Proof Synthesis via Refinement Steps: Another Application of Explicit Substitutions on Open Terms (Extended Abstract)

César Muñoz*
Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025, USA
Email: munoz@csl.sri.com
Tel: +1 (650) 859-2784, Fax: +1 (650) 859-2844

February 24, 1998

Abstract

We present a proof synthesis method for the dependent type systems where proofs are built incrementally by successive refinement steps. In particular, we show how the Dowek's proof synthesis method for the cube type systems can be adapted to the incomplete proof-term paradigm via a suitable theory with explicit substitutions and typed meta-variables.

1 Introduction

In the proof-as-term paradigm of the constructive logic, λ-terms with place-holders have been proposed as a convenient framework to represent incomplete proofs [23, 27]. Therefore a term like $\lambda x{:}A.\Box$, where $\Box$ is a place-holder, could be seen as a proof-term of $A{\rightarrow}B$ provided that we can find a term having the type B in the right context to fill the place left by $\Box$. When we use meta-variables to represent place-holders, to fill a place-holder is merely to instantiate a meta-variable. Moreover, if we use the λ-calculus *à la* De Bruijn, a meta-variable is simply a variable of the free-algebra of terms. In order to satisfy typing requirements, it is necessary to provide a mechanism to postpone the application of substitutions to place-holders [23, 10].

Calculi of explicit substitutions [1] improve the substitution mechanism of the λ-calculus by allowing substitutions to be part of the formal language by means of special constructors and reduction rules. Thus, in these calculi it is possible to consider terms with partially applied substitutions. Unfortunately, the theory of explicit substitution calculi is more complex than that of λ-calculus, and in general they do not share the same properties, notably with respect to confluence and strong normalization. This fact is reflected in the several variants of calculi of explicit substitutions that have been proposed; see among others [1, 29, 21, 18, 2, 22, 7, 19, 24, 13, 26].

For the simply-typed λ-calculus, Dowek, Hardin and Kirchner [10] show how to reduce higher-order unification into a first-order framework via explicit substitutions and typed meta-variables.

*This research was supported by National Science Foundation grant CCR-9712383.

In [25, 27] we propose an explicit substitution calculus, namely $\lambda_{\mathcal{L}\Pi}$, for dependent type theories like $\lambda\Pi$ [14] and the Calculus of Constructions [4, 5]. On well-typed expressions containing meta-variables of terms, the $\lambda_{\mathcal{L}\Pi}$-calculus is confluent and weakly normalizing. Although $\lambda_{\mathcal{L}\Pi}$ does not enjoy confluence on the full set of open expressions, i.e. $\lambda_{\mathcal{L}\Pi}$ is no more confluent when meta-variables of substitutions are considered, and it does not preserve strong normalization, i.e. arbitrary reductions on well-typed expressions may not terminate, we argue in [27] that the given properties are sufficient to represent incomplete proof-terms in a constructive logic framework.

In this paper we describe a proof synthesis method for the calculus of construction that uses the incomplete proof-term paradigm. The method we present is strongly inspired by that proposed by Dowek in [8, 9] for the Cube of Type Systems. In contrast to Dowek's work, we use the $\lambda_{\mathcal{L}\Pi}$-calculus, and not the λ-calculus, as the underlying theory. In this way, proof-terms can be built incrementally by successive refinement steps. The method proposed can also be seen as a semi-algorithm for ground higher-order unification in the calculus of construction where unification variables are just meta-variables of the $\lambda_{\mathcal{L}\Pi}$-calculus.

The paper is structured as follows. In Section 2 we introduce the method by means of a simple example. In Section 3 we give an overview to the $\lambda_{\mathcal{L}\Pi}$-calculus. Our proof synthesis method is described in Section 4. Last section presents related works and summarizes this work. We join two appendices, one with the complete description of the $\lambda_{\mathcal{L}\Pi}$-calculus, and other with the soundness proof of the proposed method.

2 An Example

In the proofs-as-terms paradigm, a method of proof synthesis is just a method to find a term of a given type. Since the set of λ-terms is enumerable, a complete method of proof synthesis in any constructive logic where type-checking is decidable consists to enumerate all the terms, and for each one of them to check if it has the right type. Of course this method is impractical for implementations. A smart enumeration of terms takes typing information into account.

Dowek shows in [8, 9] a complete term enumeration algorithm for the type systems of the Barendregt's cube. Let us present the Dowek's method, from an incomplete proof-term perspective, with an example.

Assume we search a proof of $\forall p{:}(\forall h{:}nat.(h \geq 0)).(0 \geq 0)$ in a context Γ containing the following declarations $nat{:}Type$, $\geq{:}nat \rightarrow (nat \rightarrow Type)$, $0{:}nat$. In our formalism, we search an instantiation for a meta-variable Y such that $\Gamma \vdash Y : \Pi p{:}(\Pi h{:}nat.(h \geq 0)).(0 \geq 0)$. A possible instantiation for Y has the form $\lambda p{:}(\Pi h{:}nat.(h \geq 0)).Y_1$, where

$$\Gamma,\ p{:}(\Pi h{:}nat.(h \geq 0)) \vdash Y_1 : (0 \geq 0)$$

We suspect that an instantiation for Y_1 has the form $(p\ Y_2)$ where

$$\Gamma,\ p{:}(\Pi h{:}nat.(h \geq 0)) \vdash Y_2 : nat$$

However, we cannot instantiate Y_1 with $(p\ Y_2)$ because Y_1 has the type $(0 \geq 0)$, while $(p\ Y_2)$ has the type $(Y_2 \geq 0)$. In order to guarantee the soundness of the construction, we must solve the constraint $(0 \geq 0) = (Y_2 \geq 0)$. In this case, the constraint is satisfied if we instantiate Y_2 with 0.

We can verify that the sequential application of the above instantiations gives a term in normal form of the right type, i.e. if we denote by $\{X/N\}$ the instantiation of a meta-variable X with a term N, and by $e\{X/N\}$ its application to an expression, then

$$Y\{Y/\lambda p{:}(\Pi h{:}nat.(h \geq 0)).Y_1\}\{Y_1/(p\ Y_2)\}\{Y_2/0\} = \lambda p{:}(\Pi h{:}nat.(h \geq 0)).(p\ 0)$$

and

$$\Gamma \vdash \lambda p{:}(\Pi h{:}nat.(h \geq 0)).(p\ 0) : \Pi p{:}(\Pi h{:}nat.(h \geq 0)).(0 \geq 0)$$

The above example shows that a complete term enumeration algorithm for dependent type systems must solve typing constraints. In the case of polymorphism and constructions of types, it must also deal with the problem of splitting [8], i.e. the arity of a function can not be deduced from its type.

Informally, the method to find an instantiation for a meta-variable X of type A in a context Γ can be summarized as follows:

1. Guess a form of X according to the type information given by A and Γ.
2. Declare new meta-variables for the unknown parts of the term, and introduce typing constraints in order to guarantee the soundness of the process.
3. Call recursively the method for each new meta-variable.
4. Stop when all the typing constraints have been satisfied.

Of course the hearth of the method is step 1. In Dowek's terminology this step consist to find an *elementary substitution*. In our case, we search for *elementary graftings*. Before presenting our proof synthesis method, we give an overview to the $\lambda_{\mathcal{L}\Pi}$-calculus.

3 The $\lambda_{\mathcal{L}\Pi}$-calculus and its Extension with Constraints

As we have said, the underlying theory of our proof synthesis method is the $\lambda_{\mathcal{L}\Pi}$-calculus of explicit substitutions [25]. This calculus is a dependent-typed version of a left-linear variant of $\lambda\sigma$ [1].

In $\lambda_{\mathcal{L}\Pi}$, variables use the De Bruijn indices notation and meta-variables use names. As usual in typed calculi *à la* De Bruijn, variables are declared in lists of types called *contexts*. In addition, meta-variables are declared in list structures called *signatures*. A meta-variable declaration has the form $X{:}_{\Gamma}A$, where Γ and A are, respectively, a context and a type associated to the meta-variable X.

Definition 1 (Contexts and Signatures) *Well formed* contexts *and* signatures *are defined by the following grammar:*

Contexts	Γ, Δ	$::=$	$nil \mid \Gamma.A$
Signatures	Σ	$::=$	$nil \mid \Sigma.\ X{:}_{\Gamma}A$

We consider typing assertions having one of the following forms:

- $\vdash \Sigma$, the signature Σ is valid.
- $\vdash \Sigma :\cdot \Gamma$, the context Γ is valid in the signature Σ.
- $\Sigma :\cdot \Gamma \vdash M : A$, the term M has type A in $\Sigma :\cdot \Gamma$.

If the signature Σ is empty, we usually write $\vdash \Gamma$ and $\Gamma \vdash M : A$.

In dependent type theories, meta-variables may appear in types and in contexts. Hence, we need scoping rules for meta-variables and variables.

- In the typing assertion $\vdash \Sigma_1.\ X{:}_\Gamma A.\ \Sigma_2 :\cdot \Delta$, the context Γ and the type A can use all the meta-variables declared in Σ_1, the indices in A are relative to Γ, and the context Δ can use all the meta-variables declared in the signature $\Sigma_1.\ X{:}_\Gamma A.\ \Sigma_2$.
- In the typing assertion $\Sigma :\cdot \Gamma \vdash M : A$, the term M an the type A can use all the meta-variables declared in Σ, and the indices in M and A are relative to Γ.

The full rewrite system and the typing rules of $\lambda_{\mathcal{L}\Pi}$ are shown in Appendix A. We prove in [25, 27] that $\lambda_{\mathcal{L}\Pi}$ satisfies, among others, the following properties on well-typed terms:

- Church-Rosser (and so confluence) and Weak Normalization.
- Type Uniqueness, i.e. types are unique modulo conversions.
- Subject Reduction, i.e. types are preserved under reductions.
- Instantiation Soundness, i.e. types are preserved under instantiations of meta-variables.

Now, we extend the $\lambda_{\mathcal{L}\Pi}$-system in order to deal with typing constraints.

Definition 2 (Constraints and Constrained Signatures) *A* constraint *is a triple* (M, N, Γ), *denoted by* $M =^?_\Gamma N$, *relating two terms* M, N *and one context* Γ. *A* constrained signature *is a list containing meta-variable declarations —just as in Def. 1— and constraint declarations. Formally, they are defined by the following grammar:*

$$\textbf{Constrained Signatures}\quad \Sigma_c \quad ::= \quad nil \mid \Sigma_c.\ X{:}_\Gamma A \mid \Sigma_c.\ M =^?_\Gamma N$$

Definition 3 (Equivalence Modulo Constraints) *Let* Σ_c *be a constrained signature, we define the relation* $\equiv_{\Sigma_c}$ *as the smallest equivalence relation compatible with structure such that*

- *if* $M =_{\lambda_{\mathcal{L}\Pi}} N$, *then* $M \equiv_{\Sigma_c} N$, *and*
- *if* $M =^?_\Gamma N \in \Sigma_c$, *then* $M \equiv_{\Sigma_c} N$.

Definition 4 *The type system with constraints is defined as in Appendix 1, but we replace signatures by constrained signatures, we denote typing judgments by* $\vdash_c \Sigma_c$, $\vdash_c \Sigma_c :\cdot \Gamma$ *and* $\Sigma_c :\cdot \Gamma \vdash_c M : A$, *we add the rule*

$$\frac{\Sigma_c :\cdot \Gamma \vdash_c M : A \quad \Sigma_c :\cdot \Gamma \vdash_c M' : A}{\vdash_c \Sigma_c.\ M =^?_\Gamma M'}\ \text{(Constraint)}$$

and we replace the rule (Conv) *by*

$$\frac{\Sigma_c :\cdot \Gamma \vdash_c M : A_1 \quad \Sigma_c :\cdot \Gamma \vdash_c A_2 : s \quad s \in \{Kind, Type\} \quad A_1 \equiv_{\Sigma_c} A_2}{\Sigma_c :\cdot \Gamma \vdash_c M : A_2}\ \text{(Conv)}$$

Notation: If Σ_c is a constrained signature, we denote by $(\Sigma_c)^{-c}$ the signature obtained by dropping the constraints of Σ_c.

Definition 5 (Normal Form of a Signature) *Let* Σ_c *be a constrained signature, the* normal form *of* Σ_c, *denoted by* $(\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}}$, *is obtained by the application of the following rules*[1]*:*

[1] We denote by $(M)\!\downarrow_{\lambda_{\mathcal{L}\Pi}}$ the normal-form of a well-typed term M.

$$\Sigma'_c.\ X{:}_\Gamma A.\ \Sigma''_c \xrightarrow{\lambda_{\mathcal{L}\Pi}} \Sigma'_c.\ X{:}_\Gamma (A){\downarrow_{\lambda_{\mathcal{L}\Pi}}}.\ \Sigma''_c \qquad \textit{if } (\Sigma'_c)^{-c} :\cdot \Gamma \vdash X : A \textit{ and } A \textit{ is not a } \lambda_{\mathcal{L}\Pi}\textit{-normal form}$$

$$\Sigma'_c.\ M =^?_\Gamma N.\ \Sigma''_c \xrightarrow{\lambda_{\mathcal{L}\Pi}} \Sigma'_c.\ (M){\downarrow_{\lambda_{\mathcal{L}\Pi}}} =^?_\Gamma N.\ \Sigma''_c \qquad \textit{if } (\Sigma'_c)^{-c} :\cdot \Gamma \vdash M : A \textit{ and } M \textit{ is not a } \lambda_{\mathcal{L}\Pi}\textit{-normal form}$$

$$\Sigma'_c.\ M =^?_\Gamma N.\ \Sigma''_c \xrightarrow{\lambda_{\mathcal{L}\Pi}} \Sigma'_c.\ M =^?_\Gamma (N){\downarrow_{\lambda_{\mathcal{L}\Pi}}}.\ \Sigma''_c \qquad \textit{if } (\Sigma'_c)^{-c} :\cdot \Gamma \vdash N : A \textit{ and } N \textit{ is not a } \lambda_{\mathcal{L}\Pi}\textit{-normal form}$$

$$\Sigma'_c.\ M =^?_\Gamma M.\ \Sigma''_c \xrightarrow{\lambda_{\mathcal{L}\Pi}} \Sigma'_c.\ \Sigma''_c$$

Remark 6 *The above definition is sound since the $\lambda_{\mathcal{L}\Pi}$-normal form of a well-typed expression always exists and it is unique (for details see [27]).*

We recall that an expression x is said to be *ground* if it has no occurrence of meta-variables.

Definition 7 (Failure Signature) *Let Σ_c be the normal form of a valid constrained signature, we say that Σ_c is a* failure signature *if it contains a constraint relating two ground terms in $\lambda_{\mathcal{L}\Pi}$-normal form which are not identical.*

In Appendix B we show the following proposition.

Proposition 8 *Let Σ_c be the normal form of a valid constrained signature such that $\Sigma_c \neq nil$ and Σ_c is not a failure signature. Then, Σ_c has the form $\Sigma'_c.\ X{:}_\Gamma A.\ \Sigma''_c$, where*

- *Σ'_c does not have constraints,*
- *$\vdash \Sigma'_c.\ X{:}_\Gamma A$, and*
- *A is a $\lambda_{\mathcal{L}\Pi}$-normal form.*

4 A Proof Synthesis Method via Refinement Steps

As we have seen in the informally description of the method in Section 2, instantiation of meta-variables may need the declaration of new meta-variables and constraints. Thus, we generalize the instantiation mechanism in a *grafting* mechanism.

Definition 9 (Graftings) *A* grafting *θ_c is a triple (X, M, Σ_c), denoted by $\{X/_{\Sigma_c} M\}$, where X is a meta-variable, M is a term and Σ_c is a constrained signature.*

Notation: If $\theta_c = \{X/_{\Sigma_c} M\}$, we denote by $(\theta_c)^{-c}$ the instantiation $\{X/M\}$.

Definition 10 (Valid Graftings) *Let Σ_c be a valid constrained signature, we say that $\{X/_{\Sigma'_c} M\}$ is a* valid grafting of *Σ_c, if and only if*

- *the meta-variables declared in Σ'_c are not in Σ_c, and*
- *Σ_c has the form $\Sigma_{c_1}.\ X{:}_\Gamma A.\ \Sigma_{c_2}$ where $\Sigma_{c_1}.\ \Sigma'_c :\cdot \Gamma \vdash_c M : A$.*

Definition 11 *Let* $\theta_c = \{X/_{\Sigma'_c} M\}$ *be a valid grafting of* $\Sigma_c = \Sigma_{c_1}.\ X{:}_\Gamma A.\ \Sigma_{c_2}$, *the* application of θ_c to Σ_c, *denoted by* $\Sigma_c\theta_c$, *is defined as:*

$$\Sigma_c\theta_c = \Sigma_{c_1}.\ \Sigma'_c.\ \Sigma_{c_2}\{X/M\}.$$

Definition 12 (Sequential Graftings) *Let* Σ_c *be a valid constrained signature, we say that a list* ψ *of graftings is a* sequential grafting of Σ_c *if and only if*

- ψ *is the empty list, or*
- $\psi = \theta_c, \psi'$, *where* θ_c *is a valid grafting of* Σ_c, *and* ψ' *is a sequential grafting of* $\Sigma_c\theta_c$.

The application of a sequential grafting ψ to Σ_c, *denoted by* $\Sigma_c\psi$, *consists in the sequential application of graftings in* ψ *to* Σ_c.

Definition 13 (Derivation) *Let* ψ *be a sequential grafting of a valid constrained signature* Σ_c, *we say that* ψ *is a derivation of* Σ_c *if and only if* $(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}} = nil$.

Remark 14 *Failure signatures do not have derivations.*

Definition 15 (Parallel Instantiation) *A* parallel instantiation *is a function* Ψ *from meta-variables to terms such that* $\Psi(X) \neq X$ *for a number finite of meta-variables* X. *As usual, the function* Ψ *is extended to be applied to arbitrary expressions.*

Definition 16 *The* restriction *of a parallel instantiation* Ψ *to a constrained signature* Σ_c, *denoted by* $\Psi\backslash_{\Sigma_c}$, *is defined as*

$$\begin{array}{rcll} \Psi\backslash_{\Sigma_c}(X) & = & \Psi(X) & \textit{if } X \in \Sigma_c \\ \Psi\backslash_{\Sigma_c}(X) & = & X & \textit{otherwise} \end{array}$$

Definition 17 (Solution) *Let* Σ_c *be a valid constrained signature, we say that a parallel instantiation* Ψ *is a* solution of Σ_c *if and only if*

- *for any constraint* $M_1 =^?_\Gamma M_2 \in \Sigma_c$, *we have* $\Psi(\Gamma) \vdash \Psi(M_1) : A$, $\Psi(\Gamma) \vdash \Psi(M_2) : A$ *and* $\Psi(M_1) =_{\lambda_{\mathcal{L}\Pi}} \Psi(M_2)$, *and*
- *for any meta-variable declaration* $X{:}_\Gamma A \in \Sigma_c$, *we have* $\Psi(\Gamma) \vdash \Psi(X) : \Psi(A)$.

In this case we say that Σ_c *is a* solvable *signature.*

The problem to know if a valid constrained signature is solvable is undecidable in the general case. However, we can verify easily the following property.

Remark 18 *Failure signatures are not solvable.*

Definition 19 (Composition) *Let* ψ *be a sequential grafting. The* composition *of* ψ, *denoted by* $\widetilde{\psi}$, *is the parallel instantiation defined inductively on* ψ *as follows*

$$\begin{array}{rcll} \widetilde{\psi}(Y) & = & Y & \textit{if } \psi \textit{ is the empty list} \\ \widetilde{\psi}(X) & = & \widetilde{\psi'}(M) & \textit{if } \psi = \{X/_{\Sigma_c} M\}, \psi' \\ \widetilde{\psi}(Y) & = & \widetilde{\psi'}(Y) & \textit{if } \psi = \{X/_{\Sigma_c} M\}, \psi' \textit{ and } Y \neq X \end{array}$$

The next proposition, proved in Appendix B, shows the relation between derivations and solutions. It is the key to prove the soundness of the proposed method.

Proposition 20 *If ψ is a derivation of a valid constrained signature Σ_c, then $\widetilde{\psi}\backslash_{\Sigma_c}$ is a solution of Σ_c.*

Definition 21 (Elementary Graftings) *Let Σ_c be the normal form of a valid constrained signature such that $\Sigma_c \neq nil$ and Σ_c is not a failure signature. We choose a meta-variable X in Σ_c, i.e. $\Sigma_c = \Sigma'_c.\ X\colon_\Gamma A.\ \Sigma''_c$, such that*

1. $\vdash \Sigma'_c.\ X\colon_\Gamma A$, *and*
2. A *is a* $\lambda_{\mathcal{L}\Pi}$*-normal form.*

Such a meta-variable exists by Proposition 8. We define the following graftings by case analysis on A (the cases are not disjoint):

- $A = Kind$. *We consider the grafting* $\{X/_{nil}\,Type\}$.
- $A \in \{Kind, Type\}$. *For any $A' \in \{Kind, Type\}$, we consider the grafting $\{X/_{\Sigma''_c}\Pi_H.K\}$, where $H \notin \Sigma_c$, $K \notin \Sigma_c$, and $\Sigma''_c = nil.\ H\colon_\Gamma A'.\ K\colon_{\Gamma.H} A$.*
- $A = \Pi_{A_1}.A_2$. *We consider the grafting $\{X/_{\Sigma''_c}\lambda_{A_1}.Y\}$, where $Y \notin \Sigma_c$, and $\Sigma''_c = nil.\ Y\colon_{\Gamma.A_1} A_2$.*
- $\Sigma'_c :\cdot \Gamma \vdash A : s,\ s \in \{Kind, Type\}$. *We need some definitions.*

  Definition 22 *The standard π-reduction is inductively defined as*

$$\begin{array}{lcl} \pi([S](\Pi_M.N)) & = & \Pi_{[S]M}.\pi([(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M}]N) \\ \pi([S]M) & = & [S]M \qquad \textit{if } M \textit{ is not a product} \end{array}$$

  Definition 23 *Let $\chi = X_1, H_1, K_1, X_2, H_2, K_2, \ldots$ be an infinite list of distinct names of meta-variables. We define the set $\chi(\Gamma, M, N)^i$, $i \geq 0$, as formed by the triples (Σ'_c, M', N') satisfying the following conditions:*

  - *If $i = 0$, then $\Sigma'_c = nil$, $M' = M$, and $N' = N$.*
  - *If $i > 0$ and $(\Sigma''_c, M'', N'') \in \chi(\Gamma, M, N)^{i-1}$, then*
    - *If N'' has the form $\Pi_{N_1}.N_2$, then*
      - $\Sigma'_c = \Sigma''_c.\ X_i\colon_\Gamma N_1$,
      - $M' = (M''\ X_i)$, *and*
      - $N' = \pi([\uparrow^0 \cdot X_{i:N_1}]N_2)$.
    - *Otherwise, let s' be in $\{Kind, Type\}$*
      - $\Sigma'_c = \Sigma''_c.\ H_i\colon_\Gamma s'.\ K_i\colon_{\Gamma.H_i} s.\ X_i\colon_\Gamma H_i.\ N'' =^?_\Gamma \Pi_{H_i}.K_i$,
      - $M' = (M''\ X_i)$, *and*
      - $N' = [\uparrow^0 \cdot X_{i:H_i}]K_i$.

  For $1 \leq n \leq$ Length(Γ), such that

  1. $\Sigma'_c :\cdot \Gamma \vdash \underline{n} : B$, *and*

2. $\Sigma'_c :\cdot \Gamma \vdash B : s,\ s \in \{Kind, Type\}$

we consider all the graftings

$$\{X/_{\Sigma''_c\,.\,A=^?_\Gamma A'} M\}$$

where $(\Sigma''_c, M, A') \in \chi(\Gamma, \underline{n}, (B)\downarrow_{\lambda_{\mathcal{L}\Pi}})^i$ *for some list* χ *of infinite distinct names of meta-variables not in* Σ_c*, and* $i \geq 0$.

All the graftings considered above form the set of elementary graftings *of the meta-variable* X *in* Σ_c.

Definition 24 (Search Tree) *Let* Σ_c *be a valid constrained signature, we build a* search tree *of* Σ_c*, where nodes are labeled by normal constrained signatures and edges by elementary graftings, in the following way:*

- *The root is labeled by* $(\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}}$.
- *Nodes labeled by the empty signature or by failure signatures are leaves.*
- *If a node is labeled by a signature* Σ_c *which is not the empty signature or a failure signature, we choose a meta-variable* X *in* Σ_c *such that it is well-typed without constraints and for each elementary grafting* $\theta_c = \{X/_{\Sigma'_c} M\}$ *of* X*, we grow an edge labeled by* θ_c *to a new node labeled by* $(\Sigma_c\theta_c)\downarrow_{\lambda_{\mathcal{L}\Pi}}$.

Theorem 25 (Soundness) *Let* $(\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}} \xrightarrow{\psi} nil$ *be a path of a tree search of* Σ_c*. The composition of* ψ *restricted to* Σ_c*, i.e.* $\widetilde{\psi}\backslash_{\Sigma_c}$*, is a solution of* Σ_c.

Proof. The proof is based on the fact that if $\Sigma_{c_0} \xrightarrow{\theta_{c_0}} \Sigma_{c_1} \longrightarrow \ldots \xrightarrow{\theta_{c_{i-1}}} \Sigma_{c_i}$ is a path of a tree search of Σ_c such that $\Sigma_{c_0} = (\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}}$, then the list of graftings $\psi = \theta_{c_0}, \ldots, \theta_{c_i}$ is a sequential grafting of Σ_c and $\Sigma_{c_i} = (\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}}$.

Therefore, ψ is a sequential grafting of Σ_c and $nil = (\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}}$. By Def. 13, ψ is a derivation of Σ_c. We conclude with Proposition 20 that $\widetilde{\psi}\backslash_{\Sigma_c}$ is a solution of Σ_c. □

A semi-algorithm to solve a valid constrained signature is to enumerate the nodes of a search tree in order to find a leave labeled by the *nil* signature. The enumeration must deal with infinite paths in the tree, but also with infinite branching due to the fact that the set of elementary graftings of a meta-variable is potentially infinite.

5 Related Works and Summary

Automatic proof synthesis is at the basis of proof assistant systems. A complete method for search of proof-trees based on resolution and unification was formulated by Robinson [30] for the first-order logic, and by Huet [15] for the higher-order logic. In type systems, higher-order unification (HOU) algorithms are known for the simply-typed λ-calculus [16] and for the $\lambda\Pi$-calculus of dependent types [12, 28].

For the cube type systems, Dowek [8, 9] re-formulates the unification procedure and generalizes it as a method of term enumeration. Recently, Cornes [6] proposes an extension of Dowek's method to the Calculus of Constructions with Inductive Types.

Dowek, Hardin and Kirchner [10] propose a first-order presentation of the Huet's HOU-algorithm based on explicit substitutions and typed meta-variables. This algorithm is generalized to solve higher-order equational unification by Kirchner and Ringeissen [20], and restricted to the case of higher-order patterns by Dowek, Hardin, Kirchner and Pfenning in [11].

On other hand, Briaud [3] shows how HOU can be considered as a typed narrowing in the $\lambda\upsilon$-calculus of explicit substitutions. Magnusson [23] presents a unification algorithm in the Martin-Löf's type theory with explicit substitutions. This algorithm solves first-order unification problems, but leaves unsolved the flexible-flexible constraints.

Our main contribution is the presentation of the Dowek's method of proof synthesis in a suitable theory with explicit substitutions and typed meta-variables. In this way, proof-terms can be built incrementally by successive instantiation of meta-variables.

Just as in [8, 9], the method presented here is sound and we conjecture its completeness. Thus, it can be seen as a semi-algorithm for ground higher-order unification in the calculus of constructions.

Although the underlying theory of the method proposed here is the $\lambda_{\mathcal{L}\Pi}$-calculus, we think that the same ideas can be applied to other formalisms satisfying at least the same typing properties as $\lambda_{\mathcal{L}\Pi}$, i.e. confluence and weak-normalization on well-typed terms, subject reduction and instantiation soundness. Moreover, the $\lambda_{\mathcal{L}\Pi}$-calculus has some features that are useful for our proof-synthesis method and they seem to be in unification issues:

- It is a finitary first-order rewriting system, where the α-conversion of the λ-calculus is also explicit.

- It uses general composition of substitutions and simultaneous substitutions. In [27] we discuss some efficiency improvements based on these features.

- Since substitutions go through abstraction and products, normal forms have a simple characterization. For example, the normal form of a type has the form $\Pi_{A_1}.\dots\Pi_{A_i}.A$ where A is not a product.

Acknowledgments

I gratefully acknowledge the contribution of ideas from Gilles Dowek during my stay within the Coq team at INRIA-Rocquencourt.

References

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

[2] R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *Proceedings of CSN-95: Computer Science in the Netherlands*, November 1995.

[3] D. Briaud. Higher order unification as a typed narrowing. Technical report, CRIN report 96-R-112, 1996.

[4] T. Coquand. *Une Théorie de Constructions*. Thèse de doctorat, U. Paris VII, 1985.

[5] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76:96–120, 1988.

[6] C. Cornes. *Conception d'un langage de haut niveau de représentation de preuves: récurrence par filtrage de motifs, unification en présence de types inductif primitifs. synthèse de lemmes d'inversion.* Thèse de doctorat, Université Paris 7, 1997.

[7] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, March 1996.

[8] G. Dowek. *Démonstration automatique dans le calcul des constructions.* Thèse de doctorat, U. Paris VII, 1991.

[9] G. Dowek. A complete proof synthesis method for type systems of the cube. *Journal of Logic and Computation*, 3(3):287–315, June 1993.

[10] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions (extended abstract). In *Proceedings of the Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 366–374, San Diego, California, 26–29 June 1995. IEEE Computer Society Press.

[11] G. Dowek, T. Hardin, C. Kirchner, and F. Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, Bonn, Germany, September 1996. MIT Press.

[12] C. Elliott. Higher-order unification with dependent types. In N. Dershowitz, editor, *Proceedings fo the International Conference on Rewriting Techniques and Applications (RTA-89)*, volume 355 of *LNCS*, pages 121–136, Chapel Hill, North Carolina, April 1989. Springer-Verlag.

[13] M. C. F. Ferreira, D. Kesner, and L. Puel. λ-calculi with explicit substitutions and composition which preserve β-strong normalization. *LNCS*, 1139, 1996.

[14] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[15] G. Huet. *Constrained Resolution A Complete Method for Higher Order Logic.* PhD thesis, Case Western Reserve University, 1972.

[16] G. Huet. A unification algorithm for typed lambda calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.

[17] G. Huet. *Résolution d'Équations dans les Langages d'Ordre 1,2, ...,ω.* Thèse de doctorat, U. Paris VII, 1976.

[18] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. In *Proceedings of PLILP'95*. LNCS, 1995.

[19] D. Kesner. Confluence properties of extensional and non-extensional λ-calculi with explicit substitutions (extended abstract). In H. Ganzinger, editor, *Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA-96)*, volume 1103 of *LNCS*, pages 184–199, New Brunswick, NJ, USA, 1996. Springer-Verlag.

[20] C. Kirchner and C. Ringeissen. Higher order equational unification via explicit substitutions. In *Proceedings of the International Conference PLILP/ALP/HOA'97*, volume 1298 of *LNCS*, Southampton (England), September 1997. Springer.

[21] P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$ a journey through calculi of explicit substitutions. In *Proceedings of the 21st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 60–69, January 1994.

[22] P. Lescanne and J. Rouyer-Degli. Explicit substitutions with de Bruijn's levels. In J. Hsiang, editor, *Proceedings of the International Conference on Rewriting Techniques and Applications (RTA-95)*, volume 914 of *LNCS*, pages 294–308, Chapel Hill, North Carolina, 1995. Springer-Verlag.

[23] L. Magnusson. *The Implementation of ALF—A Proof Editor Based on Martin-Löf's Monomorphic Type Theory with Explicit Substitution.* PhD thesis, Chalmers University of Technology and Göteborg University, January 1995.

[24] C. Muñoz. Confluence and preservation of strong normalisation in an explicit substitutions calculus (extended abstract). In *Proceedings of the Eleven Annual IEEE Symposium on Logic in Computer Science*, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

[25] C. Muñoz. Dependent types with explicit substitutions: A meta-theoretical development. In *Types for Proofs and Programs, Proceedings of International Workshop TYPES'96*, 1997.

[26] C. Muñoz. A left-linear variant of $\lambda\sigma$. In *Proceedings of the International Conference PLILP/ALP/HOA'97*, volume 1298 of *LNCS*, Southampton (England), September 1997. Springer.

[27] C. Muñoz. *Un calcul de substitutions pour la représentation de preuves partielles en théorie de types.* Thèse de doctorat, Université Paris 7, 1997. Version en anglais disponible comme rapport de recherche INRIA RR-3309.

[28] D. Pym. A unification algorithm for the $\lambda\Pi$-calculus. *International Journal of Foundations of Computer Science*, 3(3):333–378, September 1992.

[29] A. Ríos. *Contributions à l'étude de λ-calculs avec des substitutions explicites.* Thèse de doctorat, U. Paris VII, 1993.

[30] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

A The $\lambda_{\mathcal{L}\Pi}$-system

The set of well-formed expressions in $\lambda_{\mathcal{L}\Pi}$ is defined by the following grammar:

$$\begin{array}{lll l}
\textbf{Naturals} & n & ::= & 0 \mid Suc(n) \\
\textbf{Pre-terms} & M, N & ::= & Kind \mid Type \mid \mathbf{1} \mid \Pi_M.N \mid \lambda_M.N \mid (M\ N) \mid [S]M \\
\textbf{Pre-substitutions} & S, T & ::= & \uparrow^n \mid S \cdot M_{:N} \mid T \circ S
\end{array}$$

The $\lambda_{\mathcal{L}\Pi}$-calculus is given by the rewrite system of Fig. 1.

We recall that well-formed contexts and signatures are defined by the following grammar:

$$\begin{array}{llll}
\textbf{Contexts} & \Gamma & ::= & nil \mid \Gamma.M \\
\textbf{Signatures} & \Sigma & ::= & nil \mid \Sigma.\ X{:}_\Gamma M
\end{array}$$

and typing assertions have one of the following forms:

- $\vdash \Sigma :\cdot nil$, the signature Σ is valid.
- $\vdash \Sigma :\cdot \Gamma$, the context Γ is valid in the signature Σ.
- $\Sigma :\cdot \Gamma \vdash M : N$, the term M has type N in $\Sigma :\cdot \Gamma$.
- $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, the substitution S has as type the context Δ in $\Sigma :\cdot \Gamma$.

Notation:

- We use $\vdash \Sigma$, $\vdash \Gamma$, $\Gamma \vdash M : N$, and $\Gamma \vdash S \triangleright \Delta$ as notations for $\vdash \Sigma :\cdot nil$, $\vdash nil :\cdot \Gamma$, $nil :\cdot \Gamma \vdash M : N$, and $nil :\cdot \Gamma \vdash S \triangleright \Delta$, respectively.
- We use the lowercase letter s to range over the set of sorts $\{Kind, Type\}$.
- The expression "$X \notin \Sigma$" means that the meta-variable X is not declared in Σ, i.e. there is no context Γ and term M such that $X{:}_\Gamma M \in \Sigma$.

Typing rules for signatures, contexts, terms and substitutions are all mutually dependent. We first give the rules to valid signatures and contexts.

$$
\begin{array}{llll}
(\lambda_{M'}.M\ N) & \longrightarrow & [\uparrow^0 \cdot N_{:M'}]M & \text{(Beta)} \\
\\
[S](\lambda_M.N) & \longrightarrow & \lambda_{[S]M}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M}]N & \text{(Lambda)} \\
[S](\Pi_M.N) & \longrightarrow & \Pi_{[S]M}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M}]N & \text{(Pi)} \\
[S](M\ N) & \longrightarrow & ([S]M\ [S]N) & \text{(Application)} \\
[T][S]M & \longrightarrow & [T \circ S]M & \text{(Clos)} \\
[S \cdot M_{:N}]\mathbf{1} & \longrightarrow & M & \text{(VarCons)} \\
[\uparrow^0]M & \longrightarrow & M & \text{(Id)} \\
T \circ (S \cdot M_{:N}) & \longrightarrow & (T \circ S) \cdot [T]M_{:N} & \text{(Map)} \\
S \circ \uparrow^0 & \longrightarrow & S & \text{(IdS)} \\
(S \cdot M_{:N}) \circ \uparrow^{Suc(n)} & \longrightarrow & S \circ \uparrow^n & \text{(ShiftCons)} \\
\uparrow^m \circ \uparrow^{Suc(n)} & \longrightarrow & \uparrow^{Suc(m)} \circ \uparrow^n & \text{(ShiftShift)} \\
\uparrow^{Suc(0)} \cdot \mathbf{1}_{:M} & \longrightarrow & \uparrow^0 & \text{(Shift0)} \\
\uparrow^{Suc(n)} \cdot [\uparrow^n]\mathbf{1}_{:M} & \longrightarrow & \uparrow^n & \text{(ShiftS)} \\
[S]\,\mathit{Type} & \longrightarrow & \mathit{Type} & \text{(Type)}
\end{array}
$$

Figure 1: The $\lambda_{\mathcal{L}\Pi}$-rewrite system

Valid Signatures and Contexts

$$\frac{}{\vdash nil :\cdot nil}\ \text{(Empty)} \qquad \frac{\Sigma :\cdot \Gamma \vdash M : s}{\vdash \Sigma :\cdot \Gamma.M}\ \text{(Var-Decl)}$$

$$\frac{\vdash \Sigma :\cdot \Gamma \quad X \notin \Sigma}{\vdash \Sigma.\ X :_\Gamma \mathit{Kind}}\ \text{(Metavar-Decl}_1\text{)} \qquad \frac{\Sigma :\cdot \Gamma \vdash M : s \quad X \notin \Sigma}{\vdash \Sigma.\ X :_\Gamma M}\ \text{(Metavar-Decl}_2\text{)}$$

Valid terms and substitutions are formed as follows:

Valid Terms

$$\frac{\vdash \Sigma :\cdot \Gamma}{\Sigma :\cdot \Gamma \vdash \mathit{Type} : \mathit{Kind}}\ \text{(Type)} \qquad \frac{\Sigma :\cdot \Gamma.M \vdash N : s}{\Sigma :\cdot \Gamma \vdash \Pi_M.N : s}\ \text{(Prod)}$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Delta \vdash M : \mathit{Kind}}{\Sigma :\cdot \Gamma \vdash [S]M : \mathit{Kind}}\ \text{(Clos-Kind)} \qquad \frac{\vdash \Sigma :\cdot \Gamma.M}{\Sigma :\cdot \Gamma.M \vdash \mathbf{1} : [\uparrow^{Suc(0)}]M}\ \text{(Var)}$$

$$\frac{\Sigma :\cdot \Gamma \vdash M : \Pi_{M_1}.M_2 \quad \Sigma :\cdot \Gamma \vdash N : M_1}{\Sigma :\cdot \Gamma \vdash (M\ N) : [\uparrow^0 \cdot N_{:M_1}]M_2} \text{(Appl)}$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Delta \vdash M : N \quad \Sigma :\cdot \Delta \vdash N : s}{\Sigma :\cdot \Gamma \vdash [S]M : [S]N} \text{(Clos)}$$

$$\frac{\vdash \Sigma :\cdot \Gamma \quad X :_\Delta M \in \Sigma \quad \Delta =_{\lambda_{\mathcal{L}\Pi}} \Gamma}{\Sigma :\cdot \Gamma \vdash X : M} \text{(Metavar)}$$

$$\frac{\Sigma :\cdot \Gamma.M_1 \vdash M_2 : N \quad \Sigma :\cdot \Gamma \vdash \Pi_{M_1}.N : s}{\Sigma :\cdot \Gamma \vdash \lambda_{M_1}.M_2 : \Pi_{M_1}.N} \text{(Abs)}$$

$$\frac{\Sigma :\cdot \Gamma \vdash M : M_1 \quad \Sigma :\cdot \Gamma \vdash M_2 : s \quad M_1 =_{\lambda_{\mathcal{L}\Pi}} M_2}{\Sigma :\cdot \Gamma \vdash M : M_2} \text{(Conv)}$$

Valid Substitutions

$$\frac{\vdash \Sigma :\cdot \Gamma}{\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Gamma} \text{(Id)} \qquad \frac{\vdash \Sigma :\cdot \Gamma.M \quad \Sigma :\cdot \Gamma \vdash \uparrow^n \triangleright \Delta}{\Sigma :\cdot \Gamma.M \vdash \uparrow^{Suc(n)} \triangleright \Delta} \text{(Shift)}$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta' \quad \Sigma :\cdot \Delta' \vdash T \triangleright \Delta}{\Sigma :\cdot \Gamma \vdash S \circ T \triangleright \Delta} \text{(Comp)}$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Gamma \vdash M : [S]N \quad \Sigma :\cdot \Delta \vdash N : s}{\Sigma :\cdot \Gamma \vdash S \cdot M_{:N} \triangleright \Delta.N} \text{(Cons)}$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \vdash \Sigma :\cdot \Delta' \quad \Delta =_{\lambda_{\mathcal{L}\Pi}} \Delta'}{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta'} \text{(Conv-Subs)}$$

In [27] we show that $\lambda_{\mathcal{L}\Pi}$ satisfies the following properties:

Proposition 26 (Sort Soundness)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, *then either* $N = Kind$, *or* $\Sigma :\cdot \Gamma \vdash N : s$, *and*
2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then* $\vdash \Sigma :\cdot \Delta$.

Proposition 27 (Type Uniqueness) *Let* Γ_1 *and* Γ_2 *be such that* $\Gamma_1 =_{\lambda_{\mathcal{L}\Pi}} \Gamma_2$,

1. *if* $\Sigma :\cdot \Gamma_1 \vdash M : N_1$ *and* $\Sigma :\cdot \Gamma_2 \vdash M : N_2$, *then* $N_1 =_{\lambda_{\mathcal{L}\Pi}} N_2$, *and*
2. *if* $\Sigma :\cdot \Gamma_1 \vdash S \triangleright \Delta_1$ *and* $\Sigma :\cdot \Gamma_2 \vdash S \triangleright \Delta_2$, *then* $\Delta_1 =_{\lambda_{\mathcal{L}\Pi}} \Delta_2$.

Proposition 28 (Subject Reduction) *If* $x \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} y$, *then*

1. *if* x *is a term and* $\Sigma :\cdot \Gamma \vdash x : N$, *then* $\Sigma :\cdot \Gamma \vdash y : N$, *and*
2. *if* x *is a substitution and* $\Sigma :\cdot \Gamma \vdash x \triangleright \Delta$, *then* $\Sigma :\cdot \Gamma \vdash y \triangleright \Delta$.

Proposition 29 (Typing Soundness)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, $\Sigma :\cdot \Gamma \vdash M' : N'$ *and* $M =_{\lambda_{\mathcal{L}\Pi}} M'$, *then there exists a path of well-typed terms to convert* M *and* M', *and*

2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, $\Sigma :\cdot \Gamma \vdash S' \triangleright \Delta$ *and* $S =_{\lambda_{\mathcal{L}\Pi}} S'$, *then there exists a path of well-typed substitutions to convert* S *and* S'.

Proposition 30 (Weak Normalization)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, *then* M *is weakly normalizing, and*

2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then* S *is weakly normalizing.*

Therefore, M *and* S *have at least one* $\lambda_{\mathcal{L}\Pi}$*-normal form.*

Proposition 31 (Church-Rosser) *Let* x *and* y *be such that* $x =_{\lambda_{\mathcal{L}\Pi}} y$. *Then,* x *and* y *are* $\lambda_{\mathcal{L}\Pi}$*-joinable, i.e. there exists* w *such that* $x \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} w$ *and* $y \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} w$, *if*

1. x *is a term,* $\Sigma :\cdot \Gamma_1 \vdash x : N_1$ *and* $\Sigma :\cdot \Gamma_2 \vdash y : N_2$, *or*

2. x *is a substitution,* $\Sigma :\cdot \Gamma_1 \vdash x \triangleright \Delta_1$, $\Sigma :\cdot \Gamma_2 \vdash y \triangleright \Delta_2$ *and* $\Delta_1 =_{\lambda_{\mathcal{L}\Pi}} \Delta_2$.

Proposition 32 (Instantiation Soundness) *Let* M *be a term such that* $\Sigma' :\cdot \Gamma \vdash M : M_1$, *and* Σ *a signature having the form* $\Sigma'. X{:}_{\Gamma} M_1. \Sigma''$,

1. *if* $\vdash \Sigma :\cdot \Delta$, *then* $\vdash \Sigma'. \Sigma''\{X/M\} :\cdot \Delta\{X/M\}$,

2. *if* $\Sigma :\cdot \Delta \vdash M' : M_2$, *then* $\Sigma'. \Sigma''\{X/M\} :\cdot \Delta\{X/M\} \vdash M'\{X/M\} : M_2\{X/M\}$, *and*

3. *if* $\Sigma :\cdot \Delta \vdash S \triangleright \Delta'$, *then* $\Sigma'. \Sigma''\{X/M\} :\cdot \Delta\{X/M\} \vdash S\{X/M\} \triangleright \Delta'\{X/M\}$.

B Soundness Proof

Lemma 33 *If we denote the prefix relation of lists by* $\preceq$, *then the following typing rules are admissible.*

$$\frac{\vdash_c \Sigma_c :\cdot \Delta \quad \Gamma \preceq \Delta}{\vdash_c \Sigma_c :\cdot \Gamma}\,(\text{Undeclare-Var}) \qquad\qquad \frac{\vdash_c \Sigma'_c \quad \Sigma_c \preceq \Sigma'_c}{\vdash_c \Sigma_c}\,(\text{Undeclare-Metavar})$$

Proof. By structural induction on Δ and Σ'_c. □

Lemma 34 *Let* Σ_c *be a valid constrained signature, and* $\Sigma = (\Sigma_c)^{-c}$,

- 1. *if* $\vdash \Sigma :\cdot \Gamma$, *then* $\vdash_c \Sigma_c :\cdot \Gamma$,
  2. *if* $\Sigma :\cdot \Gamma \vdash M : N$, *then* $\Sigma_c :\cdot \Gamma \vdash_c M : N$, *and*
  3. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then* $\Sigma_c :\cdot \Gamma \vdash_c S \triangleright \Delta$,
- *and if* Σ_c *does not have constraints, i.e.* $\Sigma = \Sigma_c$,
  1. *if* $\vdash_c \Sigma_c :\cdot \Gamma$, *then* $\vdash \Sigma_c :\cdot \Gamma$, *then*
  2. *if* $\Sigma_c :\cdot \Gamma \vdash_c M : N$, *then* $\Sigma_c :\cdot \Gamma \vdash M : N$, *and*
  3. *if* $\Sigma_c :\cdot \Gamma \vdash_c S \triangleright \Delta$, *then* $\Sigma_c :\cdot \Gamma \vdash S \triangleright \Delta$.

Proof. By simultaneous induction on typing derivations. □

Lemma 35 *Let Σ_c be a valid constrained signature,*

1. $\vdash_c \Sigma_c :\cdot \Gamma$ *if and only if* $\vdash_c (\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Gamma$,
2. $\Sigma_c :\cdot \Gamma \vdash M : N$ *if and only if* $(\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Gamma \vdash_c M : N$, *and*
3. $\Sigma_c :\cdot \Gamma \vdash S \triangleright \Delta$ *if and only if* $(\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Gamma \vdash_c S \triangleright \Delta$.

Proof. By simultaneous induction on typing derivations. □

Corollary 36 *If* $\vdash_c \Sigma_c$, *then* $\vdash_c (\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}}$.

Lemma 37 *Let Σ_c be the normal form of a valid constrained signature such that $\Sigma_c \neq nil$ and Σ_c is not a failure signature. Then, Σ_c has the form $\Sigma'_c.\ X{:}_\Gamma N.\ \Sigma''_c$, where*

- *Σ'_c does not have constraints,*
- $\vdash \Sigma'_c.\ X{:}_\Gamma N$, *and*
- *N is a $\lambda_{\mathcal{L}\Pi}$-normal form.*

Proof. Since Σ_c is not empty, then it has at least one element. Assume that the first element is a constraint $M =^?_\Gamma M'$. By hypothesis and Corollary 36, we have $\vdash_c \Sigma_c$. By Lemma 33, it happens that $\vdash_c nil.\ M =^?_\Gamma M'$. The last derivation of this judgment has the premises $\Gamma \vdash_c M : N'$ and $\Gamma \vdash_c M' : N'$. Since M, M', N' are well-typed without meta-variables, they are ground, and by Lemma 34, it holds that $\Gamma \vdash M : N'$ and $\Gamma \vdash M' : N'$. Since Σ_c is a signature in normal form, M and N' are not identical. But this is not possible because Σ_c is not a failure context.

Therefore, the first element of Σ_c is not a constraint, and so, Σ_c has the form $\Sigma'_c.\ X{:}_\Gamma N.\ \Sigma''_c$, where Σ'_c does not have constraints. By Lemma 33, we have $\vdash_c \Sigma'_c.\ X{:}_\Gamma N$, and thus, by Lemma 34, $\vdash \Sigma'_c.\ X{:}_\Gamma N$. Since Σ_c is a signature in normal form, N is a $\lambda_{\mathcal{L}\Pi}$-normal form. □

Definition 38 (Rigid Terms) *The set $\mathcal{R}^n_i$ of* atomic rigid terms of arity i and header n, *where $i, n \geq 0$, is defined as follows:*

$$\begin{array}{rcl} \mathcal{R}^0_0 & = & \{Kind, Type\} \\ \mathcal{R}^n_i & = & (\underline{n}\ M_1\ \ldots\ M_i) \quad \textit{where } n \geq 1 \end{array}$$

A term M is rigid, *denoted by $M \in \mathcal{R}$, if and only if M is an abstraction or M is a product, or $M \in \mathcal{R}^n_i$ for some $i, n \geq 0$.*

Definition 39 (Flexible Terms) *The term M is said to be* flexible of arity (i, n) and header X, *where $i, n \geq 0$ and X is a meta-variable, if and only if it has the form $(M'\ M'_1\ \ldots\ M'_m)$ with $m \geq 0$, and*

- *if $i = 0$ and $n = 0$, then $M' = X$,*
- *if $i = 0$ and $n = Suc(n')$, then M' has the form $[\uparrow^n]X$, and*
- *if $i > 0$, then M' has the form $[\uparrow^n \cdot M_{i:N_i} \cdot \ldots \cdot M_{1:N_1}]X$.*

In this case we write $M \in \mathcal{F}_X^{(i,n)}$.

Lemma 40 *If* $\Sigma :\cdot \Gamma \vdash M : N$ *and* N *is a* $\lambda_{\mathcal{L}\Pi}$*-normal form, then* N *has the form* $\Pi_{N_1}. \dots \Pi_{N_i}.N'$*, where* $i \geq 0$*, and either* $N' \in \mathcal{R}_j^n$ *for some* $j, n \geq 0$*, or* $N' \in \mathcal{F}_X^{(j,n)}$ *for some* $j, n \geq 0$ *and meta-variable* X*.*

Proof. By the Sort Soundness property Proposition 26, either $N = \mathit{Kind}$ or $\Sigma :\cdot \Gamma \vdash N : s$. The case $N = \mathit{Kind}$ is trivial because $\mathit{Kind} \in \mathcal{R}_0^0$. In the other case, we proceed by case analysis on N. □

Lemma 41 *Let* θ_c *be a valid grafting of a constrained signature* Σ_c*, and* $\theta = (\theta_c)^{-c}$*,*

- *if* $\vdash_c \Sigma_c :\cdot \Gamma$*, then* $\vdash_c \Sigma_c\theta_c :\cdot \Gamma\theta$*,*
- *if* $\Sigma_c :\cdot \Gamma \vdash_c M : N$*, then* $\Sigma_c\theta_c :\cdot \Gamma\theta \vdash_c M\theta : N\theta$*, and*
- *if* $\Sigma_c :\cdot \Gamma \vdash_c S \triangleright \Delta$*, then* $\Sigma_c\theta_c :\cdot \Gamma\theta \vdash_c S\theta \triangleright \Delta\theta$*.*

Proof. By induction on typing derivations. We use Def. 10, Def. 11, and Proposition 32. □

Lemma 42 *Let* ψ *be a sequential grafting of a constrained signature* Σ_c*,*

- *if* $\vdash_c \Sigma_c :\cdot \Gamma$*, then* $\vdash_c \Sigma_c\psi :\cdot \Gamma\psi$*,*
- *if* $\Sigma_c :\cdot \Gamma \vdash_c N : N'$*, then* $\Sigma_c\psi :\cdot \Gamma\psi \vdash_c N\psi : N'\psi$*, and*
- *if* $\Sigma_c :\cdot \Gamma \vdash_c S \triangleright \Delta$*, then* $\Sigma_c\psi :\cdot \Gamma\psi \vdash_c S\psi \triangleright \Delta\psi$*.*

Proof. We reason by induction on the length of the list ψ and Lemma 41. □

Lemma 43 *Let* Σ_c *be a valid constrained signature,* θ_c *is a valid grafting of* Σ_c *if and only if* θ_c *is a valid grafting of* $(\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}}$*.*

Proof. Let $\theta_c = \{X/_{\Sigma'_c} M\}$. By Lemma 35, $(\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}}$ is a valid constrained signature. Assume that θ_c is a valid grafting of Σ_c, we verify that

1. Meta-variables declared in Σ'_c are not in $(\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}}$. This is true since by Def. 5, Σ_c and $(\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}}$ declare exactly the same meta-variables, and by hypothesis, meta-variables declared in Σ'_c are not in Σ_c,
2. $(\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}}$ has the form $\Sigma_{c_1}.\, X{:}_\Gamma N.\, \Sigma_{c_2}$ where $\Sigma_{c_1}.\, \Sigma'_c :\cdot \Gamma \vdash_c M : N$. By hypothesis, Σ_c has the form $\Sigma'_{c_1}.\, X{:}_\Gamma N'.\, \Sigma_{c'_2}$, thus $(\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}}$ has the form $\Sigma_{c_1}.\, X{:}_\Gamma N.\, \Sigma_{c_2}$, where
   (a) $\Sigma'_{c_1}.\, \Sigma'_c :\cdot \Gamma \vdash_c M : N'$,
   (b) $\Sigma_{c_1} = (\Sigma'_{c_1})\!\downarrow_{\lambda_{\mathcal{L}\Pi}}$, and
   (c) $N' \xrightarrow{\lambda_{\mathcal{L}\Pi}} N$.

   From (a) and (c), we have $\Sigma'_{c_1}.\, \Sigma'_c :\cdot \Gamma \vdash_c M : N$, and using Lemma 35 we conclude $(\Sigma'_{c_1})\!\downarrow_{\lambda_{\mathcal{L}\Pi}}.\, \Sigma'_c :\cdot \Gamma \vdash_c M : N$.

The other sense is similar. □

Lemma 44 *If* θ_c *is a valid grafting of* Σ_c*,* $((\Sigma_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}}\theta_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}} = (\Sigma_c\theta_c)\!\downarrow_{\lambda_{\mathcal{L}\Pi}}$.

Proof. Let $\theta_c = \{X/_{\Sigma'_c} M\}$. By Def. 10, Σ_c has the form $\Sigma_{c_1}.\, X\!:_\Gamma N.\, \Sigma_{c_2}$, and by Def. 11, $\Sigma_c\theta_c = \Sigma_{c_1}.\, \Sigma'_c.\, \Sigma_{c_2}\{X/M\}$. On the other hand, $(\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$ has the form $(\Sigma_{c_1}){\downarrow_{\lambda_{\mathcal{L}\Pi}}}.\, X\!:_\Gamma N'.\, \Sigma'_{c_2}$, where Σ'_{c_2} is a normal form. By Lemma 43, θ_c is a valid grafting of $(\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$ too, and thus by Def. 10, $(\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\theta_c = (\Sigma_{c_1}){\downarrow_{\lambda_{\mathcal{L}\Pi}}}.\, \Sigma'_c.\, \Sigma'_{c_2}\{X/M\}$.

Since $((\Sigma_{c_1}){\downarrow_{\lambda_{\mathcal{L}\Pi}}}.\, \Sigma'_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}} = (\Sigma_{c_1}.\, \Sigma'_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, it suffices to show that the normal form of $\Sigma'_{c_2}\{X/M\}$ is equal to the normal form of $\Sigma_{c_2}\{X/M\}$. We use Def. 5 and the fact that for arbitrary expressions x, y, if $x \xrightarrow{\lambda_{\mathcal{L}\Pi}} y$, then $x\{X/M\} \xrightarrow{\lambda_{\mathcal{L}\Pi}} y\{X/M\}$. □

Lemma 45 *Let Σ_c be a valid constrained signature, if ψ a sequential grafting of $(\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, then $((\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\psi){\downarrow_{\lambda_{\mathcal{L}\Pi}}} = (\Sigma_c\psi){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$.*

Proof. We proceed by induction on the length of ψ. The base case is trivial. Let $\psi = \theta_c, \psi'$,

- $\Rightarrow$ $(\Sigma_c\psi){\downarrow_{\lambda_{\mathcal{L}\Pi}}} = ((\Sigma_c\theta_c)\psi'){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, by Def. 12,
- $\Rightarrow$ $((\Sigma_c\theta_c)\psi'){\downarrow_{\lambda_{\mathcal{L}\Pi}}} = ((\Sigma_c\theta_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\psi'){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, by induction hypothesis,
- $\Rightarrow$ $((\Sigma_c\theta_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\psi'){\downarrow_{\lambda_{\mathcal{L}\Pi}}} = (((\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\theta_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\psi'){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, by Lemma 44,
- $\Rightarrow$ $(((\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\theta_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\psi'){\downarrow_{\lambda_{\mathcal{L}\Pi}}} = (((\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\theta_c)\psi'){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, by induction hypothesis,
- $\Rightarrow$ $(((\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\theta_c)\psi'){\downarrow_{\lambda_{\mathcal{L}\Pi}}} = ((\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\psi){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, by Def. 12.

□

Lemma 46 *Let Σ_c be a valid constrained signature, ψ is a sequential grafting of Σ_c if and only if ψ is a sequential grafting of $(\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$.*

Proof. By induction on the length of ψ. If ψ is the empty list, then the conclusion is trivial by Def. 12. Otherwise, $\psi = \theta_c, \psi'$

- $\Leftrightarrow$ θ_c is a valid grafting of Σ_c and ψ' is a sequential grafting of $\Sigma_c\theta_c$, by Def. 12,
- $\Leftrightarrow$ θ_c is a valid grafting of $(\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, by Lemma 43,
- $\Leftrightarrow$ ψ' is a sequential grafting of $(\Sigma_c\theta_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, by induction hypothesis,
- $\Leftrightarrow$ ψ' is a sequential grafting of $((\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\theta_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, by Lemma 44,
- $\Leftrightarrow$ ψ' is a sequential grafting of $(\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}\theta_c$, by induction hypothesis,
- $\Leftrightarrow$ ψ is a sequential grafting of $(\Sigma_c){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, by Def. 12.

□

Lemma 47 *Failure signatures do not have derivations.*

Proof. Let Σ_c be a valid constrained signature. Note that if $M =^?_\Gamma N \in \Sigma_c$, where M and N are different ground terms in $\lambda_{\mathcal{L}\Pi}$-normal form, then for any sequential grafting ψ of Σ_c, $M =^?_\Gamma N \in \Sigma_c\psi$, and by Lemma 42, $\vdash_c \Sigma_c\psi$. But the normal form of a failure signature is a failure signature, thus $M =^?_\Gamma N \in (\Sigma_c\psi){\downarrow_{\lambda_{\mathcal{L}\Pi}}}$, and $(\Sigma_c\psi){\downarrow_{\lambda_{\mathcal{L}\Pi}}} \neq nil$. □

Proposition 48 *If ψ is a derivation of a valid constrained signature Σ_c, then $\widetilde{\psi}\backslash_{\Sigma_c}$ is a solution of Σ_c.*

Proof. First we prove by induction on the length of ψ that for any context Γ and expression x, $\Gamma\psi = \widetilde{\psi}(\Gamma)$ and $x\psi = \widetilde{\psi}(x)$. Now, since Σ_c is a valid constrained signature we have for any constraint $M_1 =^?_\Gamma M_2$ and meta-variable declaration $X{:}_\Delta N$ in Σ_c

$$\Sigma_c :\cdot \Gamma \vdash_c M_1 : M \tag{1}$$

$$\Sigma_c :\cdot \Gamma \vdash_c M_2 : M \tag{2}$$

$$\Sigma_c :\cdot \Delta \vdash_c X : N \tag{3}$$

Because ψ is a sequential grafting of Σ_c and by Lemma 42, we have

$$\Sigma_c\psi :\cdot \Gamma\psi \vdash_c M_1\psi : M\psi \tag{4}$$

$$\Sigma_c\psi :\cdot \Gamma\psi \vdash_c M_2\psi : M\psi \tag{5}$$

$$\Sigma_c\psi :\cdot \Delta\psi \vdash_c X\psi : N\psi \tag{6}$$

By Lemma 35,

$$(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Gamma\psi \vdash_c M_1\psi : M\psi \tag{7}$$

$$(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Gamma\psi \vdash_c M_2\psi : M\psi \tag{8}$$

$$(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Delta\psi \vdash_c X\psi : M\psi \tag{9}$$

Since ψ is a derivation of Σ_c, we have $(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}} = nil$, but also $\Gamma\psi = \widetilde{\psi}(\Gamma)$, $\Delta\psi = \widetilde{\psi}(\Delta)$, $M_1\psi = \widetilde{\psi}(M_1)$, and $M_2\psi = \widetilde{\psi}(M_2)$. Therefore, $\widetilde{\psi}$ and $\widetilde{\psi}\backslash_{\Sigma_c}$ are solutions of Σ_c. ☐

Proposition 49 (Elementary Graftings) *Let Σ_c be the normal form of a valid constrained signature such that $\Sigma_c \neq nil$ and Σ_c is not a failure signature. If X is a meta-variable in Σ_c such that it is well-typed without constraints, then elementary graftings of X are valid graftings of Σ_c.*

Proof. By Lemma 37, Σ_c has the form $\Sigma_{c_1}.\ X{:}_\Gamma N.\ \Sigma_{c_2}$, where N is a $\lambda_{\mathcal{L}\Pi}$-normal form. Using Lemma 33 and the typing rules, we can verify that

$$\vdash_c \Sigma_{c_1} :\cdot \Gamma \tag{10}$$

$$N = \mathit{Kind} \text{ or } \Sigma_{c_1} :\cdot \Gamma \vdash_c N : s \tag{11}$$

We reason by case analysis on N, and we consider all the elementary graftings of X.

- $N = \mathit{Kind}$ and $\theta_c = \{X/_{nil}\,\mathit{Type}\}$. Trivially, the meta-variables declared in nil are not in Σ_c, and using Eq. 10 with the rule (Type), we get $\Sigma_{c_1} :\cdot \Gamma \vdash_c \mathit{Type} : \mathit{Kind}$. Therefore, θ_c is a valid grafting of Σ_c.

- $N \in \{\mathit{Kind}, \mathit{Type}\}$. For any $N' \in \{\mathit{Kind}, \mathit{Type}\}$, we consider the grafting $\{X/_{\Sigma''_c}\Pi_H.K\}$, where $H \notin \Sigma_c$, $K \notin \Sigma_c$, and $\Sigma''_c = nil.\ H{:}_\Gamma N'.\ K{:}_{\Gamma.H} N$. By Def. 10, it suffices to verify that $\Sigma_{c_1}.\ \Sigma''_c :\cdot \Gamma \vdash_c \Pi_H.K : N$. We consider two cases according to N'.

  - $N' = \mathit{Kind}$. We have the derivation

$$\frac{\text{Eq. } 10}{\vdash_c \Sigma_{c_1}.\ H{:}_\Gamma \mathit{Kind}}\ (\text{Metavar-Decl}_1)$$

– $N' = \mathit{Type}$. We have the derivation

$$\dfrac{\dfrac{\text{Eq. } 10}{\Sigma_{c_1} :\cdot \Gamma \vdash_c \mathit{Type} : \mathit{Kind}}\ \text{(Type)}}{\vdash_c \Sigma_{c_1} .\ H :_\Gamma \mathit{Type}}\ \text{(Metavar-Decl}_2\text{)}$$

In both cases, we have

$$\vdash_c \Sigma_{c_1} .\ H :_\Gamma N' \tag{12}$$

We continue the derivation as follows:

$$\dfrac{\dfrac{\text{Eq. } 12}{\Sigma_{c_1} .\ H :_\Gamma N' :\cdot \Gamma \vdash_c H : N'}\ \text{(Metavar)}}{\vdash_c \Sigma_{c_1} .\ H :_\Gamma N' :\cdot \Gamma.H}\ \text{(Var-Decl)}$$

Now, we consider two cases according to N.

– $N = \mathit{Kind}$. We have the derivation

$$\dfrac{\vdash_c \Sigma_{c_1} .\ H :_\Gamma N' :\cdot \Gamma.H}{\vdash_c \Sigma_{c_1} .\ H :_\Gamma N' .\ K :_{\Gamma.H} \mathit{Kind}}\ \text{(Metavar-Decl}_1\text{)}$$

– $N = \mathit{Type}$. We have the derivation

$$\dfrac{\dfrac{\vdash_c \Sigma_{c_1} .\ H :_\Gamma N' :\cdot \Gamma.H}{\Sigma_{c_1} .\ H :_\Gamma N' :\cdot \Gamma \vdash_c \mathit{Type} : \mathit{Kind}}\ \text{(Type)}}{\vdash_c \Sigma_{c_1} .\ H :_\Gamma N' .\ K :_{\Gamma.H} \mathit{Type}}\ \text{(Metavar-Decl}_2\text{)}$$

In both cases, we have

$$\vdash_c \Sigma_{c_1} .\ H :_\Gamma N' .\ K :_{\Gamma.H} N \tag{13}$$

We continue the derivation as follows:

$$\dfrac{\dfrac{\text{Eq. } 13}{\Sigma_{c_1} .\ H :_\Gamma N' .\ K :_{\Gamma.H} N :\cdot \Gamma.H \vdash_c K : N}\ \text{(Metavar)}}{\Sigma_{c_1} .\ H :_\Gamma N' .\ K :_{\Gamma.H} N :\cdot \Gamma \vdash_c \Pi_H K : N}\ \text{(Prod)}$$

- $N = \Pi_{N_1}.N_2$. We consider the grafting $\{X/_{\Sigma''_c} \lambda_{N_1}.Y\}$, where $Y \notin \Sigma_c$, and $\Sigma''_c = \mathit{nil}.\ Y :_{\Gamma.N_1} N_2$. By Def. 10, it suffices to verify that $\Sigma_{c_1} .\ \Sigma''_c :\cdot \Gamma \vdash_c \lambda_{N_1}.Y : N$. As in the previous case we can derivate

$$\Sigma_{c_1} .\ \Sigma''_c :\cdot \Gamma.n_1 \vdash_c Y : N_2 \tag{14}$$

We conclude by applying the rule (Abs) to Eq. 14 and Eq. 11.

- $\Sigma_1 :\cdot \Gamma \vdash N : s$. For $1 \leq n \leq \text{Length}(\Gamma)$ such that

  1. $\Sigma_1 :\cdot \Gamma \vdash \underline{n} : M$, and
  2. $\Sigma_1 :\cdot \Gamma \vdash M : s$

we consider all the graftings

$$\{X/_{\Sigma''_c.\, N=^?_\Gamma N'} M'\}$$

where $(\Sigma''_c, M', N') \in \chi(\Gamma, \underline{n}, (M)\downarrow_{\lambda_{\mathcal{L}\Pi}})^k$ for some list χ of infinite distinct names of meta-variables not in Σ_c, and $k \geq 0$. First we prove by induction on k that if $(\Sigma''_c, M', N') \in \chi(\Gamma, \underline{n}, (M)\downarrow_{\lambda_{\mathcal{L}\Pi}})^k$, then

$$\Sigma_{c_1}.\, \Sigma''_c :\cdot \Gamma \vdash_c M' : N' \tag{15}$$

and

$$\Sigma_{c_1}.\, \Sigma''_c :\cdot\ \vdash_c N' : s \tag{16}$$

For that we show by induction on typing derivation that the standard π-reduction preserves typing. Now, we have

$$\frac{\Sigma_{c_1}.\, \Sigma''_c :\cdot \Gamma \vdash_c N : s \quad \Sigma_{c_1}.\, \Sigma''_c :\cdot \Gamma \vdash_c N' : s}{\vdash_c \Sigma_{c_1}.\, \Sigma''_c.\, N =^?_\Gamma N'}\ (\text{Constraint})$$

Finally, we conclude with the rule (Conv) that

$$\Sigma_{c_1}.\, \Sigma''_c.\, N =^?_\Gamma N' :\cdot \Gamma \vdash_c M' : N$$

$\square$

Lemma 50 *Let* $\Sigma_{c_0} \xrightarrow{\theta_{c_0}} \Sigma_{c_1} \longrightarrow \ldots \xrightarrow{\theta_{c_{i-1}}} \Sigma_{c_i}$, $i \geq 0$, *be a path of a tree search of* Σ_c *such that* $\Sigma_{c_0} = (\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}}$. *The list of graftings* $\psi = \theta_{c_0}, \ldots, \theta_{c_i}$ *is a sequential grafting of* Σ_c, *and* $\Sigma_{c_i} = (\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}}$.

Proof. By induction on i. The case base is trivial. Assume that $i > 0$ and take $\psi' = \theta_{c_1}, \ldots, \theta_{c_i}$. By construction, θ_{c_0} is an elementary grafting of X in Σ_{c_0}. Thus, by Proposition 49, θ_{c_0} is a valid grafting of Σ_{c_0} and $\Sigma_{c_1} = (\Sigma_{c_0}\theta_{c_0})\downarrow_{\lambda_{\mathcal{L}\Pi}}$ is well-defined. By induction hypothesis, ψ' is a sequential grafting of $\Sigma_{c_0}\theta_{c_0}$, and $\Sigma_{c_i} = (\Sigma_{c_0}\theta_{c_0}\psi')\downarrow_{\lambda_{\mathcal{L}\Pi}} = (\Sigma_{c_0}\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}}$. By Def. 12, ψ is a sequential grafting of $\Sigma_{c_0} = (\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}}$. Therefore, by Lemma 46, ψ is a sequential grafting of Σ_c, and by Lemma 45, $\Sigma_{c_i} = (\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}}$. $\square$

Theorem 51 (Soundness) *Let* $(\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}} \xrightarrow{\psi} nil$ *be a path of a tree search of* Σ_c. *The composition of* ψ *restricted to* Σ_c, *i.e.* $\widetilde{\psi}\backslash_{\Sigma_c}$, *is a solution of* Σ_c.

Proof. By Lemma 50, ψ is a sequential grafting of Σ_c, and $nil = (\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}}$. Therefore, by Def. 13, ψ is a derivation of Σ_c. We conclude with Proposition 48 that $\widetilde{\psi}\backslash_{\Sigma_c}$ is a solution of Σ_c. $\square$