.\texttt{lc\_at}\ 0\ [\,]\ {t'}^{[x]}$.
  Thus, $\forall x \notin L \subseteq Id.\texttt{lc\_at}\ 0\ [\,]\ (\{0 \to x\}t')$.
  By LC_AT_K+1_FROM_K, $\forall x \notin L \subseteq Id.\texttt{lc\_at}\ 1\ [1]\ t'$.
  By LCK-ABS, $\texttt{lc\_at}\ 0\ [\,]\ (\texttt{abs}\ t')$.

- $t \equiv \texttt{app}\ t'\ v$.
  $\texttt{lc}\ (\texttt{app}\ t'\ v)$, then $\texttt{lc}\ t'$ and $\texttt{lc}\ v$.
  By induction hypothesis, $\texttt{lc\_at}\ 0\ [\,]\ t'$ and $\texttt{lc\_at}\ 0\ [\,]\ v$.
  By LCK-APP, $\texttt{lc\_at}\ 0\ [\,]\ (\texttt{app}\ t'\ v)$.

- $t \equiv \texttt{let}\ \overline{t}\ \texttt{in}\ t'$.
  $\texttt{lc}\ (\texttt{let}\ \overline{t}\ \texttt{in}\ t')$, then $\forall \overline{x}^{|\overline{t}|} \notin L \subseteq Id.\texttt{lc}\ [t:\overline{t}]^{\overline{x}}$.
  By induction hypothesis, $\forall \overline{x}^{|\overline{t}|} \notin L \subseteq Id.\texttt{lc\_at}\ 0\ [\,]\ [t:\overline{t}]^{\overline{x}}$.
  Thus, $\forall \overline{x}^{|\overline{t}|} \notin L \subseteq Id.\texttt{lc\_at}\ 0\ [\,]\ \{0 \to \overline{x}\}[t:\overline{t}]$.
  By LC_AT_K+1_FROM_K, $\forall \overline{x}^{|\overline{t}|} \notin L \subseteq Id.\texttt{lc\_at}\ 1\ [|\overline{x}|]\ [t:\overline{t}]$.
  By LCK-LET, $\texttt{lc\_at}\ 0\ [\,]\ (\texttt{let}\ \overline{t}\ \texttt{in}\ t')$.

$\Leftarrow$) By structural induction on t:

- $t \equiv \texttt{bvar}\ i\ j$.
  Trivial, since this case is not possible.
  $\texttt{lc\_at}\ 0\ [\,]\ (\texttt{bvar}\ i\ j) \Rightarrow i < 0 \wedge j < \texttt{List.nth}\ i\ [\,]$, the empty list has no elements.

- $t \equiv \texttt{fvar}\ x$.
  Trivial.

– $t \equiv$ `abs` t'.
Since `lc_at` $0\ [\,]$ (`abs` t'), `lc_at` $1\ [1]\ t'$.
By LC_AT_K_FROM_K+1, $\forall \overline{x} \subseteq Id, |\overline{x}| \geq 1.$`lc_at` $0\ [\,]\ (\{0 \to \overline{x}\}t')$.
Thus, $\forall x \notin \emptyset \subseteq Id.$`lc_at` $0\ [\,]\ {t'}^{[x]}$.
By induction hypothesis, $\forall x \notin \emptyset \subseteq Id.$`lc` ${t'}^{[x]}$.
By LC-ABS, `lc` (`abs` t').

– $t \equiv$ `app` $t'\ v$.
Since `lc_at` $0\ [\,]$ (`app` $t'\ v$), `lc_at` $0\ [\,]\ t' \wedge$ `lc_at` $0\ [\,]\ v$.
By induction hypothesis, `lc` $t' \wedge$ `lc` v.
By LC-APP, `lc` (`app` $t'\ v$).

– $t \equiv$ `let` $\overline{t}$ `in` t'.
Since `lc_at` $0\ [\,]$ (`let` $\overline{t}$ `in` t'), `lc_at` $1\ [|\overline{t}|]\ \overline{t} \wedge$ `lc_at` $1\ [|\overline{t}|]\ t'$.
By LC_AT_K_FROM_K+1, $\forall \overline{x} \subseteq Id, |\overline{x}| \geq |\overline{t}|$
(`lc_at` $0\ [\,]\ (\{0 \to \overline{x}\}\overline{t}) \wedge$ `lc_at` $0\ [\,]\ (\{0 \to \overline{x}\}t')$).
Thus, $\forall \overline{x}^{|\overline{t}|} \notin \emptyset \subseteq Id.$(`lc_at` $0\ [\,]\ \overline{t}^{\overline{x}} \wedge$ `lc_at` $0\ [\,]\ {t'}^{\overline{x}}$).
By induction hypothesis, $\forall \overline{x}^{|\overline{t}|} \notin \emptyset \subseteq Id.$(`lc` $\overline{t}^{\overline{x}} \wedge$ `lc` ${t'}^{\overline{x}}$).
By LC-LET, `lc` (`let` $\overline{t}$ `in` t').

□

6.3 Proof of Lemma 3: CLOSE_OPEN_VAR and OPEN_CLOSE_VAR

Lemma 3 states that variable opening and variable closing are inverse functions under some side conditions. Its proof requires another two auxiliary lemmas. Lemma 12 expresses that opening a term at level k and then closing the result at the same level with the same names produces the original term whenever the chosen names to develop the opening and closing operations are fresh in the term.

Lemma 12.
CLOSE_OPEN_VAR_K `fresh` $\overline{x}$ `in` $t \Rightarrow \{k \leftarrow \overline{x}\}(\{k \to \overline{x}\}t) = t$

Proof. By structural induction on t:

– $t \equiv$ `bvar` $i\ j$.

$$\begin{aligned} &\{k \leftarrow \overline{x}\}(\{k \to \overline{x}\}(\texttt{bvar}\ i\ j)) \\ &\quad = \begin{cases} \{k \leftarrow \overline{x}\}(\texttt{fvar}\ (\texttt{List.nth}\ j\ \overline{x})) & \text{if } i = k \wedge j < |\overline{x}| \\ \{k \leftarrow \overline{x}\}(\texttt{bvar}\ i\ j) & \text{otherwise} \end{cases} \\ &\quad = \texttt{bvar}\ i\ j. \end{aligned}$$

– $t \equiv$ `fvar` x.
If `fresh` $\overline{x}$ `in` (`fvar` x), then $x \notin \overline{x}$.
Thus, $\{k \leftarrow \overline{x}\}(\{k \to \overline{x}\}(\texttt{bvar}\ i\ j)) = \{k \leftarrow \overline{x}\}(\texttt{fvar}\ x) = \texttt{fvar}\ x$.

– $t \equiv \texttt{abs}\ t'$.
If $\texttt{fresh}\ \overline{x}\ \texttt{in}\ (\texttt{abs}\ t')$, then $\texttt{fresh}\ \overline{x}\ \texttt{in}\ t'$. Thus,

$$\begin{aligned}\{k \leftarrow \overline{x}\}(\{k \to \overline{x}\}(\texttt{abs}\ t')) &= \{k \leftarrow \overline{x}\}(\texttt{abs}\ (\{k+1 \to \overline{x}\}t')) \\ &= \texttt{abs}\ (\{k+1 \leftarrow \overline{x}\}(\{k+1 \to \overline{x}\}t')) \\ \text{I.H.} \qquad &= \texttt{abs}\ t'.\end{aligned}$$

– $t \equiv \texttt{app}\ t'\ v$.
If $\texttt{fresh}\ \overline{x}\ \texttt{in}\ (\texttt{app}\ t'\ v)$, then $\texttt{fresh}\ \overline{x}\ \texttt{in}\ t'$ and $\texttt{fresh}\ \overline{x}\ \texttt{in}\ v$. Thus,

$$\begin{aligned}\{k \leftarrow \overline{x}\}(\{k \to \overline{x}\}(\texttt{app}\ t'\ v)) &= \{k \leftarrow \overline{x}\}(\texttt{app}\ (\{k \to \overline{x}\}t')\ (\{k \to \overline{x}\}v)) \\ &= \texttt{app}\ (\{k \leftarrow \overline{x}\}(\{k \to \overline{x}\}t'))\ (\{k \leftarrow \overline{x}\}(\{k \to \overline{x}\}v)) \\ \text{I.H.} \qquad &= \texttt{app}\ t'\ v.\end{aligned}$$

– $t \equiv \texttt{let}\ \bar{t}\ \texttt{in}\ t'$.
If $\texttt{fresh}\ \overline{x}\ \texttt{in}\ (\texttt{let}\ \bar{t}\ \texttt{in}\ t')$, then $\texttt{fresh}\ \overline{x}\ \texttt{in}\ \bar{t}$ and $\texttt{fresh}\ \overline{x}\ \texttt{in}\ t'$. Thus,

$$\begin{aligned}&\{k \leftarrow \overline{x}\}(\{k \to \overline{x}\}(\texttt{let}\ \bar{t}\ \texttt{in}\ t')) \\ &\quad = \{k \leftarrow \overline{x}\}(\texttt{let}\ (\{k+1 \to \overline{x}\}\bar{t})\ \texttt{in}\ (\{k+1 \to \overline{x}\}t')) \\ &\quad = \texttt{let}\ (\{k+1 \leftarrow \overline{x}\}(\{k+1 \to \overline{x}\}\bar{t}))\ \texttt{in}\ (\{k+1 \leftarrow \overline{x}\}(\{k+1 \to \overline{x}\}t')) \\ \text{I.H.} &\quad = \texttt{let}\ \bar{t}\ \texttt{in}\ t'.\end{aligned}$$

□

The second result (Lemma 13) establishes that closing a term at level k and then opening the result with the same names at the same level gives back the original term, when the term is closed at level k.

Lemma 13.
OPEN_CLOSE_VAR_K $\qquad \texttt{lc\_at}\ k\ \overline{n}\ t \Rightarrow \{k \to \overline{x}\}(\{k \leftarrow \overline{x}\}t) = t$

Proof. By structural induction on t:

– $t \equiv \texttt{bvar}\ i\ j$.
If $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{bvar}\ i\ j)$, then $i < k$ and $j < \texttt{List.nth}\ i\ \overline{n}$.
Thus, $\{k \to \overline{x}\}(\{k \leftarrow \overline{x}\}(\texttt{bvar}\ i\ j)) = \{k \to \overline{x}\}(\texttt{bvar}\ i\ j) = \texttt{bvar}\ i\ j$.

– $t \equiv \texttt{fvar}\ x$.

$$\begin{aligned}&\{k \to \overline{x}\}(\{k \leftarrow \overline{x}\}(\texttt{fvar}\ x)) \\ &\qquad = \begin{cases}\{k \to \overline{x}\}(\texttt{bvar}\ k\ j) & \text{if } \exists j : 0 \le j < |\overline{x}|. x = \texttt{List.nth}\ j\ \overline{x} \\ \{k \to \overline{x}\}(\texttt{fvar}\ x) & \text{otherwise}\end{cases} \\ &\qquad = \begin{cases}\texttt{fvar}\ (\texttt{List.nth}\ j\ \overline{x}) & \text{if } \exists j : 0 \le j < |\overline{x}|. x = \texttt{List.nth}\ j\ \overline{x} \\ \texttt{fvar}\ x & \text{otherwise}\end{cases} \\ &\qquad = \texttt{fvar}\ x.\end{aligned}$$

– $t \equiv \texttt{abs}\ t'$.
If $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{abs}\ t')$, then $\texttt{lc\_at}\ (k+1)\ [1:\overline{n}]\ t'$. Thus,

$$\begin{aligned}\{k \to \overline{x}\}(\{k \leftarrow \overline{x}\}(\texttt{abs}\ t')) &= \{k \to \overline{x}\}(\texttt{abs}\ (\{k+1 \leftarrow \overline{x}\}t')) \\ &= \texttt{abs}\ (\{k+1 \to \overline{x}\}(\{k+1 \leftarrow \overline{x}\}t')) \\ {\scriptstyle \text{I.H.}}\quad &= \texttt{abs}\ t'\end{aligned}$$

– $t \equiv \texttt{app}\ t'\ v$.
If $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{app}\ t'\ v)$, then $\texttt{lc\_at}\ k\ \overline{n}\ t'$ and $\texttt{lc\_at}\ k\ \overline{n}\ v$. Thus,

$$\begin{aligned}\{k \to \overline{x}\}(\{k \leftarrow \overline{x}\}(\texttt{app}\ t'\ v)) &= \{k \to \overline{x}\}(\texttt{app}\ (\{k \leftarrow \overline{x}\}t')\ (\{k \leftarrow \overline{x}\}v)) \\ &= \texttt{app}\ (\{k \to \overline{x}\}(\{k \leftarrow \overline{x}\}t'))\ (\{k \to \overline{x}\}(\{k \leftarrow \overline{x}\}v)) \\ {\scriptstyle \text{I.H.}}\quad &= \texttt{app}\ t'\ v\end{aligned}$$

– $t \equiv \texttt{let}\ \overline{t}\ \texttt{in}\ t'$.
If $\texttt{lc\_at}\ k\ \overline{n}\ (\texttt{let}\ \overline{t}\ \texttt{in}\ t')$, then
$\texttt{lc\_at}\ (k+1)\ [|\overline{t}|:\overline{n}]\ \overline{t}$ and $\texttt{lc\_at}\ (k+1)\ [|\overline{t}|:\overline{n}]\ t'$. Thus,

$$\begin{aligned}&\{k \to \overline{x}\}(\{k \leftarrow \overline{x}\}(\texttt{let}\ \overline{t}\ \texttt{in}\ t')) \\ &\quad = \{k \to \overline{x}\}(\texttt{let}\ (\{k+1 \leftarrow \overline{x}\}\overline{t})\ \texttt{in}\ (\{k+1 \leftarrow \overline{x}\}t')) \\ &\quad = \texttt{let}\ (\{k+1 \to \overline{x}\}(\{k+1 \leftarrow \overline{x}\}\overline{t}))\ \texttt{in}\ (\{k+1 \to \overline{x}\}(\{k+1 \leftarrow \overline{x}\}t')) \\ {\scriptstyle \text{I.H.}}\ &\quad = \texttt{let}\ \overline{t}\ \texttt{in}\ t'.\end{aligned}$$

□

Now the proof of Lemma 3 is straightforward.

Lemma 3

CLOSE_OPEN_VAR $\quad \texttt{fresh}\ \overline{x}\ \texttt{in}\ t \Rightarrow {}^{\backslash\overline{x}}(t^{\overline{x}}) = t$

OPEN_CLOSE_VAR $\quad \texttt{lc}\ t \Rightarrow ({}^{\backslash\overline{x}}t)^{\overline{x}} = t$

Proof.

– CLOSE_OPEN_VAR is a corollary of Lemma 12 (take $k = 0$).
– OPEN_CLOSE_VAR is a corollary of Lemma 13 (take $k = 0$).

□

6.4 Proof of Lemma 4: OK_SUBS_OK

Lemmas 14 and 15 are needed to prove Lemma 4. Every variable in the domain of a heap where variable x has been substituted by y is either in the domain of the original heap, or coincides with y.

Lemma 14.

DOM_SUBS_UNION $\quad \texttt{dom}(\Gamma[y/x]) \subseteq \texttt{dom}(\Gamma) \cup \{y\}$

Proof. By induction on the size of Γ:

– $\Gamma = \emptyset$. Trivial.

– $\Gamma = (\Delta, z \mapsto t)$.

$$\begin{aligned}\mathtt{dom}(\Gamma[y/x]) &= \mathtt{dom}((\Delta[y/x], z[y/x] \mapsto t[y/x])) = \mathtt{dom}(\Delta[y/x]) \cup \{z[y/x]\} \\ &\overset{IH}{\subseteq} \mathtt{dom}(\Delta) \cup \{y\} \cup \{z[y/x]\}\end{aligned}$$

- $z = x$.
  $\mathtt{dom}(\Gamma[y/x]) \subseteq \mathtt{dom}(\Delta) \cup \{y\} \cup \{y\} \subseteq \mathtt{dom}(\Delta) \cup \{y\} \cup \{z\} = \mathtt{dom}(\Gamma) \cup \{y\}$

- $z \neq x$.
  $\mathtt{dom}(\Gamma[y/x]) \subseteq \mathtt{dom}(\Delta) \cup \{y\} \cup \{z\} = \mathtt{dom}(\Gamma) \cup \{y\}$

□

Next lemma establishes that substitution preserves local closure

Lemma 15.
LC_SUBS_LC $\quad \mathtt{lc}\ t \Rightarrow \mathtt{lc}\ t[y/x]$

Proof. By structural induction on t:

– $t \equiv \mathtt{bvar}\ i\ j$.
Trivial.

– $t \equiv \mathtt{fvar}\ x$.
Trivial.

– $t \equiv \mathtt{abs}\ t'$.
$\mathtt{lc}\ (\mathtt{abs}\ t') \Rightarrow \forall z \notin L \subseteq Id\ .\ \mathtt{lc}\ t'^{[z]}$.
Let $L' = L \cup \{x\} \Rightarrow \forall z \notin L' \subseteq Id\ .\ \mathtt{lc}\ t'^{[z]}$.
By induction hypothesis, $\forall z \notin L' \subseteq Id\ .\ \mathtt{lc}\ (t'^{[z[y/x]]})$.
Since $z \neq x$, $\forall z \notin L' \subseteq Id\ .\ \mathtt{lc}\ (t'[y/x])^{[z]}$.
By LC-ABS, $\mathtt{lc}\ (\mathtt{abs}\ (t'[y/x]))$.
Thus, $\mathtt{lc}\ (\mathtt{abs}\ t')[y/x]$.

– $t \equiv \mathtt{app}\ t'\ v$.
$\mathtt{lc}\ (\mathtt{app}\ t'\ v) \Rightarrow \mathtt{lc}\ t' \wedge \mathtt{lc}\ v$.
By induction hypothesis, $\mathtt{lc}\ t'[y/x] \wedge \mathtt{lc}\ v[y/x]$.
By LC-APP, $\mathtt{lc}\ \mathtt{app}\ (t'[y/x])\ (v[y/x])$.
Thus, $\mathtt{lc}\ (\mathtt{app}\ t'\ v)[y/x]$.

– $t \equiv \mathtt{let}\ \overline{t}\ \mathtt{in}\ t'$.
$\mathtt{lc}\ \mathtt{let}\ \overline{t}\ \mathtt{in}\ t' \Rightarrow \forall \overline{z}^{|\overline{t}|} \notin L \subseteq Id\ .\ \mathtt{lc}\ [t : \overline{t}]^{\overline{z}}$.
Let $L' = L \cup \{x\} \Rightarrow \forall \overline{z}^{|\overline{t}|} \notin L' \subseteq Id\ .\ \mathtt{lc}\ [t : \overline{t}]^{\overline{z}}$.
By induction hypothesis, $\forall \overline{z}^{|\overline{t}|} \notin L' \subseteq Id\ .\ \mathtt{lc}\ ([t : \overline{t}]^{\overline{z}}[y/x])$.
Since $x \notin \overline{z}$, $\forall \overline{z}^{|\overline{t}|} \notin L' \subseteq Id\ .\ \mathtt{lc}\ ([t : \overline{t}][y/x]^{\overline{z}})$.
By LC-LET, $\mathtt{lc}\ (\mathtt{let}\ (\overline{t}[y/x])\ \mathtt{in}\ (t'[y/x]))$.
Thus, $\mathtt{lc}\ (\mathtt{let}\ \overline{t}\ \mathtt{in}\ t')[y/x]$.

□

Now we can prove Lemma 4:

Lemma 4
OK_SUBS_OK $\quad$ $\texttt{ok}\ \Gamma \wedge y \notin \texttt{dom}(\Gamma) \Rightarrow \texttt{ok}\ \Gamma[y/x]$

Proof. By rule induction on the size of Γ:

- $\Gamma = \emptyset$. Trivial.

- $\Gamma = (\Delta, z \mapsto t)$.
  $\texttt{ok}\ (\Delta, z \mapsto t) \Rightarrow \texttt{ok}\ \Delta \wedge z \notin \texttt{dom}(\Delta) \wedge \texttt{lc}\ t$.
  Let $y \notin \texttt{dom}(\Delta, z \mapsto t) = \texttt{dom}(\Delta) \cup \{z\} \Rightarrow y \notin \texttt{dom}(\Delta) \wedge y \neq z$.
  By induction hypothesis, $\texttt{ok}\ \Delta[y/x]$.
  - Case $z \neq x$:
    $\texttt{dom}(\Delta[y/x]) \overset{L14}{\subseteq} \texttt{dom}(\Delta) \cup \{y\}$.
    Since $z \notin \texttt{dom}(\Delta)$ and $z \neq y$, then $z \notin \texttt{dom}(\Delta[y/x])$.

  - Case $z = x$:
    $z = x \Rightarrow x \notin \texttt{dom}(\Delta) \Rightarrow \texttt{dom}(\Delta[y/x]) = \texttt{dom}(\Delta)$.
    Thus, $y \notin \texttt{dom}(\Delta[y/x])$.

  By Lemma 15, $\texttt{lc}\ t[y/x]$.
  Thus, $\texttt{ok}\ (\Delta, z \mapsto t)[y/x]$, i.e., $\texttt{ok}\ \Gamma[y/x]$.

□

6.5 Proof of Lemma 5: REGULARITY

Lemma 5
REGULARITY $\quad$ $\Gamma : t \Downarrow \Delta : w \Rightarrow \texttt{ok}\ \Gamma \wedge \texttt{lc}\ t \wedge \texttt{ok}\ \Delta \wedge \texttt{lc}\ w$.

Proof. By rule induction:

- LNLam.
  Trivial.

- LNVar.
  By induction hypothesis, $\texttt{ok}\ \Gamma \wedge \texttt{lc}\ t \wedge \texttt{ok}\ \Delta \wedge \texttt{lc}\ w$.
  Since $x \notin \texttt{dom}(\Gamma)$, $x \notin \texttt{dom}(\Delta)$,
  then $\texttt{ok}\ (\Gamma, x \mapsto t)$ and $\texttt{ok}\ (\Delta, x \mapsto w)$ and $\texttt{lc}\ (\texttt{fvar}\ x)$ by definition.

- LNApp.
  By induction hypothesis, $\texttt{ok}\ \Gamma \wedge \texttt{lc}\ t \wedge \texttt{ok}\ \Theta \wedge \texttt{lc}\ (\texttt{abs}\ u)$.
  By induction hypothesis, $\texttt{ok}\ \Theta \wedge \texttt{lc}\ u^{[x]} \wedge \texttt{ok}\ \Delta \wedge \texttt{lc}\ w$.
  Since $\texttt{lc}\ t$ and $\texttt{lc}\ (\texttt{fvar}\ x)$, then $\texttt{lc}\ (\texttt{app}\ t\ (\texttt{fvar}\ x))$.

- LNLet.
  By induction hypothesis,
  $\forall \overline{x}^{|\overline{t}|} \notin L . \texttt{ok}\ (\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) \wedge \texttt{lc}\ t^{\overline{x}} \wedge \texttt{ok}\ (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) \wedge \texttt{lc}\ w^{\overline{x}}$.
  Particularly for $\overline{y}^{|\overline{t}|} \notin L . \texttt{ok}\ (\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) \wedge \texttt{lc}\ w^{\overline{y}}$.
  Since $\forall \overline{x}^{|\overline{t}|} \notin L . \texttt{ok}\ (\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}})$, then $\texttt{ok}\ \Gamma \wedge \forall \overline{x}^{|\overline{t}|} \notin L . (\overline{x} \notin \texttt{dom}(\Gamma) \wedge \texttt{lc}\ \overline{t}^{\overline{x}})$.
  Since $\forall \overline{x}^{|\overline{t}|} \notin L . (\texttt{lc}\ \overline{t}^{\overline{x}} \wedge \texttt{lc}\ t^{\overline{x}})$, then $\texttt{lc}\ (\texttt{let}\ \overline{t}\ \texttt{in}\ t)$.

□

6.6 Proofs of Lemmas 6 and 7: DEF_NOT_LOST and ADD_VARS

Lemma 6
DEF_NOT_LOST $\quad \Gamma : t \Downarrow \Delta : w \Rightarrow \texttt{dom}(\Gamma) \subseteq \texttt{dom}(\Delta)$.

Proof. By rule induction:

- LNLam.
  Trivial.

- LNVar.
  By induction hypothesis,
  $\texttt{dom}(\Gamma) \subseteq \texttt{dom}(\Delta) \Rightarrow \texttt{dom}(\Gamma, x \mapsto t) \subseteq \texttt{dom}(\Delta, x \mapsto w)$.

- LNApp.
  By induction hypothesis, $\texttt{dom}(\Gamma) \subseteq \texttt{dom}(\Theta)$ and $\texttt{dom}(\Theta) \subseteq \texttt{dom}(\Delta)$.
  By transitivity, $\texttt{dom}(\Gamma) \subseteq \texttt{dom}(\Delta)$.

- LNLet.
  By induction hypothesis,
  $\forall \overline{x}^{|\overline{t}|} \notin L \subseteq Id\ .\ \texttt{dom}(\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) \subseteq \texttt{dom}(\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}})$.
  Particularly for $\overline{y}^{|\overline{t}|} \notin L \subseteq Id$,
  $\texttt{dom}(\Gamma, \overline{y} \mapsto \overline{t}^{\overline{y}}) = \texttt{dom}(\Gamma) \cup \{\overline{y}\} \subseteq \texttt{dom}(\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}})$.
  Thus, $\texttt{dom}(\Gamma) \subseteq \texttt{dom}(\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}})$.

□

Lemma 7
ADD_VARS $\quad \Gamma : t \Downarrow \Delta : w$
$\Rightarrow (x \in \texttt{names}(\Delta : w) \Rightarrow (x \in \texttt{dom}(\Delta) \vee x \in \texttt{names}(\Gamma : t)))$.

Proof. It is equivalent to prove

$$\Gamma : t \Downarrow \Delta : w \Rightarrow \texttt{names}(\Delta : w) \subseteq \texttt{dom}(\Delta) \cup \texttt{names}(\Gamma : t).$$

By rule induction:

- LNLam.
  Trivial.

– LNVAR.

$$\begin{aligned}\texttt{names}((\Delta, x \mapsto w) : w) &= \texttt{names}(\Delta : w) \cup \{x\} \\ &\overset{IH}{\subseteq} \texttt{dom}(\Delta) \cup \texttt{names}(\Gamma : t) \cup \{x\} \\ &= \texttt{dom}(\Delta) \cup \texttt{names}(\Gamma) \cup \texttt{fv}(t) \cup \{x\} \\ &= \texttt{dom}(\Delta, x \mapsto w) \cup \texttt{names}(\Gamma, x \mapsto t) \cup \texttt{fv}(\texttt{fvar}\ x) \\ &= \texttt{dom}(\Delta, x \mapsto w) \cup \texttt{names}((\Gamma, x \mapsto t) : \texttt{fvar}\ x).\end{aligned}$$

– LNAPP.

$$\begin{aligned}\texttt{names}(\Delta : w) &\overset{IH}{\subseteq} \texttt{dom}(\Delta) \cup \texttt{names}(\Theta : u^{[x]}) \\ &\subseteq \texttt{dom}(\Delta) \cup \texttt{names}(\Theta) \cup \texttt{fv}(u) \cup \{x\} \\ &= \texttt{dom}(\Delta) \cup \texttt{names}(\Theta) \cup \texttt{fv}(\texttt{abs}\ u) \cup \texttt{fv}(\texttt{fvar}\ x) \\ &= \texttt{dom}(\Delta) \cup \texttt{names}(\Theta : \texttt{abs}\ u) \cup \texttt{fv}(\texttt{fvar}\ x) \\ &\overset{IH}{\subseteq} \texttt{dom}(\Delta) \cup \texttt{dom}(\Theta) \cup \texttt{names}(\Gamma : t) \cup \texttt{fv}(\texttt{fvar}\ x) \\ &\overset{L6}{=} \texttt{dom}(\Delta) \cup \texttt{names}(\Gamma : t) \cup \texttt{fv}(\texttt{fvar}\ x) \\ &= \texttt{dom}(\Delta) \cup \texttt{names}(\Gamma) \cup \texttt{fv}(t) \cup \texttt{fv}(\texttt{fvar}\ x) \\ &= \texttt{dom}(\Delta) \cup \texttt{names}(\Gamma) \cup \texttt{fv}(\texttt{app}\ t\ (\texttt{fvar}\ x)) \\ &= \texttt{dom}(\Delta) \cup \texttt{names}(\Gamma : \texttt{app}\ t\ (\texttt{fvar}\ x)).\end{aligned}$$

– LNLET.

$\forall \overline{x}^{|\overline{t}|} \notin L \subseteq Id$

$$\texttt{names}((\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}}) \overset{IH}{\subseteq} \texttt{dom}(\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) \cup \texttt{names}((\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}}).$$

Particularly for $\overline{y}^{|\overline{t}|} \notin L \subseteq Id$:

$$\begin{aligned}\texttt{names}((\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) : w^{\overline{y}}) &\overset{IH}{\subseteq} \texttt{dom}(\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) \cup \texttt{names}((\Gamma, \overline{y} \mapsto \overline{t}^{\overline{y}}) : t^{\overline{y}}) \\ &= \texttt{dom}(\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) \cup \texttt{names}(\Gamma) \cup \{\overline{y}\} \cup \texttt{fv}(\overline{t}^{\overline{y}}) \cup \texttt{fv}(t^{\overline{y}}) \\ &\subseteq \texttt{dom}(\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) \cup \texttt{names}(\Gamma) \cup \{\overline{y}\} \cup \texttt{fv}(\overline{t}) \cup \{\overline{y}\} \cup \texttt{fv}(t) \cup \{\overline{y}\} \\ &= \texttt{dom}(\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) \cup \texttt{names}(\Gamma) \cup \texttt{fv}(\texttt{let}\ \overline{t}\ \texttt{in}\ t) \\ &= \texttt{dom}(\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) \cup \texttt{names}(\Gamma : \texttt{let}\ \overline{t}\ \texttt{in}\ t).\end{aligned}$$

□

6.7 Proof of Lemma 8: RENAMING

Before proving the renaming lemma (Lemma 8) we need some auxiliary results: Corollaries 1 and 2, that are proved by Lemmas 16 and 17 respectively.

Lemma 16.
NOT_OPENK_FV $\quad \texttt{fresh}\ y\ \texttt{in}\ \{k \to \overline{x}\}t \Rightarrow \texttt{fresh}\ y\ \texttt{in}\ t$

Proof. By structural induction on t:

– $t \equiv \texttt{bvar}\ i\ j$.
Trivial, since $\texttt{fv}(\texttt{bvar}\ i\ j) = \emptyset$.

– $t \equiv \texttt{fvar}\ z$.
Trivial, since $\texttt{fv}(\{k \to \overline{x}\}\texttt{fvar}\ z) = \texttt{fv}(\texttt{fvar}\ z) = \{z\}$.

- $t \equiv \texttt{abs}\ t'$.
  Since $\texttt{fresh}\ y\ \texttt{in}\ \{k \to \overline{x}\}(\texttt{abs}\ t')$,
  $y \notin \texttt{fv}(\{k \to \overline{x}\}\texttt{abs}\ t') = \texttt{fv}(\texttt{abs}\ (\{k+1 \to \overline{x}\}t')) = \texttt{fv}(\{k+1 \to \overline{x}\}t')$.
  By induction hypothesis, $y \notin \texttt{fv}(t') = \texttt{fv}(\texttt{abs}\ t')$.

- $t \equiv \texttt{app}\ t'\ v$.
  Since $\texttt{fresh}\ y\ \texttt{in}\ \{k \to \overline{x}\}(\texttt{app}\ t'\ v)$,
  $y \notin \texttt{fv}(\{k \to \overline{x}\}\texttt{app}\ t'\ v)$
  $= \texttt{fv}(\texttt{app}\ (\{k \to \overline{x}\}t')\ (\{k \to \overline{x}\}v))$
  $= \texttt{fv}(\{k \to \overline{x}\}t') \cup \texttt{fv}(\{k \to \overline{x}\}v)$.
  By induction hypothesis, $y \notin \texttt{fv}(t') \wedge y \notin \texttt{fv}(v)$.
  Therefore, $y \notin \texttt{fv}(t') \cup \texttt{fv}(v) = \texttt{fv}(\texttt{app}\ t'\ v)$.

- $t \equiv \texttt{let}\ \bar{t}\ \texttt{in}\ t'$.
  Since $\texttt{fresh}\ y\ \texttt{in}\ \{k \to \overline{x}\}(\texttt{let}\ \bar{t}\ \texttt{in}\ t')$,
  $y \notin \texttt{fv}(\{k \to \overline{x}\}\texttt{let}\ \bar{t}\ \texttt{in}\ t')$
  $= \texttt{fv}(\texttt{let}\ (\{k+1 \to \overline{x}\}\bar{t})\ \texttt{in}\ (\{k+1 \to \overline{x}\}t'))$
  $= \texttt{fv}(\{k+1 \to \overline{x}\}\bar{t}) \cup \texttt{fv}(\{k+1 \to \overline{x}\}t')$.
  By induction hypothesis, $y \notin \texttt{fv}(\bar{t}) \wedge y \notin \texttt{fv}(t')$.
  Therefore, $y \notin \texttt{fv}(\bar{t}) \cup \texttt{fv}(t') = \texttt{fv}(\texttt{let}\ \bar{t}\ \texttt{in}\ t')$.

□

Corollary 1.
NOT_OPEN_FV $\quad \texttt{fresh}\ y\ \texttt{in}\ t^{\overline{x}} \Rightarrow \texttt{fresh}\ y\ \texttt{in}\ t$

Proof. This is a particular case of Lemma 16 ($k = 0$). □

Lemma 17.
FREE_VAR_OPENK $\quad \texttt{fresh}\ \overline{y}\ \texttt{in}\ t \wedge \overline{y} \cap \overline{x} = \emptyset \Rightarrow \texttt{fresh}\ \overline{y}\ \texttt{in}\ \{k \to \overline{x}\}t$

Proof. By structural induction on t:

- $t \equiv \texttt{bvar}\ i\ j$.
  $\overline{y} \notin \texttt{fv}(t) \wedge \overline{y} \cap \overline{x} = \emptyset$.
  $$\texttt{fv}(\{k \to \overline{x}\}(\texttt{bvar}\ i\ j)) = \begin{cases} \texttt{fv}(\texttt{fvar}\ (\texttt{List.nth}\ j\ \overline{x})) & \text{if } i = k \wedge j < |\overline{x}| \\ \texttt{fv}(\texttt{bvar}\ i\ j) & \text{otherwise} \end{cases}$$
  $$= \begin{cases} \texttt{List.nth}\ j\ \overline{x} & \text{if } i = k \wedge j < |\overline{x}| \\ \emptyset & \text{otherwise} \end{cases}$$
  In both cases $\overline{y} \notin \texttt{fv}(\{k \to \overline{y}\}(\texttt{bvar}\ i\ j))$.

- $t \equiv \texttt{fvar}\ z$.
  Trivial, since $\texttt{fv}(\{k \to \overline{x}\}\texttt{fvar}\ z) = \texttt{fv}(\texttt{fvar}\ z) = \{z\}$.

- $t \equiv \texttt{abs}\ t'$.
  $\overline{y} \notin \texttt{fv}(\texttt{abs}\ t') = \texttt{fv}(t') \wedge \overline{y} \cap \overline{x} = \emptyset$.
  By induction hypothesis, $\overline{y} \notin \texttt{fv}(\{k+1 \to \overline{x}\}t') = \texttt{fv}(\{k \to \overline{x}\}\texttt{abs}\ t')$.

– $t \equiv \texttt{app}\ t'\ v$.
$\overline{y} \notin \texttt{fv}(\texttt{app}\ t'\ v) = \texttt{fv}(t') \cup \texttt{fv}(v) \wedge \overline{y} \cap \overline{x}$.
By induction hypothesis $\overline{y} \notin \texttt{fv}(\{k \to \overline{x}\}t') \wedge \overline{y} \notin \texttt{fv}(\{k \to \overline{x}\}v)$.
Thus,
$y \notin \texttt{fv}(\{k \to \overline{x}\}t') \cup \texttt{fv}(\{k \to \overline{x}\}v)$
$= \texttt{fv}(\texttt{app}\ (\{k \to \overline{x}\}t')\ (\{k \to \overline{x}\}v))$
$= \texttt{fv}(\{k \to \overline{x}\}\texttt{app}\ t'\ v)$

– $t \equiv \texttt{let}\ \overline{t}\ \texttt{in}\ t'$.
$\overline{y} \notin \texttt{fv}(\texttt{let}\ \overline{t}\ \texttt{in}\ t') = \texttt{fv}(\overline{t}) \cup \texttt{fv}(t') \wedge \overline{y} \cap \overline{x}$.
By induction hypothesis $\overline{y} \notin \texttt{fv}(\{k+1 \to \overline{x}\}\overline{t}) \wedge \overline{y} \notin \texttt{fv}(\{k+1 \to \overline{x}\}t')$.
Thus,
$y \notin \texttt{fv}(\{k+1 \to \overline{x}\}\overline{t}) \cup \texttt{fv}(\{k+1 \to \overline{x}\}t')$
$= \texttt{fv}(\texttt{let}\ (\{k+1 \to \overline{x}\}\overline{t})\ \texttt{in}\ (\{k+1 \to \overline{x}\}t'))$
$= \texttt{fv}(\{k \to \overline{x}\}\texttt{let}\ \overline{t}\ \texttt{in}\ t')$.

□

Corollary 2.
FREE_VAR_OPEN $\quad \texttt{fresh}\ \overline{y}\ \texttt{in}\ t \wedge \overline{y} \cap \overline{x} = \emptyset \Rightarrow \texttt{fresh}\ \overline{y}\ \texttt{in}\ t^{\overline{x}}$

Proof. Take $k = 0$ in Lemma 17. □

Another auxiliary result is needed:

Lemma 18.
NOT_SUBS_DOM $\quad z \notin \texttt{dom}(\Gamma[y/x]) \wedge z \neq x \Rightarrow z \notin \texttt{dom}(\Gamma)$

Proof. By induction on the size of Γ:

– $\Gamma = \emptyset$. Trivial.

– $\Gamma = (\Delta, x' \mapsto t)$.
$\texttt{dom}(\Gamma[y/x]) = \texttt{dom}((\Delta[y/x], x'[y/x] \mapsto t[y/x])) = \texttt{dom}(\Delta[y/x]) \cup \{x'[y/x]\}$.
$z \notin \texttt{dom}(\Gamma[y/x]) = \texttt{dom}(\Delta[y/x]) \cup \{x'[y/x]\} \stackrel{IH}{\Rightarrow} z \notin \texttt{dom}(\Delta) \cup \{x'[y/x]\}$.
  - $x' = x$.
  $z \notin \texttt{dom}(\Delta) \cup \{y\} \stackrel{z \neq x}{\Rightarrow} z \notin \texttt{dom}(\Delta) \cup \{y\} \cup \{x\} = \texttt{dom}(\Gamma) \cup \{y\}$
  $\Rightarrow z \notin \texttt{dom}(\Gamma)$.
  - $x' \neq x$.
  $z \notin \texttt{dom}(\Delta) \cup \{x'\} = \texttt{dom}(\Gamma)$.

□

The last auxiliary result that is needed establishes that if a variable x does not belong to the domain of a heap then the domain of the heap where x is substituted by y coincides with the domain of the heap:

Lemma 19.
DOM_SUBS $\quad x \notin \texttt{dom}(\Gamma) \Rightarrow \texttt{dom}(\Gamma[y/x]) = \texttt{dom}(\Gamma)$

Proof. By induction on the size of Γ:

- $\Gamma = \emptyset$. Trivial.

- $\Gamma = (\Delta, z \mapsto t)$.
  $$x \notin \texttt{dom}(\Gamma) \Rightarrow x \notin \texttt{dom}(\Delta) \cup \{z\} \Rightarrow \begin{cases} x \notin \texttt{dom}(\Delta) \overset{IH}{\Rightarrow} \texttt{dom}(\Delta[y/x]) = \texttt{dom}(\Delta) \\ x \neq z \end{cases}$$
  $$\begin{aligned}\texttt{dom}(\Gamma[y/x]) &= \texttt{dom}(\Delta[y/x], z \mapsto t[y/x]) = \texttt{dom}(\Delta[y/x]) \cup \{z\} \\ &= \texttt{dom}(\Delta) \cup \{z\} = \texttt{dom}(\Gamma)\end{aligned}$$

□

And now we prove the renaming lemma.

Lemma 8
RENAMING $\quad \Gamma : t \Downarrow \Delta : w \wedge \texttt{fresh } y \texttt{ in } (\Gamma : t) \wedge \texttt{fresh } y \texttt{ in } (\Delta : w)$
$\Rightarrow \Gamma[y/x] : t[y/x] \Downarrow \Delta[y/x] : w[y/x]$.

Proof. By rule induction:

- LNLAM.
  $\Gamma : \texttt{abs } t \Downarrow \Gamma : \texttt{abs } t \Rightarrow \{\texttt{ok } \Gamma\} \wedge \{\texttt{lc abs } t\}$.
  $\texttt{ok } \Gamma \wedge y \notin \texttt{names}(\Gamma : \texttt{abs } t) \Rightarrow \texttt{ok } \Gamma \wedge y \notin \texttt{dom}(\Gamma) \overset{L4}{\Rightarrow} \texttt{ok } \Gamma[y/x]$.
  $\texttt{lc } (\texttt{abs } t) \overset{L15}{\Rightarrow} \texttt{lc } (\texttt{abs } t)[y/x]$.
  By rule LNLAM, $\Gamma[y/x] : (\texttt{abs } t)[y/x] \Downarrow \Gamma[y/x] : (\texttt{abs } t)[y/x]$.

- LNVAR.
  $(\Gamma, z \mapsto t) : (\texttt{fvar } z) \Downarrow (\Delta, z \mapsto w) : w \Rightarrow$
  $\Gamma : t \Downarrow \Delta : w \wedge \{z \notin \texttt{dom}(\Gamma) \cup \texttt{dom}(\Delta)\}$.
  $$\begin{aligned} y \notin\ & \texttt{names}((\Gamma, z \mapsto t) : \texttt{fv}(z)) \cup \texttt{names}((\Delta, z \mapsto w) : w) \\ &= \texttt{names}(\Gamma) \cup \texttt{names}(\Delta) \cup \{z\} \cup \texttt{fv}(t) \cup \texttt{fv}(w) \\ &\Rightarrow y \notin \texttt{names}(\Gamma) \cup \texttt{names}(\Delta) \cup \texttt{fv}(t) \cup \texttt{fv}(w) \\ &\Rightarrow y \notin \texttt{names}(\Gamma : t) \cup \texttt{names}(\Delta : w). \end{aligned}$$
  By induction hypothesis, $\Gamma[y/x] : t[y/x] \Downarrow \Delta[y/x] : w[y/x]$.
  To prove: $z[y/x] \notin \texttt{dom}(\Gamma[y/x]) \cup \texttt{dom}(\Delta[y/x])$

  1. $z \neq x \Rightarrow z \neq y$
     $\texttt{dom}(\Gamma[y/x]) \cup \texttt{dom}(\Delta[y/x]) \overset{L14}{\subseteq} \texttt{dom}(\Gamma) \cup \texttt{dom}(\Delta) \cup \{y\}$.
     $z \notin \texttt{dom}(\Gamma) \cup \texttt{dom}(\Delta) \wedge y \neq z \Rightarrow z \notin \texttt{dom}(\Gamma) \cup \texttt{dom}(\Delta) \cup \{y\}$
     $\Rightarrow z \notin \texttt{dom}(\Gamma[y/x]) \cup \texttt{dom}(\Delta[y/x])$.
  2. $z = x \Rightarrow z[y/x] = y$.
     $y \notin \texttt{names}(\Gamma) \cup \texttt{names}(\Delta)$
     $\Rightarrow y \notin \texttt{dom}(\Gamma) \cup \texttt{dom}(\Delta) \overset{L19}{=} \texttt{dom}(\Gamma[y/x]) \cup \texttt{dom}(\Delta[y/x])$.

  By rule LNVAR, $(\Gamma, z \mapsto t)[y/x] : (\texttt{fvar } z)[y/x] \Downarrow (\Delta, z \mapsto w)[y/x] : w[y/x]$.

– LNApp.
$\Gamma : \texttt{app}\ t\ (\texttt{fvar}\ z) \Downarrow \Delta : w$
$\Rightarrow \Gamma : t \Downarrow \Theta : \texttt{abs}\ u \wedge \Theta : u^{[z]} \Downarrow \Delta : w \wedge \{z \notin \texttt{dom}(\Gamma) \Rightarrow z \notin \texttt{dom}(\Delta)\}$.

$$\begin{aligned}
\texttt{names}(\Gamma : t) &\subseteq \texttt{names}(\Gamma : \texttt{app}\ t\ (\texttt{fvar}\ z)) \\
&\subseteq \texttt{names}(\Gamma : \texttt{app}\ t\ (\texttt{fvar}\ z)) \cup \texttt{names}(\Delta : w).
\end{aligned}$$

$$\begin{aligned}
\texttt{names}(\Theta : \texttt{abs}\ u) &\overset{L7}{\subseteq} \texttt{dom}(\Theta) \cup \texttt{names}(\Gamma : t) \overset{L6}{\subseteq} \texttt{dom}(\Delta) \cup \texttt{names}(\Gamma : t) \\
&\subseteq \texttt{names}(\Delta) \cup \texttt{names}(\Gamma) \cup \texttt{fv}(t) \\
&\subseteq \texttt{names}(\Delta) \cup \texttt{names}(\Gamma) \cup \texttt{fv}(\texttt{app}\ t\ (\texttt{fvar}\ z)) \cup \texttt{fv}(w) \\
&= \texttt{names}(\Gamma : \texttt{app}\ t\ (\texttt{fvar}\ z)) \cup \texttt{names}(\Delta : w).
\end{aligned}$$

$y \notin \texttt{names}(\Gamma : \texttt{app}\ t\ (\texttt{fvar}\ z)) \cup \texttt{names}(\Delta : w)$
$\Rightarrow y \notin \texttt{names}(\Gamma : t) \cup \texttt{names}(\Theta : \texttt{abs}\ u)$.

By induction hypothesis,

$$\Gamma[y/x] : t[y/x] \Downarrow \Theta[y/x] : \underbrace{(\texttt{abs}\ u)[y/x]}_{\texttt{abs}\ u[y/x]} \tag{1}$$

By open_var_fv in [4] ($\texttt{fv}(u^{[z]}) \subseteq \texttt{fv}(u) \cup \{z\}$),
$\texttt{names}(\Theta : u^{[z]}) = \texttt{names}(\Theta) \cup \texttt{fv}(u^{[z]}) \subseteq \texttt{names}(\Theta) \cup \texttt{fv}(u) \cup \{z\}$.

$$\left.\begin{array}{l} y \notin \texttt{names}(\Theta : \texttt{abs}\ u) = \texttt{names}(\Theta) \cup \texttt{fv}(u) \\ y \notin \texttt{names}(\Gamma : \texttt{app}\ t\ (\texttt{fvar}\ z)) \Rightarrow y \neq z \end{array}\right\} \Rightarrow y \notin \texttt{names}(\Theta : u^{[z]}).$$

By induction hypothesis,

$$\Theta[y/x] : \underbrace{(u^{[z]})[y/x]}_{u[y/x]^{[z[y/x]]}} \Downarrow \Delta[y/x] : w[y/x] \tag{2}$$

To prove: $z[y/x] \notin \texttt{dom}(\Gamma[y/x]) \Rightarrow z[y/x] \notin \texttt{dom}(\Delta[y/x])$.

- $z \neq x \Rightarrow z[y/x] = z$
  $\texttt{dom}(\Delta[y/x]) \overset{L14}{\subseteq} \texttt{dom}(\Delta) \cup \{y\}$.
  $$\left.\begin{array}{l} z \notin \texttt{dom}(\Gamma[y/x]) \overset{L18}{\Rightarrow} z \notin \texttt{dom}(\Gamma) \overset{hip.}{\Rightarrow} z \notin \texttt{dom}(\Delta). \\ y \notin \texttt{names}(\Gamma : \texttt{app}\ t\ (\texttt{fvar}\ z)) \Rightarrow y \neq z \end{array}\right\}$$
  $\Rightarrow z \notin \texttt{dom}(\Delta) \cup \{y\} \Rightarrow z \notin \texttt{dom}(\Delta[y/x])$
- $z = x \Rightarrow z[y/x] = y$.
  $y \notin \texttt{dom}(\Gamma[y/x]) \Rightarrow x \notin \texttt{dom}(\Gamma) \overset{hip.}{\Rightarrow} x \notin \texttt{dom}(\Delta) \overset{L19}{\Rightarrow} \texttt{dom}(\Delta) = \texttt{dom}(\Delta[y/x])$.
  $y \notin \texttt{names}(\Delta : w) \Rightarrow y \notin \texttt{dom}(\Delta) \Rightarrow y \notin \texttt{dom}(\Delta[y/x])$

Therefore,

$$z[y/x] \notin \texttt{dom}(\Gamma[y/x]) \Rightarrow z[y/x] \notin \texttt{dom}(\Delta[y/x]) \tag{3}$$

By 1, 2, 3 and rule LNApp, $\Gamma[y/x] : (\texttt{app}\ t\ (\texttt{fvar}\ z))[y/x] \Downarrow \Delta[y/x] : w[y/x]$.

– LNLet.
$\Gamma : \mathtt{let}\ \overline{t}\ \mathtt{in}\ t \Downarrow (\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) : w^{\overline{y}}$
$\Rightarrow \forall \overline{x}^{|\overline{t}|} \notin L \subseteq Id.(\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}} \wedge \{\overline{y}^{|\overline{t}|} \notin L \subseteq Id\}.$

case: $y \in L$.

- <u>subcase</u>: $x \notin L$.
  Let $L' = L \cup \{x\} - \{y\}$.
  To prove: $\forall \overline{x} \notin L'$.
  $(\Gamma[y/x], \overline{x} \mapsto \overline{t}[y/x]^{\overline{x}}) : t[y/x]^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z}[y/x] \mapsto \overline{u}[y/x]^{\overline{x}}) : w[y/x]^{\overline{x}}$
  Let $\overline{x} \notin L'$.

  subsubcase: $\overline{x} \cap \{y\} = \emptyset \Rightarrow \overline{x} \notin L \cup \{x\} \Rightarrow \overline{x} \cap \{x\} = \emptyset$.
  $\overline{x} \notin L \cup \{x\} \Rightarrow (\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}}$
  $\overline{x} \cap \{y\} = \emptyset$
  $\wedge y \notin \mathtt{names}(\Gamma : \mathtt{let}\ \overline{t}\ \mathtt{in}\ t) \cup \mathtt{names}((\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) : w^{\overline{y}})$
  $= \mathtt{names}(\Gamma) \cup \mathtt{fv}(\overline{t}) \cup \mathtt{fv}(t) \cup \overline{y} \cup \overline{z} \cup \mathtt{fv}(\overline{u}^{\overline{y}}) \cup \mathtt{fv}(w^{\overline{y}})$
  $\overset{C1}{\Rightarrow} y \notin \mathtt{names}(\Gamma) \cup \mathtt{fv}(\overline{t}) \cup \mathtt{fv}(t) \cup \overline{y} \cup \overline{z} \cup \mathtt{fv}(\overline{u}) \cup \mathtt{fv}(w)$
  $\overset{C2}{\Rightarrow} y \notin \mathtt{names}(\Gamma) \cup \mathtt{fv}(\overline{t}^{\overline{x}}) \cup \mathtt{fv}(t^{\overline{x}}) \cup \overline{y} \cup \overline{z} \cup \mathtt{fv}(\overline{u}^{\overline{x}}) \cup \mathtt{fv}(w^{\overline{x}}) \cup \overline{x}$
  $\Rightarrow y \notin \mathtt{names}((\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}}) \cup \mathtt{names}((\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}})$.
  By induction hypothesis,
  $(\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}})[y/x] : (t^{\overline{x}})[y/x] \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}})[y/x] : (w^{\overline{x}})[y/x] \overset{x \cap \overline{x} = \emptyset}{\Rightarrow}$
  $(\Gamma[y/x], \overline{x} \mapsto \overline{t}[y/x]^{\overline{x}}) : t[y/x]^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z}[y/x] \mapsto \overline{u}[y/x]^{\overline{x}}) : w[y/x]^{\overline{x}}$.

  subsubcase: $\overline{x} \cap \{y\} \neq \emptyset$.
  Without lost of generality, consider $\overline{x} = [y : \overline{x'}]$ with $\overline{x'} \cap \{y\} = \emptyset$.
  $\overline{x} \notin L' \Rightarrow \overline{x} \cap \{x\} = \emptyset$.
  Let $\overline{x''} = [x : \overline{x'}] \Rightarrow \overline{x''} \notin L \Rightarrow$
  $(\Gamma, [x : \overline{x'}] \mapsto \overline{t}^{[x:\overline{x'}]}) : t^{[x:\overline{x'}]} \Downarrow ([x : \overline{x'}] +\!\!+ \overline{z} \mapsto \overline{u}^{[x:\overline{x'}]}) : w^{[x:\overline{x'}]}$
  $y \cap \overline{x''} = \emptyset$
  $\wedge y \notin \mathtt{names}(\Gamma : \mathtt{let}\ \overline{t}\ \mathtt{in}\ t) \cup \mathtt{names}((\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) : w^{\overline{y}})$
  $= \mathtt{names}(\Gamma) \cup \mathtt{fv}(\overline{t}) \cup \mathtt{fv}(t) \cup \overline{y} \cup \overline{z} \cup \mathtt{fv}(\overline{u}^{\overline{y}}) \cup \mathtt{fv}(w^{\overline{y}})$
  $\overset{C1}{\Rightarrow} y \notin \mathtt{names}(\Gamma) \cup \mathtt{fv}(\overline{t}) \cup \mathtt{fv}(t) \cup \overline{y} \cup \overline{z} \cup \mathtt{fv}(\overline{u}) \cup \mathtt{fv}(w)$
  $\overset{C2}{\Rightarrow} y \notin \mathtt{names}(\Gamma) \cup \mathtt{fv}(\overline{t}^{\overline{x''}}) \cup \mathtt{fv}(t^{\overline{x''}}) \cup \overline{y} \cup \overline{z} \cup \mathtt{fv}(\overline{u}^{\overline{x''}}) \cup \mathtt{fv}(w^{\overline{x''}}) \cup \overline{x''}$
  $\Rightarrow y \notin \mathtt{names}((\Gamma, \overline{x''} \mapsto \overline{t}^{\overline{x''}}) : t^{\overline{x''}}) \cup \mathtt{names}((\overline{x''} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x''}}) : w^{\overline{x''}})$.
  By induction hypothesis,
  $(\Gamma, [x : \overline{x'}] \mapsto \overline{t}^{[x:\overline{x'}]})[y/x] : (t^{[x:\overline{x'}]})[y/x] \Downarrow ([x : \overline{x'}] +\!\!+ \overline{z} \mapsto \overline{u}^{[x:\overline{x'}]})[y/x] : (w^{[x:\overline{x'}]})[y/x] \Rightarrow$
  $(\Gamma[y/x], [y : \overline{x'}] \mapsto \overline{t}[y/x]^{[y:\overline{x'}]}) : t[y/x]^{[y:\overline{x'}]} \Downarrow ([y : \overline{x'}] +\!\!+ \overline{z}[y/x] \mapsto \overline{u}[y/x]^{[y:\overline{x'}]}) : w[y/x]^{[y:\overline{x'}]} \Rightarrow$
  $(\Gamma[y/x], \overline{x} \mapsto \overline{t}[y/x]^{\overline{x}}) : t[y/x]^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z}[y/x] \mapsto \overline{u}[y/x]^{\overline{x}}) : w[y/x]^{\overline{x}}$.

- <u>SUBCASE:</u> $x \in L$.
  Let $L' = L$.
  To prove: $\forall \overline{x} \notin L'$.
  $(\Gamma[y/x], \overline{x} \mapsto \overline{t}[y/x]^{\overline{x}}) : t[y/x]^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z}[y/x] \mapsto \overline{u}[y/x]^{\overline{x}}) : w[y/x]^{\overline{x}}$
  Let $\overline{x} \notin L' = L$.
  $\overline{x} \notin L \Rightarrow (\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}}$
  $\overline{x} \cap \{y\} = \emptyset$
  $\wedge y \notin \texttt{names}(\Gamma : \texttt{let } \overline{t} \texttt{ in } t) \cup \texttt{names}((\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) : w^{\overline{y}})$
  $= \texttt{names}(\Gamma) \cup \texttt{fv}(\overline{t}) \cup \texttt{fv}(t) \cup \overline{y} \cup \overline{z} \cup \texttt{fv}(\overline{u}^{\overline{y}}) \cup \texttt{fv}(w^{\overline{y}})$
  $\overset{C1}{\Rightarrow} y \notin \texttt{names}(\Gamma) \cup \texttt{fv}(\overline{t}) \cup \texttt{fv}(t) \cup \overline{y} \cup \overline{z} \cup \texttt{fv}(\overline{u}) \cup \texttt{fv}(w)$
  $\overset{C2}{\Rightarrow} y \notin \texttt{names}(\Gamma) \cup \texttt{fv}(\overline{t}^{\overline{x}}) \cup \texttt{fv}(t^{\overline{x}}) \cup \overline{y} \cup \overline{z} \cup \texttt{fv}(\overline{u}^{\overline{x}}) \cup \texttt{fv}(w^{\overline{x}}) \cup \overline{x}$
  $\Rightarrow y \notin \texttt{names}((\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}}) \cup \texttt{names}((\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}})$.

  By induction hypothesis,
  $(\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}})[y/x] : (t^{\overline{x}})[y/x] \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}})[y/x] : (w^{\overline{x}})[y/x]$
  $\overset{x \cap \overline{x} = \emptyset}{\Rightarrow} (\Gamma[y/x], \overline{x} \mapsto \overline{t}[y/x]^{\overline{x}}) : t[y/x]^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z}[y/x] \mapsto \overline{u}[y/x]^{\overline{x}}) : w[y/x]^{\overline{x}}$.

CASE: $y \notin L$.
$\forall \overline{x} \notin L.(\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}}$,
$\Rightarrow \forall \overline{x} \notin L \cup \{y\}.(\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}}$.
Therefore we have now $y \in L \cup \{y\}$ and we are in the previous case.

□

6.8 Proof of Lemma 9 : LET_INTRO

Lemma 9
LET_INTRO $\quad (\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}} \wedge \texttt{fresh } \overline{x} \texttt{ in } (\Gamma : \texttt{let } \overline{t} \texttt{ in } t)$
$\Rightarrow \Gamma : \texttt{let } \overline{t} \texttt{ in } t \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}}$.

Proof. We have to find a finite set L such that $\overline{x} \notin L$ and
$\forall \overline{y} \notin L.(\Gamma, \overline{y} \mapsto \overline{t}^{\overline{y}}) : t^{\overline{y}} \Downarrow (\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) : w^{\overline{y}}$.
Consider $L' = \texttt{names}((\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}}) \cup \texttt{names}((\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}})$.
By hypothesis, $(\Gamma, \overline{x} \mapsto \overline{t}^{\overline{x}}) : t^{\overline{x}} \Downarrow (\overline{x} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{x}}) : w^{\overline{x}}$.
Applying Lemma 8, $\forall \overline{y} \notin L'.(\Gamma, \overline{y} \mapsto \overline{t}^{\overline{y}}) : t^{\overline{y}} \Downarrow (\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) : w^{\overline{y}}$.
Let $L = L' \backslash \{\overline{x}\}$.
Therefore, $\forall \overline{y} \notin L.(\Gamma, \overline{y} \mapsto \overline{t}^{\overline{y}}) : t^{\overline{y}} \Downarrow (\overline{y} +\!\!+ \overline{z} \mapsto \overline{u}^{\overline{y}}) : w^{\overline{y}}$.

□