Standardization and Confluence in Pure $\lambda$-Calculus
Formalized for the Matita Theorem Prover

Ferruccio Guidi
Department of Computer Science and Engineering
University of Bologna
Mura Anteo Zamboni 7, 40127, Bologna, ITALY
e-mail: fguidi@cs.unibo.it

We present a formalization of pure $\lambda$-calculus for the Matita interactive theorem prover, including
the proofs of two relevant results in reduction theory: the confluence theorem and the standard-
ization theorem. The proof of the latter is based on a new approach recently introduced by Xi
and refined by Kashima that, avoiding the notion of development and having a neat inductive
structure, is particularly suited for formalization in theorem provers.

1. INTRODUCTION

Standardization and confluence are two fundamental properties of $\beta$-reduction se-
quencies in pure $\lambda$-calculus [Bar85]. In order to state these properties, we will denote
$\lambda$-terms with the capital letters $A$, $B$, $C$, $D$, $M$, $N$, and will agree that by a com-
putation we mean a reduction sequence. In particular, a sequential computation is
such that a single redex is contracted in every step.

The standardization theorem asserts that if $M$ computes to $N$, then there is
computation from $M$ to $N$ which is standard in that redexes contracted in each
step are not “left” residuals of redexes contracted in the previous steps.

The confluence theorem, also known as the Church-Rosser property, asserts that
if $M$ computes to $N_1$ and to $N_2$, then there is an $N$ which both $N_1$ and $N_2$ compute
to. This is to say that $N_1$ and $N_2$ have a common reduct.

We remark that by “$\lambda$-calculus” we presently mean the $\lambda$K$\beta$-calculus.

Here we discuss how we formalized the proofs of these propositions for the Matita
interactive theorem prover [ARST11], with the final aim of providing a set of rele-
vant results on pure $\lambda$-calculus to the HELM digital library [APS+03].

Matita uses the Calculus of Inductive Constructions [CP90] as metalanguage, in
a variant similar to the one underlying the Coq proof management system [Coq12].
We tried to present things in a such a way that a limited knowledge of Matita’s
specification language, both in its syntactical and semantical aspects, is required.

As to confluence, we follow the proof of Tait and Matin-Löf based on parallel
reduction [Tak95], which has already been formalized in several logical frameworks
(see for instance [Pfe92]). This proof was chosen because it does not require to
extend the pure $\lambda$-calculus with marks for tracing redexes.

As to standardization, we follow essentially the proof of Xi [Xi99] in the partic-
ularly clear presentation made by Kashima [Kas00]. This novel approach entirely
avoids delicate notions such as residual or development and is entirely based on
structural induction on $\lambda$-terms and derivations, resulting particularly appealing
for formalization in interactive provers. Previous formal proofs of standardization, like [MP99], are sensibly more entangled than the one we shall describe here.

In order to define the notion of standard derivation, Kashima associates to each redex a natural number, termed its degree in [RP04], corresponding to its position in a “left-to-right” visit of the term. This system of reference is a bit inconvenient in the perspective of a formalization: firstly it requires an ancillary function to compute the number of redexes occurring in a term, secondly it forces to add at least two clauses to the definition of the relation “M reduces to N contracting one redex that is referred as n” with respect to the case in which n is not specified.

To solve these inconveniences, we adopt a system of reference by pointers that are paths in the tree representation of terms, for the specification of which, strings of boolean values suffice. Following the ideas of Kashima, pointers can be given an order reflecting the standard reduction order as defined by means of residuals, and that is equivalent to the “left-to-right” order induced by degrees.

Our exposition is organized as follows. Section 2 contains the part of our formalization that is not directly related to \( \lambda \)-calculus. As a formal system, pure \( \lambda \)-calculus has two components, which we discuss in Section 3 and Section 4. The structural component comprises the description of terms, with \( \alpha \)-conversion, and with substitution. The behavioral component includes reduction and computation. Conversion is not considered in this work. Our conclusions are in Section 5.

The formalization we are presenting in this article takes advantage of our long-time experience in the machine-checked specification of \( \lambda \)-calculus [Gui06] acquired by developing the formal system \( \lambda \delta \) [Gui09] since 2004.

2. BACKGROUND AND PRINCIPLES OF THE FORMALIZATION

Our formalization is for Matita 0.99.1 and depends on some notions and facts contained in the library provided with the prover. In particular we use logical connectives, existential quantifiers (the universal quantifier is in the metalanguage), Leibniz equality, general properties of relations especially concerning reflexive and transitive closures, arithmetics of natural numbers, and monomorphic lists.

In this section we discuss the additions we felt necessary in order to facilitate the formalization process. Some of them are now included in the library.

Of the many tactics provided by Matita’s proof engine, we use the following for our proofs: intro, apply, apply, elim, rewrite, cases, destruct, normalize, auto.

In particular, we manage to avoid generalize, and inversion. Our aversion for the latter tactic, comes from Coq 7, which generates oversize proof terms by inlining an inversion lemma in the proof at each invocation of the tactic. For instance, Lemma \( \text{pr}_0\_\text{confluence}^1 \) of [Gui06] contains 49 invocations of inversion producing a proof that, as a tree, has 46103 nodes and takes more than six minutes to validate on an AMD Athlon MP 1800+ at 1.5 GHz.

A newer and still unpublished version of the proof, produced without invoking inversion, is 29% smaller than the former and consists of just 32593 nodes.

Moreover the inversion of Coq does not invert some recursive relations properly since all inductive premises generated by the inversion procedure are removed from

---

1[^1]: <http://lambdadelta.info/static/matita/basic_1/pr0/pr0/pr0_confluence.con.body.html>.

---

the proof context. Nevertheless, this is not the expected behavior if one inverts the type judgment of a \( \lambda \)-calculus that includes the type conversion rule.

Matita’s inversion does not inline an inversion lemma and keeps the inductive premises in place, yet we prefer to encapsulate the inversion proof steps in inversion lemmas that follow the style of the well-known “inversion lemma” for Pure Type Systems \([\text{Bar93}]\) and that we prove by a combination of cases and destruct.

Furthermore, we hide types as much as possible following \([\text{Gui10}]\). For this reason we avoid the tactics that they take types as parameters.

We make a substantial use of auto to automate backward reasoning. Matita automation tactics takes a few optional parameters to prune the search space. The most relevant ones are depth and width: the first one is the maximum number of applicative nodes in a branch of the proof, while the second one is the maximum number of mutually dependent open goals along each branch.

Having experienced dramatic delays in the execution of auto due to the large span of the proofs search space even at low depths, some precautions must be adopted.

On one hand, most propositions that look problematic in view of a backward automatic synthesis of the proof should not be indexed for automation. These include some transitive-like properties of relations and the propositions that are designed to be applied in a forward reasoning manner as the inversion lemmas.

On the other hand we limit the search space depth by preferring flat \( n \)-ary logical constructions over nested ones of fixed arity. This is to say that the proposition:

\[
\exists x_1, \ldots, x_n. P_1(x_1, \ldots, x_n) \land \ldots \land P_m(x_1, \ldots, x_n)
\]

is formalized with a single multiple existential quantifier rather than with:

\[
\exists x_1. (\ldots (\exists x_n. (\ldots (P_1(x_1, \ldots, x_n) \land \ldots) \ldots) \land P_m(x_1, \ldots, x_n)) \ldots)
\]

that is, by means of the standard existential quantifier and binary conjunction.

In the first form, the proposition is atomized at depth 1, i.e. by the backward application of one constructor, whereas in the second form, the proposition is atomized at depth \( m + n - 1 \), i.e. by applying \( m + n - 1 \) constructors.

To this aim, we implemented an application that generates Matita’s scripts with the definitions of multiple existential quantifies for a given set of pairs \( (m, n) \).

We stress that the case \((6, 6)\) is realistic in the formalization of \( \lambda \)-calculus (see Lemma \( \text{pr0_gen_appl}^2 \) and Lemma \( \text{pr2_gen_appl}^3 \) of \([\text{Gui06}]\)).

The existential quantifier \((2, 1)\) is in Matita’s library. In this formalization we also use the instances: \((2, 2), (2, 3), (3, 2), (3, 3), (4, 3)\), and a ternary disjunction.

2.1 Logic and Arithmetics

We use the following property of the existential quantifier \((2, 1)\) in the proof of the confluence theorem. Our multiple existential quantifiers are denoted by \( \exists \).

\[\text{lemma ex2_commute: } \forall A_0. \forall P_0, P_1 : A_0 \to \text{Prop}. \]

\[\exists x_0. P_0 x_0 \land P_1 x_0 \to \exists x_0. P_1 x_0 \land P_0 x_0.\]

\(^2<\text{http://lambdadelta.info/static/matita/basic_1/pr0/fwd/pr0_gen_appl_con.html}>.

\(^3<\text{http://lambdadelta.info/static/matita/basic_1/pr2/fwd/pr2_gen_appl_con.html}>.

---

We decided to avoid the logical negation (\(\neg\)) and thus we activated the notation for the logical contradiction (\(\bot\)) that is not active in Matita’s library.

The following propositions use this notation and our machine-generated ternary disjunction connective (\(\lor\lor\)) in the context of natural arithmetics:

```plaintext
lemma lt_refl_false : \A. n < n \rightarrow \bot.
lemma lt_zero_false : \A. n < 0 \rightarrow \bot.
lemma plus_lt_false : \A. m + n < m \rightarrow \bot.
lemma lt_or_eq_or_gt : \A. \lor m < n | n = m | n < m.
```

We also use the trichotomy operator \(\text{tri}\) returning \(a_1\), \(a_2\), or \(a_3\) depending on \(n_1\) being less, equal, or greater than \(n_2\). This function could be defined on top of the library, but we believe that a direct definition is easier to deal with:

```plaintext
let rec tri (A:Type) n1 n2 a1 a2 a3 on n1 : A
def =
| O \Rightarrow \text{match} n2 with [ O \Rightarrow a2 | S n2 \Rightarrow a1 ]
| S n1 \Rightarrow \text{match} n2 with [ O \Rightarrow a3 | S n2 \Rightarrow \text{tri} A n1 n2 a1 a2 a3 ]
|.
```

```plaintext
lemma tri_lt : \A,a1,a2,a3,n2,n1. n1 < n2 \rightarrow \text{tri} A n1 n2 a1 a2 a3 = a1.
lemma tri_eq : \A,a1,a2,a3,n. \text{tri} A n n a1 a2 a3 = a2.
lemma tri_gt : \A,a1,a2,a3,n1,n2. n2 < n1 \rightarrow \text{tri} A n1 n2 a1 a2 a3 = a3.
```

Finally we need a general induction principle on a type \(A\) based on weight \(f\):

```plaintext
lemma f_ind_aux: \A. \forall A \rightarrow \text{nat}. \forall P:\text{predicate} A.
(\forall n. (\forall a. f a < n \rightarrow P a) \rightarrow \forall a. f a = n \rightarrow P a) \rightarrow \
\forall n,a. f a = n \rightarrow P a.
lemma f_ind: \A. \forall A \rightarrow \text{nat}. \forall P:\text{predicate} A.
(\forall n. (\forall a. f a < n \rightarrow P a) \rightarrow \forall a. f a = n \rightarrow P a) \rightarrow \forall a. P a.
```

### 2.2 Relations and Lists

The reflexive and transitive closure of a relation is in the library:

```plaintext
inductive star (A:Type) (R:relation A) (a:A): A \rightarrow Prop 
| sstep: \forall b,c. star A R a \rightarrow R b c \rightarrow star A R a c
| srefl : star A R a a.
```

and comes with some standard properties about it.

We added the eliminator \text{star_ind}\_l, that is not provided by Matita:

```plaintext
lemma star_ind_l: \forall A,R,a2. \forall P:\text{predicate} A.
P a2 \rightarrow 
(\forall a. R a a2 \rightarrow \text{star} \ldots R a a2 \rightarrow P a \rightarrow P a1) \rightarrow 
\forall a. \text{star} \ldots R a a2 \rightarrow P a1.
```

We also added a general notion of a confluent relation (see Subsection 4.4). We show, through the well-known “strip” lemma, that the reflexive and transitive closure of a confluent relation is confluent. This result will be used in Subsection 4.5.

**definition** singlevalued: \( \forall A,B. \) predicate (relation2 \( A B \) \( \stackrel{\text{def}}{=} \) \( \lambda A,B.R. \) \( \forall a,b1. \) \( R a b1 \rightarrow \forall b2. \) \( R a b2 \rightarrow b1 = b2. \))

**definition** confluent1: \( \forall A. \) relation \( A \rightarrow \) predicate \( A \) \( \stackrel{\text{def}}{=} \) \( \lambda A,R,a0. \) \( \forall a1. \) \( R a0 a1 \rightarrow \forall a2. \) \( R a0 a2 \rightarrow \exists a. \) \( R a1 a & R a2 a. \)

**definition** confluent: \( \forall A. \) predicate (relation \( A \)) \( \stackrel{\text{def}}{=} \) \( \lambda A,R. \) \( \forall a0. \) confluent1 \( \ldots R a0. \)

**lemma** star_strip: \( \forall A,R. \) confluent \( A R \rightarrow \forall a0,a1. \) star \( \ldots R a0 a1 \rightarrow \forall a2. \) \( R a0 a2 \rightarrow \exists a. \) \( R a1 a & \) star \( \ldots R a2 a. \)

**lemma** star_confluent: \( \forall A,R. \) confluent \( A R \rightarrow \) confluent \( A (\text{star} \ldots R). \)

A labeled sequential reduction (see Subsection 4.3) is a binary relation on terms taking an extra argument and needing a dedicated reflexive and transitive closure:

**inductive** lstar \((A: \text{Type}[0]) (B: \text{Type}[0]) (R: A \rightarrow \text{relation} \ B)\): list \( A \rightarrow \text{relation} \ B \) \( \stackrel{\text{def}}{=} \)

- \text{lstar\_nil} : \( \forall b. \) lstar \( A B R ([]) b b \)
- \text{lstar\_cons} : \( \forall a,b1,b. \) \( R a b1 b \rightarrow \forall l,b2. \) lstar \( A B R l b2 b \rightarrow \) lstar \( A B R (a::l) b1 b2 \)

**definition** ltransitive: \( \forall A,B: \text{Type}[0]. \) predicate (list \( A \rightarrow \text{relation} \ B) \) \( \stackrel{\text{def}}{=} \) \( \lambda A,B,R. \) \( \forall l1,b1,b. \) \( R l1 b1 b \rightarrow \forall l2,b2. \) \( R l2 b b2 \rightarrow \) \( R (l1@l2) b1 b2. \)

The notion \text{lstar} comes with the standard properties derived from those of \text{star}. In addition, we proved the following:

**lemma** lstar\_inv\_nil: \( \forall A,B,R,l,b1,b2. \) lstar \( A B R l b1 b2 \rightarrow [] = l \rightarrow b1 = b2. \)

**lemma** lstar\_inv\_cons: \( \forall A,B,R,l,b1,b2. \) lstar \( A B R l b1 b2 \rightarrow \forall a0,l0. \) \( a0::l0 = l \rightarrow \exists b. \) \( R a0 b1 b & \) lstar \( A B R l0 b b2. \)

**lemma** lstar\_inv\_step: \( \forall A,B,R,a,b1,b2. \) lstar \( A B R ([a]) b1 b2 \rightarrow R a b1 b2. \)

**lemma** lstar\_inv\_pos: \( \forall A,B,R,l,b1,b2. \) lstar \( A B R l b1 b2 \rightarrow 0 < |l| \rightarrow \exists a,l,l,b. \) \( a::ll = 1 \& R a b1 b & \) lstar \( A B R ll b b2. \)

Lemma \text{lstar\_inv\_pos} seems to be one of the deep grounds of the standardization theorem as we understand from Lemma \text{st\_lsred\_swap} of Subsection 4.7. We activated an alternative notation for the empty list (\( \langle \rangle \)), because the notation provided by the library (\( [ [ ] ] \)) clashes with our notation for substitution.
Finally, we added a variant of $\text{All}$ acting on a binary predicate (see Subsection 3.6) and a function $\text{map} \_\text{cons}$ whose application is denoted by $a :: l$:

\begin{verbatim}
let rec Allr (A:Type[0]) (R:relation A) (l : list A) on l : Prop =
match l with
| nil ⇒ True |
| cons a1 l ⇒ match l with | nil ⇒ True | cons a2 _ ⇒ R a1 a2 ∧ Allr A R l |

definition map_cons: ∀A. A → list ( list A) → list ( list A) def = λA,a.
map ... (cons ... a).
\end{verbatim}

3. THE STRUCTURAL COMPONENT OF PURE $\lambda$-CALCULUS

In this section we describe the structure of pure $\lambda$-terms and two related topics: the support for $\alpha$-conversion and for substitution. The last subsection is devoted to the location of subterms through paths on the tree representation of $\lambda$-terms.

All the material in this section is relatively standard, and we only include it to introduce the notation and for the sake of completeness.

3.1 Pure $\lambda$-Terms

Pure $\lambda$-terms are inductively generated from variable references ($\text{VRef}$), function formations ($\text{Abst}$), and function applications ($\text{Appl}$):

\begin{verbatim}
inductive term: Type[0] def =
| \text{VRef}: nat → term |
| \text{Abst}: term → term |
| \text{Appl}: term → term → term |
\end{verbatim}

Variable references occur by depth [dB94] for a convenient management of $\alpha$-conversion and reduction. In particular, with this encoding two $\alpha$-convertible terms are syntactically equal. Our depths, also known as “de Bruijn indexes”, are natural numbers starting at 0 rather than at 1. So all depths are legal and they will be typically ranged over by the metavariables $i$ and $j$.

The formation of the function with body $M$ is denoted by $\lambda . M$, in which the character $\lambda$ is typeset in upright face (contrast with the character $\lambda$).

The application of the function $M$ to the argument $N$ is denoted here by $@N.M$ according to the “reversed” notation suggested in [KN96], that, among other benefits, avoids precedence problems and keeps the matching abstraction-application pairs close together improving the visual understanding of redexes.

Then we define some notions of compatibility with term constructors and we prove that these notions are preserved by reflexive and transitive closures.

\begin{verbatim}
definition compatible_abst: predicate ( relation term) def = λR.

definition compatible_sn: predicate ( relation term) def = λR.
\end{verbatim}
definition compatible_dx: predicate (relation term) def \( \lambda R. \forall B,A1,A2. R A1 A2 \rightarrow R (@B.A1) (@B.A2). \)


lemma star_compatible_abst: \( \forall R. \text{compatible_abst } R \rightarrow \text{compatible_abst } (\text{star } \ldots R). \)

lemma star_compatible_sn: \( \forall R. \text{compatible_sn } R \rightarrow \text{compatible_sn } (\text{star } \ldots R). \)

lemma star_compatible_dx: \( \forall R. \text{compatible_dx } R \rightarrow \text{compatible_dx } (\text{star } \ldots R). \)

lemma star_compatible_appl: \( \forall R. \text{reflexive } ? R \rightarrow \text{compatible_appl } R \rightarrow \text{compatible_appl } (\text{star } \ldots R). \)

3.2 Relocation

The management of de Bruijn indexes requires the well-known function \( \text{lift} \), here denoted by \( \uparrow[d,h] \), that is related to the function \( \tau_h \) of [dB94]. Here and in the following \( d \) and \( e \) denote a level whereas \( h \) and \( k \) denote a relocation offset. The function \( \text{lift} \) extends to terms the function \( d\uparrow_h : \text{nat} \rightarrow \text{nat} \) acting on indexes:

\[
d\uparrow_h i \overset{\text{def}}{=} \begin{cases} 
  i & \text{if } i < d \\
  i + h & \text{if } i \geq d
\end{cases}
\]

By applying this function some indexes are increased by the offset \( h \) (lifting). The inverse operation, which decreases some indexes, is named delifting. Our theory of \( \text{lift} \) is taken from [Gui06] and can be traced back to [Hue94].

let rec lift h d M on M def = match M with
| VRef i ⇒ #(tri ... i d i (i + h) (i + h))
| Abst A ⇒ λ. (lift h (d+1) A)
| Appl B A ⇒ @(lift h d B). (lift h d A)
|).

lemma lift_vref_lt: \( \forall d,h,i. i < d \rightarrow \uparrow[d,h] #i = #i. \)

lemma lift_vref_ge: \( \forall d,h,i. d \leq i \rightarrow \uparrow[d,h] #i = #(i+h). \)

lemma lift_id: \( \forall M,d. \uparrow[d, 0] M = M. \)

We provide the following inversion lemmas for \( \text{lift} \):

lemma lift_inv_vref lt: \( \forall j,d. j < d \rightarrow \forall h,M. \uparrow[d, h] M = #j \rightarrow M = #j. \)

lemma lift_inv_vref ge: \( \forall j,d. d \leq j \rightarrow \forall h,M. \uparrow[d, h] M = #j \rightarrow d + h \leq j \land M = #(-j-h). \)

lemma lift_inv_vref be: \( \forall j,d,h. d \leq j \rightarrow j < d + h \rightarrow \forall M. \uparrow[d,h] M = #j \rightarrow \bot. \)
lemma lift_vref_ge_plus: \( \forall j, d, h. \ d + h \leq j \rightarrow \\forall M. \ \uparrow[d, h] M = \#j \rightarrow M = \#(j-h). \)

lemma lift_abst: \( \forall C, d, h, M. \ \uparrow[d, h] M = \lambda C \rightarrow \exists A. \ \uparrow[d+1, h] A = C \land M = \lambda A. \)

lemma lift аппl: \( \forall D, C, d, h, M. \ \uparrow[d, h] M = @D.C \rightarrow \exists A, B. \ \uparrow[d, h] B = D \land \uparrow[d, h] A = C \land M = @B.A. \)

Then we compute the compound relocation \( d_2 \uparrow^{h_2} d_1 \uparrow^{h_1} \) in the three cases: \( d_2 \leq d_1 \) ("less or equal"), \( d_1 \leq d_2 \leq d_1 + h_1 \) ("between"), \( d_1 + h_1 \leq d_2 \) ("greater or equal").

The proofs are by induction on the structure of \( M \).

The inverse propositions of the above are also provided.

The proofs are by induction on the structure of \( M_1 \).

In the end we define the notion of stability under the application of \textit{lift} and we prove that stability is preserved by reflexive and transitive closures.

definition liftable: predicate (relation term) \( \text{def} \lambda R. \ \forall h, M_1, M_2. \ R M_1 M_2 \rightarrow \forall d. \ (\uparrow[d, h] M_1) (\uparrow[d, h] M_2). \)

definition deliftable_sn: predicate (relation term) \( \text{def} \lambda R. \ \forall h, N_1, N_2. \ R N_1 N_2 \rightarrow \exists M. \ (\uparrow[d, h] M_1) M = N_2 \land \exists M. \ (\uparrow[d, h] M_2) M = N_1. \)

lemma star_liftable: \( \forall R. \ \text{liftable} R \rightarrow \text{liftable} (\text{star} \ldots R). \)
Standardization and Confluence

3.3 Delifting Substitution

When we operate with de Bruijn indexes, substituting $N$ in $M$ means replacing the occurrences of the first free variable in $M$ with $N$. The commonly accepted version of substitution we are presenting here, also delifts the occurrences of the other free variables in $M$. This practice makes the theory of substitution not so elegant, but removes the delifting operation from the definition of reduction.

We remark that both [dB94] and [Gui09] do not follow this practice.

The substitution of $N$ for the occurrences of the first free variable at level $d$ in $M$ ($\text{dsubst}$) is denoted here by $\lfloor d \uparrow N \rfloor M$, where the slanted arrow reminds that some delifting is performed. Other widespread notations include $M[N/d]$ and $M[d ← N]$. If the parameter $d$ is missing, the value 0 is implied for it.

As for $\text{lift}$, we take our theory of $\text{dsubst}$ from [Gui06] and [Hue94]. Given that $\text{dsubst}$ is not injective, its theory is much shorter than the one of $\text{lift}$.
Lemma \texttt{dsubst} is the well-known “substitution lemma”, which is proved here by induction on the structure of its first premise.

\begin{verbatim}
\textbf{theorem} dsubst_dsubst_ge: \forall D1,D2,M,d1,d2. d1 \leq d2 \rightarrow 
\quad [d2 \not\in D2] [d1 \not\in D1] M = 
\quad [d1 \not\in [d2 - d1 \not\in D2] D1] [d2 + 1 \not\in D2] M.
\end{verbatim}

\begin{verbatim}
\textbf{theorem} dsubst_dsubst_lt: \forall D1,D2,M,d1,d2. d2 < d1 \rightarrow 
\quad [d2 \not\in [d1 - d2 - 1 \not\in D1] D2] [d1 \not\in D1] M = 
\quad [d1 - 1 \not\in D1] [d2 \not\in D2] M.
\end{verbatim}

The theory ends with some preservation results about the notion of stability under substitution in the spirit of Subsection 3.2.

\begin{verbatim}
\textbf{definition} dsubstable_dx: \text{predicate (relation term)} \defeq \lambda R. 
\quad \forall D,M1,M2. R M1 M2 \rightarrow \forall d. R ([d \not\in D] M1) ([d \not\in D] M2).
\end{verbatim}

\begin{verbatim}
\textbf{definition} dsubstable: \text{predicate (relation term)} \defeq \lambda R. 
\quad \forall D1,D2. R D1 D2 \rightarrow \forall M1,M2. R M1 M2 \rightarrow 
\quad \forall d. R ([d \not\in D1] M1) ([d \not\in D2] M2).
\end{verbatim}

\begin{verbatim}
\textbf{lemma} star_dsubstable_dx: \forall R. dsubstable_dx R \rightarrow dsubstable_dx (star \ldots R).
\end{verbatim}

\begin{verbatim}
\textbf{lemma} lstar_dsubstable_dx: \forall T,R. (\forall t. dsubstable_dx (R t)) \rightarrow 
\quad \forall l. dsubstable_dx (lstar T \ldots R l).
\end{verbatim}

\begin{verbatim}
\textbf{lemma} star_dsubstable: \forall R. reflexive ? R \rightarrow 
\quad dsubstable R \rightarrow dsubstable (star \ldots R).
\end{verbatim}

3.4 Size

In this formalization, the \textit{size} of a term \(M\), denoted by \(|M|\), is the number of inner nodes in the tree representation of \(M\). In the literature (see for instance \cite{Xi99}) the “size” of \(M\) may refer to the total numbers of nodes in \(M\) as well.

We need this notion in the present work to set up a well-founded induction in the proof of the so-called “diamond property” (see Subsection 4.4).

The proof of the lemma is by induction on the structure of \(M\).

\begin{verbatim}
\textbf{let rec} size M on M \defeq \text{match M with} 
\quad [ VRef i \Rightarrow 0 ] 
\quad | Abst A \Rightarrow \text{size} A + 1 
\quad | Appl B A \Rightarrow (\text{size} B) + (\text{size} A) + 1 
\quad ].
\end{verbatim}

3.5 Pointers to Subterms

In this formalization we wish to qualify a subterm \(N\) of a term \(M\) with the path that connects the root of \(M\) to the root of \(N\) in the tree representation of \(M\). We
will refer to such a path as to the \textit{pointer} to \( N \). Clearly, the case of \( N \) being a redex is of particular importance in the following sections.

A pointer, ranged over by the metavariables \( p \) and \( q \), is a list of steps and \( c \) will be a metavariable ranging over steps. We recognize three steps named \( rc \) ("rectum" or "straight"), \( sn \) ("sinister" or "left") and \( dx \) ("dexter" or "right"). We note that \( rc \) and \( sn \) can be identified as long as head reductions [Bar85] are not considered.

The subterm of \( M \) pointed by a step and by a list of steps is defined as follows:

1. \( rc \) points to \( A \) in \( \lambda A \);
2. \( sn \) points to \( B \) and \( dx \) points to \( A \) in \( @B.A \);
3. \( ♦ \) points to \( M \) in \( M \) itself;
4. if \( c \) points to \( N \) in \( M \) and \( p \) points to \( N \) in \( N \), then \( c :: p \) points to \( N \) in \( M \).

A term \( M \) with a pointer in bijection with an approximation [RP04] of \( M \) with a single instance of the term \( Ω \).

Steps (\texttt{ptr\_step}) and pointers (\texttt{ptr}) are formally defined in the following lines with the notions of "compatibility with \( λ\)-constructions" concerning relations.

\begin{verbatim}
inductive ptr_step: Type[0] def |
| rc: ptr_step |
| sn: ptr_step |
| dx: ptr_step |

definition ptr: Type[0] def list ptr_step.


\end{verbatim}

The theory ends with the definition of the predicate \texttt{in\_whd} selecting the pointers made just of \( dx \) steps, which locate the "weak head redex" of a term. An elimination principle for reasoning about these pointers is provided.

Its proof is by induction on the structure of \( p \).

\begin{verbatim}
definition is_dx: predicate ptr\_step def λc. dx = c.

definition in_whd: predicate ptr def All ... is_dx.

\end{verbatim}

3.6 Lists of Pointers to Subterms

In this formalization we use lists of pointers to trace the redexes contracted in finite sequential computations. Thus a computation is as long as its list of pointers.

Lists of pointers, ranged over by the metavariables $r$ and $s$, are defined in the following lines and come with compatibility notions and results (see Subsection 3.5).

\begin{definition}
\text{ptrl : Type[0]} \overset{\text{def}}{=} \text{list ptr}.
\end{definition}

\begin{definition}
\text{ho_compatible_rc : predicate (ptrl \to relation term)} \overset{\text{def}}{=} \lambda R. \\
\forall s, A_1, A_2. R s A_1 A_2 \to R (rc ::: s) (\lambda A_1) (\lambda A_2).
\end{definition}

\begin{definition}
\text{ho_compatible_sn : predicate (ptrl \to relation term)} \overset{\text{def}}{=} \lambda R. \\
\forall s, B_1, B_2, A. R s B_1 B_2 \to R (sn ::: s) (@B_1.A) (@B_2.A).
\end{definition}

\begin{definition}
\text{ho_compatible_dx : predicate (ptrl \to relation term)} \overset{\text{def}}{=} \lambda R. \\
\forall s, B, A_1, A_2. R s A_1 A_2 \to R (dx ::: s) (@B.A_1) (@B.A_2).
\end{definition}

\begin{lemma}
\text{lstar_compatible_rc :} \forall R. \text{compatible_rc R \to ho_compatible_rc (lstar \ldots R)}.
\end{lemma}

\begin{lemma}
\text{lstar_compatible_sn :} \forall R. \text{compatible_sn R \to ho_compatible_sn (lstar \ldots R)}.
\end{lemma}

\begin{lemma}
\text{lstar_compatible_dx :} \forall R. \text{compatible_dx R \to ho_compatible_dx (lstar \ldots R)}.
\end{lemma}

The predicate $\text{is_vhd}$ selects the lists of $\text{in_vhd}$ pointers, which we name $\text{vhd}$ lists. These lists are meant to trace weak head reduction sequences [Pey87].

The two proofs are by induction on the structure of $s$ and $r$ respectively.

\begin{definition}
\text{is_vhd : predicate ptrl} \overset{\text{def}}{=} \text{All \ldots in_vhd}.
\end{definition}

\begin{lemma}
\text{is_vhd_dx :} \forall s. \text{is_vhd s \to is_vhd (dx ::: s)}.
\end{lemma}

\begin{lemma}
\text{is_vhd_append :} \forall s. \text{is_vhd r \to \forall s. is_vhd s \to is_vhd (r@s)}.
\end{lemma}

4. THE BEHAVIORAL COMPONENT OF PURE $\lambda$-CALCULUS

In this section we formalize $\beta$-reduction and the related notion of $\beta$-reduction sequence (or $\beta$-computation) both in the "labeled sequential" and "parallel" forms.

The section starts by introducing an order on pointers modeling the "leftmost outermost" (or "left-to-right") reduction strategy for the pointed redexes.

4.1 Standard Order on Pointers to Subterms

The order on pointers we will define in this section is meant to model the order in which the pointed redexes are contracted in the "leftmost outermost" (or "left-to-right") reduction strategy underlying the standardization theorem.

This order, which is our alternative to the order of [Kas00], is based on a precedence relation $\prec$ (pprec) defined in the following lines.

The meaning of $p \prec q$ is: first contract the redex pointed by $p$ in a given term $M$ and then the redex pointed by $q$ in the contractum of $M$. 

---

We justify our definition as follows. With \texttt{pprec Nil} we state that if \(M\) is itself a redex, it must be contracted before other redexes. With \texttt{pprec R} we state that redexes in functions must be contracted before redexes in function bodies. With \texttt{pprec S} we state that redexes in function bodies must be contracted before redexes in function arguments. With \texttt{pprec Comp} we state that the order must be preserved when contracting inner redexes. With \texttt{pprec Skip} we state that contracting a redex in a function may produce a redex to be contracted afterwards.

The \textit{standard order} \(\leq\) (\texttt{ple}) is defined as the reflexive and transitive closure of the precedence relation, and comes with standard properties.

\begin{definition}
\texttt{ple}: relation \(\text{ptr}\) \(\overset{\text{def}}{=} \text{star} \ldots \text{pprec}.
\end{definition}

\texttt{lemma ple_step_Rc}: \(\forall p, q. p < q \rightarrow p \leq q\).

\texttt{lemma ple_step_Sn}: \(\forall p1, p, p2. p1 < p \rightarrow p \leq p2 \rightarrow p1 \leq p2\).

\texttt{lemma ple_Rc}: \(\forall p, q. \text{dx}:p \leq \text{rc}:q\).

\texttt{lemma ple_Sn}: \(\forall p, q. \text{rc}:p \leq \text{sn}:q\).

\texttt{lemma ple_Skip}: \(\text{dx}::\Diamond \leq \Diamond\).

\texttt{lemma ple_Nil}: \(\forall p. \Diamond \leq p\).

\texttt{lemma ple_Comp}: \(\forall p1, p2. p1 \leq p2 \rightarrow \forall c. (c :: p1) \leq (c :: p2)\).

\texttt{lemma ple_Skip_ple}: \(\forall p. p \leq \Diamond \rightarrow \text{dx}:p \leq \Diamond\).

\texttt{theorem ple_trans}: transitive \ldots \texttt{ple}.

\texttt{lemma ple_cons}: \(\forall p, q. \text{dx}:p \leq \text{sn}:q\).

\texttt{lemma ple_dichotomy}: \(\forall p1, p2: \text{ptr}. p1 \leq p2 \lor p2 \leq p1\).

\texttt{lemma ple_inv_sn}: \(\forall p, q. p \leq q \rightarrow \forall p0. \text{sn}:p0 = p \rightarrow \exists q0. p0 \leq q0 \& \text{sn}:q0 = q\).
This statement characterizes the in_whd pointers in terms of the order:

The following are equivalent for \( p \): \( \text{in}_\text{whd} p; \ p \leq \diamond; \ \forall q. \ p \leq q. \)

The proofs are by induction on the structure of the first premise.

\[
\begin{align*}
\text{lemma } & \text{pprec_fwd_in_whd}: \forall p, q. \ p \prec q \rightarrow \text{in}_\text{whd} q \rightarrow \text{in}_\text{whd} p. \\
\text{lemma } & \text{in_whd_ple-nil}: \forall p. \ \text{in}_\text{whd} p \rightarrow p \leq \diamond. \\
\text{theorem } & \text{in_whd_ple}: \forall p. \ \text{in}_\text{whd} p \rightarrow \forall q. \ p \leq q. \\
\text{lemma } & \text{ple-nil_inv_in_whd}: \forall p. \ (\forall q. \ p \leq q) \rightarrow \text{in}_\text{whd} p. \\
\text{lemma } & \text{ple_inv_in_whd}: \forall p. \ (\forall q. \ p \leq q) \rightarrow \text{in}_\text{whd} p.
\end{align*}
\]

The fact that the standard order is cyclic, in that \( \diamond \prec dx :: \diamond \prec \diamond \), is not surprising and is due to the existence of cyclic computations in \( \lambda \)-calculus.

One can derive from this chain of inequalities that the transitive closure of the precedence relation is reflexive (just remember Clause \text{pprec_comp}).

### 4.2 Standard Lists of Pointers to Subterms

The predicate \text{is_standard} selects the lists of pointers occurring in non-decreasing standard order, which we name standard lists. According to the justification of the standard precedence relation we gave in Subsection 4.1, it is plausible that these lists trace standard reduction sequences. As one expects, Lemma \text{is_whd_is_standard} asserts that a whd list is standard as is a weak head reduction sequence.

The proofs are by induction on the structure of \( r \) or \( s \).

\[
\begin{align*}
\text{definition } & \text{is_standard}: \text{predicate ptrl} \overset{\text{def}}{=} \text{Allr } \ldots \text{ ple.} \\
\text{lemma } & \text{is_standard_compatible}: \forall c, s. \ \text{is_standard} s \rightarrow \text{is_standard} (c :: s). \\
\text{lemma } & \text{is_standard_cons}: \forall p, s. \ \text{is_standard} s \rightarrow \text{is_standard} ((dx :: p) :: sn :: s). \\
\text{lemma } & \text{is_standard_append}: \forall r. \ \text{is_standard} r \rightarrow \forall s. \ \text{is_standard} s \rightarrow \text{is_standard} ((dx :: r) @ sn :: s). \\
\text{theorem } & \text{is_whd_is_standard}: \forall s. \ \text{is_whd} s \rightarrow \text{is_standard} s. \\
\text{lemma } & \text{is_standard_in_whd}: \forall p. \ \text{in_whd} p \rightarrow \forall s. \ \text{is_standard} s \rightarrow \text{is_standard} (p :: s). \\
\text{theorem } & \text{is_whd_is_standard_trans}: \forall r. \ \text{is_whd} r \rightarrow \forall s. \ \text{is_standard} s \rightarrow \text{is_standard} (r @ s). 
\end{align*}
\]

We justify informally our notion of standard list in terms of residuals as follows. Take a term \( C \) with a two redexes \( M \) and \( N \) pointed by \( p \) and \( q \) respectively. If \( M \) is a residual at the “left” of \( N \), then reducing \( M \) after \( N \) is not standard and we argue that \( q \leq p \) is forbidden by the order in the most relevant cases.

If \( C \equiv M \equiv \beta_f B \cdot \lambda A.A \), then \( N \) is a subterm of \( B \) or of \( A \), which is to say that \( q \) has the form \( sn :: q_0 \) or \( dx :: rc :: q_0 \). Now \( q \leq p \) and \( p \equiv \diamond \) imply \text{in_whd} q.
Now take $M$ and $N$ in the arguments of $C \equiv \@D.\@B.A$, so $M$ is a subterm of $B$ and $N$ is a subterm of $D$ (remember that $C$ is $(A B D)$ in standard notation). So $q$ has the form $\text{sn} :: q_0$ and $p$ has the form $\text{dx} :: \text{sn} :: p_0$ in contrast with Lemma $\text{ple\_inv\_sn}$ by which $q \leq p$ yields that $p$ must have the form $\text{sn} :: p_1$.

4.3 Labeled Sequential Reduction

The advantage of our path-based redex pointers over the degree-based ones used in [Kas00], clearly appears when (finally!) we come to define a step of sequential reduction labeled with the pointer to the redex we are contracting.

This relation ($\text{lsred}$), which we denote with $M \Rightarrow [p] N$, comes with standard properties such as stability under relocation and substitution. In addition, some inversion lemmas prove that the relation is single-valued for given $p$ and $M$.

\begin{verbatim}
| inductive lsred: ptr \rightarrow \text{relation\_term} | def |
| lsred_beta : \forall B,A. lsred (\hat{)} (@B.A.A) ([\hat{B}].A) |
| lsred_abst : \forall p,A1,A2. lsred p A1 A2 \rightarrow lsred (rc :: p) (\lambda.A1) (\lambda.A2) |
| lsred_appl_dx : \forall p,B,A1,A2. lsred p A1 A2 \rightarrow lsred (dx :: p) (@B.A1) (@B.A2) |


| lemma lsred_inv_vref: \forall p,M,N. M \Rightarrow [p] N \rightarrow \forall i. #i = M \rightarrow \bot. |
| lemma lsred_inv_nil: \forall p,M,N. M \Rightarrow [p] N \rightarrow \forall q = p \rightarrow \exists B,A. @B. \lambda.A = M \& [\hat{B}].A = N. |
| lemma lsred_inv_rc: \forall p,M,N. M \Rightarrow [p] N \rightarrow \forall q, rc::q = p \rightarrow \exists A1,A2. A1 \Rightarrow [q] A2 \& \lambda.A1 = M \& \lambda.A2 = N. |
| lemma lsred_inv_sn: \forall p,M,N. M \Rightarrow [p] N \rightarrow \forall q, sn::q = p \rightarrow \exists B1,B2,A. B1 \Rightarrow [q] B2 \& @B1.A = M \& @B2.A = N. |
| lemma lsred_inv_dx: \forall p,M,N. M \Rightarrow [p] N \rightarrow \forall q, dx::q = p \rightarrow \exists B,A1,A2. A1 \Rightarrow [q] A2 \& @B.A1 = M \& @B.A2 = N. |
| lemma lsred_lift: \forall p. liftable (lsred p). |
| lemma lsred_inv_lifth: \forall p. deliftable_sn (lsred p). |
| lemma lsred_dsubst: \forall p. dsubstable_dx (lsred p). |
| theorem lsredMono: \forall p. singlevalued \ldots (lsred p). |
\end{verbatim}

4.4 Parallel Reduction

The purpose of parallel reduction in this formalization is to support the confluence of sequential computation by providing its well-known “diamond” property.

One step of parallel reduction ($\text{pred}$), which we denote with $M \Rightarrow N$, contracts an arbitrary set of redexes that do not need to be traced for our present aims.

This relation comes with standard properties such as reflexivity and stability under relocation and substitution. Some inversion lemmas are provided as well.
The confluence of parallel reduction (also known as its “diamond” property), stating that $M_0 \Rightarrow M_1$ and $M_0 \Rightarrow M_2$ imply $M_1 \Rightarrow M$ and $M_2 \Rightarrow M$ for some $M$, is proved by induction on the size of $M_0$. The induction generates six cases, three of which are proved as separate lemmas because one of them is invoked twice.

In this way we take advantage of the symmetry existing between $M_1$ and $M_2$. 

```plaintext
lemma pred_conf1_vref: ∀i. confluent1 ... pred (#i).
#i #M1 #H1 #M2 #H2
<(pred_inv_vref ... H1) − H1 [3: // 2: skip ]
<(pred_inv_vref ... H2) − H2 [3: // 2: skip ]
/2 width=3/
qed−.

lemma pred_conf1_abst: ∀A. confluent1 ... pred A → confluent1 ... pred (λ.A).
#A #IH #M1 #H1 #M2 #H2
elim (pred_inv_abst ... H1 ??) − H1 [3: // 2: skip ] #A1 #HA1 #H destruct
elim (pred_inv_abst ... H2 ??) − H2 [3: // 2: skip ] #A2 #HA2 #H destruct
elim (IH ... HA1 ... HA2) − A /3 width=3/
qed−.
```
lemma pred_conf1_appl_beta: \( \forall B, B_1, B_2, C, C_1, M_1. \)
\[
(\forall M_0. \ |M_0| < |B|+|\lambda. C|+1 \rightarrow \text{confluent1 ? pred} M_0) \rightarrow
\]
\[
B \Rightarrow B_1 \rightarrow B \Rightarrow B_2 \rightarrow \lambda. C \Rightarrow M_1 \rightarrow C \Rightarrow C_2 \rightarrow
\]
\[
\exists M. A_B, M_1 \Rightarrow M & \|B_2 \|C_2 \Rightarrow M.
\]
\[
#B #B_1 #B_2 #C #C_1 #M_1 #IH #HB_1 #HB_2 #H_1 #HC_2
\]
\[
\text{elim (pred_inv_abst ... H1 ??) } - H_1 [3: \|2: \text{skip } ] #C_1 #HC_1 #H \text{ destruct}
\]
\[
\text{elim (IH B ... HB_1 ... HB_2)} - HB_1 - HB_2 //
\]
\[
\text{elim (IH C ... HC_1 ... HC_2) normalize } /\!\!/ - B - C /3 \text{ width}=5/
\]
\[
\text{qed } .
\]

theorem pred_conf: confluent ... pred.
\[
#M @((\text{ind} \ldots \text{length} \ldots M) - M \#n \#IH * \text{normalize}
\]
\[
| /2 \text{ width}=3 \text{ by pred_conf1_vref}/
\]
\[
| /3 \text{ width}=4 \text{ by pred_conf1_abst}/
\]
\[
| #B #A #H #M_1 #H_1 #M_2 #H_2 \text{ destruct}
\]
\[
\text{elim (pred_inv_appl ... H1 ??) } - H_1 [5: \|2,3: \text{skip } ] *
\]
\[
\text{elim (pred_inv_appl ... H2 ??) } - H_2 [5,10: \|2,3,7,8: \text{skip } ] *
\]
\[
| #B_2 #A_2 #HB_2 #HA_2 #B_1 #A_1 #HB_1 #HA_1 #H_1 \text{ destruct}
\]
\[
\text{elim (IH A ... HA_1 ... HA_2)} - HA_1 - HA_2 //
\]
\[
\text{elim (IH B ... HB_1 ... HB_2)} - A - B /3 \text{ width}=5/
\]
\[
| #C #B_2 #C_2 #HC_2 #H_2 #HM_2 #B_1 #N #HB_1 #H #HM_1 \text{ destruct}
\]
\[
\text{elim (IH B ... HB_1 ... HB_2) - HB_1 - HB_2 //}
\]
\[
\text{elim (IH C ... HC_1 ... HC_2) normalize } /\!\!/ - B - C /3 \text{ width}=5/
\]
\[
\]
\[
\text{qed } .
\]

4.5 Parallel Computation

Parallel computation (preds) from \( M \) to \( N \) is denoted by \( M \Rightarrow^* N \) and is defined as the reflexive and transitive closure of parallel reduction. This notion comes with standard properties including confluence and the well-known “strip” lemma.

All proofs are immediate corollaries of general results on relations.

definition preds: relation term \( \Rightarrow^* \) star \ldots pred.

lemma preds_refl: reflexive \ldots preds.

lemma preds_step_an: \( \forall M_1, M. M_1 \Rightarrow M \rightarrow \forall M_2. M \Rightarrow^* M_2 \rightarrow M_1 \Rightarrow^* M_2. \)

lemma preds_step_dx: \( \forall M_1, M. M \Rightarrow^* M \rightarrow \forall M_2. M \Rightarrow M_2 \rightarrow M_1 \Rightarrow^* M_2. \)

lemma preds_step_rc: \( \forall M_1, M_2. M_1 \Rightarrow M_2 \rightarrow \forall M \Rightarrow M_1 \Rightarrow^* M_2. \)
4.6 Labeled Sequential Computation

Labeled sequential computation (lsreds) from $M$ to $N$ along the redexes pointed by the elements of $s$, is denoted here by $M \xrightarrow{*}[s] N$. This notion is defined as the reflexive and transitive closure of labeled sequential reduction and comes with standard properties including stability under relocation and substitution.

Most proofs are immediate corollaries of general results on relations.

**definition** lsreds : ptrl → relation term def = lstar ... lsred.

**lemma** lsreds_refl: reflexive ... (lsreds ($\Diamond$)).

**lemma** lsreds_step_sn: $\forall p,M1,M. M1 \xrightarrow{[p]} M \xrightarrow{\forall s,M2} M \xrightarrow{*[s]} M2 \xrightarrow{M1 \xrightarrow{*[p:s]} M2}.$

**lemma** lsreds_step_dx: $\forall s,M1,M. M1 \xrightarrow{*[s]} M \xrightarrow{\forall p,M2} M \xrightarrow{*[p]} M2 \xrightarrow{M1 \xrightarrow{*[s@p::\Diamond]} M2}.$

**lemma** lsreds_step_rc: $\forall p,M1,M2. M1 \xrightarrow{[p]} M2 \xrightarrow{M1 \xrightarrow{*[p::\Diamond]} M2}.$

**lemma** lsreds_inv_nil: $\forall s,M1,M2. M1 \xrightarrow{*[s]} M2 \xrightarrow{\Diamond = s \rightarrow M1 = M2}.$

**lemma** lsreds_inv_cons: $\forall s,M1,M2. M1 \xrightarrow{*[s]} M2 \xrightarrow{\forall q,r. q:r = s \rightarrow \exists M. M1 \xrightarrow{[q]} M & M \xrightarrow{*[r]} M2}.$

**lemma** lsreds_inv_step_rc: $\forall p,M1,M2. M1 \xrightarrow{*[p::\Diamond]} M2 \xrightarrow{M1 \xrightarrow{*[p]} M2}.$
The equivalence between labeled sequential reduction and parallel reduction, in that $M \Rightarrow^* N$ iff $\exists s. M \Rightarrow^*[s] N$, takes to the Confluence Theorem $\text{lsreds} \text{conf}$.

The first three proofs are by induction on the structure of the first premise.

Weak head computations play a fundamental role in the proof of the standardization theorem (Subsection 4.7), but cannot be characterized in terms of degree-based pointers. Therefore Kashima must define a dedicated relation for them, that he names “hap” in [Kas00], and that comes with its own theory. On the contrary, our path-based pointers allow to avoid this device since we easily characterize a weak head computation form $M$ to $N$ as $M \Rightarrow^*[s] N$ under the assumption $\text{is\_whd\_s}$.

We want to stress that our Lemma $\text{in\_whd\_ind}$ (Subsection 3.5) allows us to reason about weak head computations without generating more proof cases than Kashima does reasoning about his “hap” computation.
4.7 Decomposed Standard Computation

This section is devoted to the proof of the standardization theorem using an inductively defined decomposition of a standard computation from $M$ to $N$ as weak head computation followed by an inner computation. To this aim, Kashima defines a computation, denoted here by $M \xrightarrow{\circ} N$, that he names “st” in [Kas00].

Our version of $st$ comes with standard properties that include three inversion lemmas. Transitivity is proved after standardization rather than directly.

\[
\text{inductive } st: \text{ relation term } \defeq
\]

| st_vref : $\forall s,M,i. \text{ is\_whd } s \rightarrow M \xrightarrow{[s]} \#i \rightarrow st M (\#i)$ |
| st_abst : $\forall s,M,A1,A2. \text{ is\_whd } s \rightarrow M \xrightarrow{[s]} \lambda A1 \rightarrow st A1 A2 \rightarrow st M (\lambda A2)$ |
| st_appl : $\forall s,M,B1,B2,A1,A2. \text{ is\_whd } s \rightarrow M \xrightarrow{[s]} \Lambda B1.A1 \rightarrow st B1 B2 \rightarrow st A1 A2 \rightarrow st M (@B2.A2)$ |

\[
\text{lemma st_invref: } \forall M,N. M \xrightarrow{\circ} N \rightarrow \forall j. \#j = N \rightarrow \\
\exists s, \text{ is\_whd } s & M \xrightarrow{[s]} \#j. \\
\]

\[
\text{lemma st_invabst: } \forall M,N. M \xrightarrow{\circ} N \rightarrow \forall C2. \lambda C2 = N \rightarrow \\
\exists s,C1. \text{ is\_whd } s & M \xrightarrow{[s]} \lambda C1 & C1 s \xrightarrow{\circ} C2. \\
\]

\[
\text{lemma st_invappl: } \forall M,N. M \xrightarrow{\circ} N \rightarrow \forall D2,C2. @D2.C2 = N \rightarrow \\
\exists s,D1,C1. \text{ is\_whd } s & M \xrightarrow{[s]} @D1.C1 & D1 s \xrightarrow{\circ} D2 & C1 s \xrightarrow{\circ} C2. \\
\]

\[
\text{lemma st_ref: reflexive } st. \\
\]

\[
\text{lemma st_step_sn: } \forall N1,N2. N1 \xrightarrow{\circ} N2 \rightarrow \\
\forall s,M. \text{ is\_whd } s \rightarrow M \xrightarrow{[s]} N1 \rightarrow M \xrightarrow{\circ} N2. \\
\]

\[
\text{lemma st_step_rc: } \forall s,M1,M2. \text{ is\_whd } s \rightarrow M1 \xrightarrow{[s]} M2 \rightarrow M1 \xrightarrow{\circ} M2. \\
\]

\[
\text{lemma st_lift: liftable } st. \\
\]

\[
\text{lemma st_invlift: deliftable } st. \\
\]

\[
\text{lemma st_dsubst: dsubstable } st. \\
\]

The Standardization Theorem $\text{lsreds_standard}$ is proved in two steps. Firstly we prove that $M \xrightarrow{\circ} [s] N$ implies $M \xrightarrow{\circ} N$ (Lemma $st\_lsreds$) by induction on $s$. Lemma $st\_step\_sx$ (Lemma 3.6 of [Kas00]) is the inductive case of the proof. Secondly we prove that $M \xrightarrow{\circ} N$ implies $M \xrightarrow{\circ} [r] N$ such that $\text{is\_standard } r$ (Lemma $st\_inv\_lsreds\_is\_standard$) by induction on the structure of the premise.

We show the main proofs in full. The reader may note that the first proof case of Lemma $st\_step\_sx$ is Lemma 3.5 of [Kas00].
lemma st_step_dx: \forall p, M, M_2. M \to^p [p] M_2 \to^\forall M_1. M \to^\forall M_1 \to^\forall M_2.

\[ p \; \# M \; \# M_2 \; \# H \; \text{elim} \; H \to p \; \to M \; \to M_2. \]

\[ \forall M, M_1. M \to^\forall \to M_1 \to^\forall M. \]

\[ \text{elim} \; (\text{st_inv_appl} \; \ldots \; H ) \implies -H \; [4: // \; 2,3: \; \text{skip} ] \]

\[ \# s \; \# B_1 \; # M_3 \; # H_M_1 \; # H_B_1 \; # H \]

\[ \text{elim} \; (\text{st_inv_abst} \; \ldots \; H ) \implies -H \; [3: // \; 2: \; \text{skip} \; \# r \; \# A_1 \; \# H_r \; \# H_A_1 \]

\[ \text{lapply} \; (\text{lrsreds_trans} \; \ldots \; H ) \implies \text{HM_1} \; \to^\forall \; (\text{dx::r} \; @ (B_1.A_1) \; ) \; [2 \; \text{width}=1/ \; -M \; # H_M_1 \]

\[ \text{lapply} \; (\text{lrsreds_step_dx} \; \ldots \; H ) \implies \text{HM_1} \; \to^\forall \; (\text{dx::r} \; (\lambda \; A_1) \; ) \; [4: // \; 2,3: \; \text{skip} ] \quad \text{HM_1} \]

\[ @ (\text{st_step_sn} \; \ldots \; \text{HM_1}) /2 \; \text{width}=1/ /4 \; \text{width}=1/ \]

\[ \text{lapply} \; (\text{lrsreds_trans} \; \ldots \; H ) \implies -H \; [4: // \; 2,3: \; \text{skip} \; /3 \; \text{width}=7/ \]

\[ | \quad \text{elim} \; (\text{st_inv_abst} \; \ldots \; H ) \implies -H \; [3: // \; 2: \; \text{skip} \; /3 \; \text{width}=5/ \]

\[ | \quad \text{elim} \; (\text{st_inv_appl} \; \ldots \; H ) \implies -H \; [4: // \; 2,3: \; \text{skip} \; /3 \; \text{width}=7/ \]

\[ ]

\[ \text{qed} \; . \]

lemma st_lrsreds: \forall s, M_1, M_2. M_1 \to^* [s] M_2 \to M_1 \to^\forall M_2.

lemma st_inv_lrsreds_is_standard: \forall M, N. M \to^\forall \to N \to \exists r. M \to^\forall \to^* [r] N \land \text{is_standard} \; r.

\[ \# M \; \# N \; \# H \; \text{elim} \; H \to -M \; -N \]

\[ \# s \; \# M \; \# i \; # H_M \]

\[ \text{lapply} \; (\text{is_whd_is_standard} \; \ldots \; H_s) \; H_s \; [2 \; \text{width}=3/ \]

\[ | \quad \# s \; \# M \; \# A_1 \; \# A_2 \; \# H_M \; \# H_A_1 \; \# H_r \]

\[ \text{lapply} \; (\text{lrsreds_trans} \; \ldots \; \text{HM} \; (\text{rec::r}) \; (\lambda \; A_2) \; ) \; [2 \; \text{width}=1/ \; -A_1 \; # H_M \]

\[ @ (\text{ex2_intro} \; \ldots \; \text{HM}) \; -M \; -A_2 \; /3 \; \text{width}=1/ \]

\[ | \quad \# s \; \# M \; \# B_1 \; \# B_2 \; \# A_1 \; \# A_2 \; \# H_s \; \# H_A_1 \; \# H_r \]

\[ * \; # r_b \; # H_B_1 \; # H_r \; * \; # r_a \; # H_A_1 \; # H_r \]

\[ \text{lapply} \; (\text{lrsreds_trans} \; \ldots \; \text{HM} \; (\text{dx::r}) \; (\text{B_1.A_2}) \; ) \; [2 \; \text{width}=1/ \; -A_1 \; # H_M \]

\[ \text{lapply} \; (\text{lrsreds_trans} \; \ldots \; \text{HM} \; (\text{sn::r}) \; (\text{B_2.A_2}) \; ) \; [2 \; \text{width}=1/ \; -B_1 \; # H_M \]

\[ @ (\text{ex2_intro} \; \ldots \; \text{HM}) \; -M \; -B_2 \; -A_2 \; \to \text{associative_append} \; /3 \; \text{width}=1/ \]

\[ ]

\[ \text{qed} \; . \]

theorem st_trans: transitive \ldots \; st .

theorem lsreds_standard: \forall s, M, N. M \to^* [s] N \to \exists r. M \to^\forall \to^* [r] N \land \text{is_standard} \; r.

Finally, we add Theorem lrsreds_lsred_swap, which states that weak head reduction steps can be taken in front of a computation. The inductive case of the proof, that we show in full, is very similar to Lemma 4.1 of [Kas00] stating that head reduction steps can be taken in front of a computation. Theorem lrsreds_lsred_swap is the starting point for proving that a term has a weak head normal form if and every computation starting from it contains finitely many weak head reduction steps.
In the proof of Lemma \texttt{st lsred swap} the reader should note two invocations of the tactic \texttt{"lapply (rewrite r ... is whd ... Hq)}", which is the procedural counterpart of the declarative construction \texttt{"cut (is whd (q::r)) [ >Hq]"}.

5. CONCLUSIONS AND FUTURE WORK

In this article we reported on a formalization of pure \(\lambda\)-calculus for the Matita interactive theorem prover \cite{ARST11}, which contains the proofs of two relevant results in reduction theory: the Confluence Theorem \texttt{lsreds conf} of Subsection 4.6 and the Standardization Theorem \texttt{lsreds standard} of Subsection 4.7.

Our aim was to provide what appears to be the first formalization of Kashima’s standardization proof \cite{Kas00} for a Logical Framework. On the other hand, confluence is a due result we provide for completeness following Tait’s and Matin-Lof’s proof based on the so-called “diamond” property of parallel reduction steps (Subsection 4.4) leading to confluence of parallel computations (Subsection 4.5), that has been formalized many times already (see for instance \cite{Pfe92}).
Our main contribution was to adopt path-based pointers (Subsection 3.5) to locate the redexes of $\lambda$-terms. In our view this tool has two major benefits over Kashima’s degree-based pointers. Firstly, we note that one step of arbitrary sequential reduction is defined by four clauses [Bar85], and we can modify this definition including a pointer to the contracted redex without increasing the number of clauses (Subsection 4.3). Secondly, we can characterize the weak head computations with our predicate is whd of Subsection 3.6. So Kashima’s “hap” relation is avoided. Such a characterization is impossible with Kashima’s pointers.

On the other hand, our pointers can be ordered (see Subsection 4.1) as well as Kashima’s ones in such a way that ordered sequences of pointers to redexes should correspond to standard computations. We plan to solve this conjecture by testing our definition of a standard computation against the definition based on residuals.

We also plan to prove the results on leftmost computations mentioned in [Kas00], such as the leftmost reduction theorem and the quasi-leftmost reduction theorem.

In this respect we note that Kashima’s pointers allow a clean characterization of a leftmost reduction step as one for which the pointer to the contracted redex is 1.

We expect to obtain a characterization as well by saying that $p$ is leftmost in $M$ when is minimal in $M$, that is $M \rightarrow_{[q]} N$ implies $p \leq q$ for every $q$ and $N$.

Finally we plan to consider $\eta$-reduction following Kashima in [Kas01].

The source files of our formalization are on Matita’s Web site\footnote{\url{http://matita.cs.unibo.it/library.shtml}}, while the corresponding proof objects are in the HELM directory\footnote{\url{http://helm.cs.unibo.it/library.html}}.<cic:/matita/lambda/>.

References


ACKNOWLEDGMENTS

My sincere thanks go to professor A. Asperti for his valuable comments and suggestions, which greatly improved this text.

I dedicate this work on the beautiful subject of $\lambda$-calculus to (in alphabetical order without repetitions): Ada, Angela, Annalisa, Camilla, Chiara, Elena, Élodie,
Fiorella, Francesca, Gloria, Ida, Isabel, Laura, Loredana, Lucia, Luki, Mariella, Marta, Patrizia, Rosanna, Silvia, and to the other friends of SPaCe, Bologna.

“[Beautiful] is what you love.”
Sappho, Book I.