

Principality and Type Inference for Intersection Types Using Expansion Variables*

A. J. Kfoury
Boston University
Boston, MA 02215, U.S.A.
kfoury@cs.bu.edu
<http://www.cs.bu.edu/~kfoury>

J. B. Wells
Heriot-Watt University
EDINBURGH, EH14 4AS, Scotland
jbw@cee.hw.ac.uk
<http://www.cee.hw.ac.uk/~jbw>

August 20, 2003

Abstract

Principality of typings is the property that for each typable term, there is a typing from which all other typings are obtained via some set of operations. Type inference is the problem of finding a typing for a given term, if possible. We define an intersection type system which has principal typings and types exactly the strongly normalizable λ -terms. More interestingly, every finite-rank restriction of this system (using Leivant's first notion of rank) has principal typings and also has decidable type inference. This is in contrast to System F where the finite rank restriction for every finite rank at 3 and above has neither principal typings nor decidable type inference. Furthermore, the notion of principal typings for our system involves only one operation, substitution, rather than several operations (not all substitution-based) as in earlier presentations of principality for intersection types (without rank restrictions). In our system the earlier notion of *expansion* is integrated in the form of *expansion variables*, which are subject to substitution as are ordinary variables. A unification-based type inference algorithm is presented using a new form of unification, β -unification.

*This is an expanded version of a report that appeared in the Proceedings of the 1999 ACM Symposium on *Principles of Programming Languages*, under the title "Principality and Decidable Type Inference for Finite-Rank Intersection Types" [KW99]. This work has been partly supported by EPSRC grants GR/L 36963 and GR/R 41545/01, by NATO grant CRG 971607, by NSF grants 9417382 (CCR), 9806745 (EIA), 9988529 (CCR), 0113193 (ITR), and by Sun Microsystems equipment grant EDUD-7826-990410-US.

Contents

1	Introduction	3
1.1	Background and Motivation	3
1.2	Contributions of This Paper	4
1.3	Future Work	5
1.4	Acknowledgements	5
2	Intersection Types with Expansion Variables	5
2.1	The Type System	6
2.2	Substitution	8
3	Properties of Substitutions	13
3.1	On Types in General	13
3.2	On Well-Named Types Only	15
3.3	Finite-Support Substitutions	21
4	Lambda-Compatible Beta-Unification	22
5	Algorithm for Lambda-Compatible Beta-Unification	27
6	Type Inference Algorithm	38
7	Skeletons and Derivations at Finite Ranks	41
8	Termination and Decidability at Finite Ranks	47
A	Complete Run of the Type-Inference Procedure	54

1 Introduction

1.1 Background and Motivation

The Desire for Polymorphic Type Inference Programming language designers now generally recognize the benefits (as well as the costs!) of strong static typing. Languages such as Haskell [PJHH⁺93], Java [GJS96], and ML [MTHM97] were all designed with strong typing in mind. To avoid imposing an undue burden on the programmer, the compiler is expected to infer as much type information as possible. To avoid rejecting perfectly safe programs, the type inference algorithm should support as much *type polymorphism* as possible. The main options for polymorphism are universal types, written $\forall\alpha.\tau$, and intersection types, written $\sigma \wedge \tau$. (Their duals are the existential types, written $\exists\alpha.\tau$, and union types, written $\sigma \vee \tau$.)

The most popular type inference algorithm is algorithm \mathcal{W} by Damas and Milner [DM82] for the type system commonly called Hindley/Milner which supports polymorphism with a restricted form of universal types. In practice this type system is somewhat inflexible, sometimes forcing the programmer into contortions to convince the compiler that their code is well typed. This has motivated a long search for more expressive type systems with decidable typability. In this search, there have been a great number of negative results, e.g., undecidability of System F [Wel94], finite rank restrictions of F above 3 [KW94], F_{\leq} [Pie94], F_{ω} [Urz97], $F+\eta$ [Wel96], and unrestricted intersection types [Pot80]. Along the way, there have been a few positive results, some extensions of the Damas/Milner approach, but, perhaps more interestingly, some with intersection types.

What are Principal Typings? Many systems with intersection types have had a *principal typings* property. Jim [Jim96] explains the difference with the *principal types* property of the Hindley/Milner system as follows:

Principal Types

Given: a term M typable in type environment A .

There exists: a type σ representing all possible types for M in A .

Principal Typings

Given: a typable term M .

There exists: a judgement $A \vdash M : \tau$ representing all possible typings for M .

Wells [Wel02] gives a system-independent abstract definition of principal typings. Specifically, a typing (A, τ) for a program fragment M is principal for M exactly when that typing is at least as strong as all other typings for M . A typing (A, τ) is *at least as strong as* a typing (A', τ') if and only $A \vdash M : \tau$ being derivable implies $A' \vdash M : \tau'$ is derivable for every M . Jim explains how principal typings support the possibility of true separate compilation and Wells discusses how they help with compositional software analysis.

Principal Typings with Intersection Types Intersection types were first introduced by Coppo and Dezani [CDC80] and, independently, by Pottinger [Pot80].¹ The first system of intersection types for which principal typings was proved (as far as we are aware) was presented by Coppo, Dezani, and Venneri [CDCV80] (a later version is [CDCV81]). Like many systems of intersection types, it is similar to ours in that “ \wedge ” can not appear to the right of “ \rightarrow ” and \wedge -elimination can only occur at λ -term variables. Like our system, this system is restricted so that the binding type of the bound variable of an abstraction must be an intersection of exactly the set of types at which it is used. However, this system differs from ours by allowing one of the types in the intersection type for a bound variable to be used for multiple occurrences of the bound variable. It also has a rule to assign the special type ω (representing the intersection of 0 types) to any term.

There is a general approach for an algorithm for finding principal typings that was followed by Coppo, Dezani, and Venneri for their type system as well as by Ronchi della Rocca and Venneri [RDRV84] and van Bakel [vB93] for other systems of intersection types. In this approach, the principal typing algorithm first finds a normal form (or *approximate* normal form) and then creates a typing for the normal form. A separate proof shows that any typing for the normal form is also a typing for the original term. The algorithms of this

¹Despite appearing earlier, [Sal78] was preceded by the internal report version of [CDC80] and gives credit to Coppo and Dezani for introducing intersection types.

approach are intrinsically impractical, not only due to the expense of normalization but, more importantly, because there is no possibility of a short cut to normalization. The principality of the principal typing is shown using a technique of several different kinds of operations: *expansion* (sometimes called *duplication*), *lifting* (sometimes called *rise*), and *substitution*. The biggest difference with the approach we present in this paper is that we use *expansion variables* to formalize expansion in a much simpler way as part of substitution. This allows our approach to be based on both substitution and unification. This opens the possibility of more efficient algorithms by adding additional (unnecessary) constraints to the unification problem to shortcut the solution, an adaptation we leave to future work.

Sayag and Mauny [SM97, SM96] continue the earlier work cited above, and succeed in defining a simpler notion of principal typings for a system of intersection types. An important difference with our analysis is the continued use of an expansion operation, although considerably simplified from earlier formulations, in part because they restrict attention to λ -terms in normal form. Moreover, their approach is not substitution-based and it is not immediately clear how to extend it to arbitrary λ -terms not in normal form.

The first unification-based approach to principal typing with intersection types is by Ronchi della Rocca [RDR88]. Of course, the general method here will diverge for some terms in the full type system, but a decidable restriction is presented which bounds the height of types. Unfortunately, this approach uses the old, complicated approach to expansion which makes it very difficult to understand. It also appears to have trouble with commutativity and associativity of “ \wedge ”.

Subsequent unification-based approaches to principal typing with intersection types have focused on the *rank-2* restriction of intersection types, using Leivant’s notion of rank [Lei83]. Van Bakel presents a unification algorithm for principal typing for a rank-2 system [vB93]. Later independent work by Jim also attacks the same problem, but with more emphasis on handling practical programming language issues such as recursive definitions, separate compilation, and accurate error messages [Jim96]. Successors to Jim’s method include Banerjee’s [Ban97], which integrates flow analysis, and Jensen’s [Jen98], which integrates strictness analysis. Other approaches to principal typings and type inference with intersection types include [CG92] and [JMZ92].

1.2 Contributions of This Paper

The main contributions of this paper are the following:

- A fully substitution-based notion of principality for a system of intersection types (with or without a rank restriction on types). Expansion variables abstractly represent the possibility of multiple sub-derivations for the same term, supporting a substitution-based approach in place of the old notion of expansion.

This contribution makes the technology of intersection types significantly more accessible to non-theorists. The notions of expansion in earlier literature are so complicated that few but the authors could understand them.

- A unification-based type inference algorithm for intersection types using a novel form of unification, β -unification. The algorithm always returns principal typings when it halts. The algorithm is terminating when restricted to finite-rank types.

This algorithm is the first understandable type inference algorithm for intersection types beyond the rank-2 restriction which does not require that terms first be β -reduced to normal form. Although it may seem that there is quite a bit of material in this report, the vast majority of it exists only to prove properties of the algorithm. The actual algorithm is largely contained in definitions 2.12, 2.14, 2.15, 2.17, 3.8, 3.11, 3.13, 5.1, and 6.1, together with theorem 6.7. This algorithm has been implemented and can currently be found at <http://www.church-project.org/modular/compositional-analysis/>.

- Decidability of type inference and principal typings for the restrictions to every finite rank.

Ours is the first system of intersection types for which this has been shown. At rank 3, our system already types terms not typable in the very powerful system F_ω , e.g., the term

$$(\lambda x.z(x(\lambda f u.f u))(x(\lambda v g.g v)))(\lambda y.y y y) ,$$

which was shown untypable in F_ω by Urzyczyn [Urz97].

1.3 Future Work

Using Intersection Types in Practice This work is carried out in the context of the Church Project (<http://www.church-project.org/>) an ongoing multi-institutional effort investigating the theory and practice of using advanced types and semantics in the design and implementation of programming language systems. The Church Project is actively implementing and evaluating intersection-type-based technology in an ML compiler. A number of practical concerns need to be addressed to finish the task of making the technology presented in this report usable in the overall project effort. In particular, the following tasks are important:

1. Adapt the technology to type systems in which “ \wedge ” is associative, commutative, and idempotent. This will be vital for reducing the space and time complexity of our algorithm, because it will enable the expression of the rank restrictions without requiring an essentially linear flow analysis.
2. Add support for sum types, e.g., booleans and conditionals. This seems likely to require the addition of union types or some form of conditional type.
3. Add support for recursive definitions, e.g., a fix-point operator or letrec bindings. This will significantly complicate the analysis, because it will interfere with the invariant that λ -compatible constraint sets (definition 4.8) have constraints neatly divided into positive and negative types (definition 4.1). Also, polymorphic/polyvariant analysis of recursion is notoriously difficult.
4. Take advantage of the new notion of substitution developed in this report to devise efficient representations of polyvariant program analyses. This is particularly promising.

Theoretical Concerns The work presented here inspires the following possible tasks:

- Investigate the relationship between β -unification and other forms of unification – decidable and undecidable. In particular, investigate the relationship with second-order unification and semi-unification.
- Further develop the meta-theory of β -unification. In particular, investigate conditions under which β -unification (1) satisfies a principality property and (2) is decidable. Use this to develop more sophisticated type inference algorithms.
- Investigate the complexity of the decidable finite-rank restriction of β -unification introduced in section 7. Separately, investigate the complexity of the set of programs typable in the various finite-rank restrictions.

1.4 Acknowledgements

Torben Amtoft, Gang Chen, and Lyn Turbak carefully read drafts of this paper and found bugs. Gang Chen went further and suggested specific bug fixes. Bradley Alan implemented the β -unification constraint solver given in the POPL '99 version of this paper [KW99] and pointed out some errors in examples in the paper. Geoff Washburn completed the implementation by adding the generation of the constraint sets from λ -terms, the use of the results of β -unification to make principal typings and typing derivations, and a nice web-based user interface. The web interface can currently be found at <http://www.church-project.org/modular/compositional-analysis/>. Sébastien Carrier, Geoff Washburn, and Bennett Yates have made helpful comments and have also helped to carry this research forward by starting to address various items of future work mentioned in section 1.3. We are also indebted to the anonymous referees, who made numerous suggestions to clarify our analysis and overall organization of the paper.

2 Intersection Types with Expansion Variables

This section defines a system of intersection types for the λ -calculus with the additional feature of expansion variables. The expansion variables do not affect what is typable; instead they support reasoning about principal typings via a notion of substitution.

Throughout the paper, the notation \vec{X}^n is meta-notation standing for the notation X_1, \dots, X_n . The notation \vec{X} stands for \vec{X}^n for some $n \geq 0$ which either does not matter or is clear from the context. A warning: Some authors use the notation \bar{X} for similar purposes, but in this paper the bar mark on a symbol is *not* used to stand for a sequence.

2.1 The Type System

The set of natural numbers is denoted \mathcal{N} throughout.

Definition 2.1 (λ -Terms). Let x and y range over $\lambda\text{-Var}$, the set of λ -term variables. We use the usual set of λ -terms

$$M, N \in \Lambda ::= x \mid \lambda x.M \mid MN,$$

quotiented by α -conversion as usual, and the usual notion of reduction

$$(\lambda x.M)N \rightarrow_{\beta} M[x := N].$$

As usual, $\text{FV}(M)$ denotes the set of free variables of M . □

The following definition gives a structure to type variable names that will be helpful later when we need to rename them distinctly.

Definition 2.2 (Type Variables and Expansion Variables). The set of *basic type variables* or *basic T-variables* is $\text{TVar}_b = \{a_i \mid i \in \mathcal{N}\}$. The set of *basic expansion variables* or *basic E-variables* is $\text{EVar}_b = \{F_i \mid i \in \mathcal{N}\}$. We assume TVar_b and EVar_b are disjoint sets, and use Var_b to denote the union $\text{EVar}_b \cup \text{TVar}_b$. Let b (possibly decorated) be a metavariable ranging over Var_b .

We use binary strings in $\{0, 1\}^*$, called *offset labels*, to name (and later to rename) variables. If $s, t \in \{0, 1\}^*$, we write $s \cdot t$ for their concatenation. The statement $s \leq t$ holds iff $t = s \cdot s'$ for some $s' \in \{0, 1\}^*$. Let p, q, r, s, t (possibly decorated) be metavariables ranging over $\{0, 1\}^*$.

The set of *type variables* or *T-variables* and the set of *expansion variables* or *E-variables* are, respectively:

$$\text{TVar} = \{a_i^s \mid i \in \mathcal{N}, s \in \{0, 1\}^*\} \quad \text{and} \quad \text{EVar} = \{F_i^s \mid i \in \mathcal{N}, s \in \{0, 1\}^*\}.$$

Let TVar and EVar properly extend TVar_b and EVar_b by taking a_i to be a_i^ε and F_i to be F_i^ε , where ε denotes the empty string. Let $(a_i^s)^t$ denote $a_i^{s \cdot t}$ and let $(F_i^s)^t$ denote $F_i^{s \cdot t}$. Let α and β be metavariables ranging over TVar and let F (in italics) be a metavariable ranging over EVar . For example, if α denotes a_i^s , then α^t denotes $a_i^{s \cdot t}$. We use v (appropriately decorated) as a metavariable ranging over the disjoint union $\text{Var} = \text{EVar} \cup \text{TVar}$. □

Definition 2.3 (Types). Let “ \rightarrow ” and “ \wedge ” be binary type constructors. The set \mathbb{T} of *types* and its subset \mathbb{T}^\rightarrow as well as metavariables over these sets are given as follows:

$$\begin{aligned} \bar{\tau} \in \mathbb{T}^\rightarrow & ::= \alpha \mid (\tau \rightarrow \bar{\tau}) \\ \tau \in \mathbb{T} & ::= \bar{\tau} \mid (\tau \wedge \tau') \mid (F \tau) \end{aligned}$$

Note that $\bar{\tau}$ is only a metavariable over \mathbb{T}^\rightarrow . The letters ρ and σ will be used later to range over certain other subsets of \mathbb{T} . Observe in the tree representation of any $\tau \in \mathbb{T}$ that no “ \wedge ” and no E-variable can occur as the right child of “ \rightarrow ”. We sometimes omit parentheses according to the rule that “ \rightarrow ” and “ \wedge ” associate to the right and the application of an expansion variable (e.g., $F \tau$) has higher precedence than “ \wedge ” which has higher precedence than “ \rightarrow ”. For example, $F \tau_1 \wedge \tau_2 \rightarrow \tau_3 = ((F \tau_1) \wedge \tau_2) \rightarrow \tau_3$. □

Definition 2.4 (Type Environments). A *type environment* is a function A from $\lambda\text{-Var}$ to \mathbb{T} with a finite domain of definition. A type environment may be written as a finite list of pairs, as in

$$x_1 : \tau_1, \dots, x_k : \tau_k$$

for some distinct $x_1, \dots, x_k \in \lambda\text{-Var}$, some $\tau_1, \dots, \tau_k \in \mathbb{T}$ and some $k \geq 0$. If A is a type environment, then $A[x \mapsto \tau]$ is the type environment such that

$$(A[x \mapsto \tau])(y) = \begin{cases} A(y) & \text{if } y \neq x, \\ \tau & \text{if } y = x, \end{cases}$$

and $A \setminus x$ is the type environment such that

$$(A \setminus x)(y) = \begin{cases} A(y) & \text{if } y \neq x, \\ \text{undefined} & \text{if } y = x. \end{cases}$$

If A and B are type environments, then $A \wedge B$ is a new type environment given by:

$$(A \wedge B)(x) = \begin{cases} A(x) \wedge B(x) & \text{if both } A(x) \text{ and } B(x) \text{ defined,} \\ A(x) & \text{if only } A(x) \text{ defined,} \\ B(x) & \text{if only } B(x) \text{ defined,} \\ \text{undefined} & \text{if both } A(x) \text{ and } B(x) \text{ undefined.} \end{cases}$$

If $F \in \text{EVar}$ is an E-variable and A is a type environment, then $F A$ is the type environment such that $(F A)(x) = F(A(x))$. \square

Inference Rules for both Skeletons and Derivations		
VAR $x : \bar{\tau} \vdash x : \bar{\tau}$	$\wedge \frac{A_1 \vdash? M : \tau_1; \quad A_2 \vdash? M : \tau_2}{A_1 \wedge A_2 \vdash_e M : \tau_1 \wedge \tau_2}$	
ABS-I $\frac{A[x \mapsto \tau] \vdash M : \bar{\tau}}{A \vdash (\lambda x.M) : (\tau \rightarrow \bar{\tau})}$	ABS-K $\frac{A \vdash M : \bar{\tau}}{A \vdash (\lambda x.M) : (\bar{\tau}' \rightarrow \bar{\tau})}$ if $x \notin \text{FV}(M)$	
Inference Rule for Skeletons Only		
(S) APP $\frac{A_1 \vdash M : \bar{\tau}; \quad A_2 \vdash? N : \tau}{A_1 \wedge A_2 \vdash M N : \bar{\tau}'}$	Inference Rule for Derivations Only	
	(D) APP $\frac{A_1 \vdash M : \tau \rightarrow \bar{\tau}; \quad A_2 \vdash? N : \tau}{A_1 \wedge A_2 \vdash M N : \bar{\tau}}$	
Inference Rule for both Skeletons and Derivations: Introducing Expansion Variable F		
$F \frac{A \vdash? M : \tau}{F A \vdash_e M : F \tau}$		

Figure 1: Inference Rules of System II.

Definition 2.5 (Judgements, Rule Names, and Skeletons). The sets of *judgements*, *rule names*, and *pre-skeletons* are determined by the following grammar:

$$\begin{aligned} J \in \text{Judg} &::= A \vdash M : \tau \mid A \vdash_e M : \tau \\ R \in \text{Rule} &::= \text{VAR} \mid \text{ABS-K} \mid \text{ABS-I} \mid \text{APP} \mid \wedge \mid F \\ Q \in \text{PSkel} &::= \langle R, J, \vec{Q} \rangle \end{aligned}$$

Judgements formed with the \vdash_e symbol will be used to restrict the \wedge and F rules so these rules are used only for subterms which are the arguments of an application. Observe that a pre-skeleton \mathcal{S} is a rule name R , a final judgement J , and zero or more subskeletons \vec{Q} . The order of the subskeletons is significant. Note that F is a rule name for every $F \in \text{EVar}$.

A *skeleton* \mathcal{S} of System II is a pre-skeleton Q such that, for every sub-pre-skeleton $Q' = \langle R, J, \vec{Q} \rangle$ occurring in Q , it holds that the judgement J is obtained from the end judgements of the pre-skeletons \vec{Q} (whose order is significant) by rule R and rule R is one of the rules for skeletons of System II in figure 1. The order of the pre-skeletons \vec{Q} determines the order in which their end judgements must match the premises of the rule R . A skeleton $\langle R, J, Q_1 \dots Q_n \rangle$ may be written instead as:

$$\frac{Q_1 \quad \dots \quad Q_n}{J} R$$

There are two rules named APP in figure 1: Only rule “(S) APP” is used in skeletons. In interpreting the rules in figure 1, the pattern $A \vdash? M : \tau$ can refer to either $A \vdash M : \tau$ or $A \vdash_e M : \tau$. Observe that the

rule ABS-K is not a special case of the rule ABS-I. This is because there is no rule or other provision for “weakening” (adding redundant type assumptions to a type environment) in our system and therefore, if there is a proof for the judgement $A \vdash M : \tau$ where $x \notin \text{FV}(M)$, then $A(x)$ is not defined. \square

Definition 2.6 (Derivations and Typings). A *derivation* \mathcal{D} of System \mathbb{I} is a skeleton \mathcal{S} such that every use of the rule named “(S) APP” also qualifies as a use of the more restrictive rule named “(D) APP” in figure 1. The set Deriv of derivations is therefore a proper subset of the set Skel of skeletons. Henceforth, all skeletons and derivations belong to System \mathbb{I} .

Let the statement $A \vdash_{\mathbb{I}} M : \tau$ hold iff there exists a derivation \mathcal{D} of System \mathbb{I} whose final judgement is $A \vdash M : \tau$. When this holds, we say that \mathcal{D} is a *typing* for M . A term M is *typable* in System \mathbb{I} iff $A \vdash_{\mathbb{I}} M : \tau$ holds for some A and τ . Note that every typing is a derivation, but not the other way around. \square

The following result is merely what anyone would expect from a system of intersection types formulated without a rule for ω .

Theorem 2.7 (Strong Normalization). *A λ -term M is strongly normalizable (i.e., there is no infinite β -reduction sequence starting from M) if and only if M is typable in System \mathbb{I} .* \square

Proof. A λ -term M is typable in System \mathbb{I} iff M is typable in System $\lambda^{\rightarrow, \wedge}$ of [Kfo99], which is shown there to hold iff M is β -SN. System $\lambda^{\rightarrow, \wedge}$ of [Kfo99] is in fact a non-essential variation of earlier systems of intersection types which were shown to type exactly the strongly normalizable λ -terms: The first (correct) proof of the “if” part in the theorem is due to Pottinger [Pot80], and the first (correct) proof of the “only-if” part, as far as we are aware, is due to Amadio and Curien [AC98]. \square

Corollary 2.8 (Undecidability of Typability). *It is undecidable whether an arbitrarily chosen λ -term M is typable in System \mathbb{I} .* \square

Later in the paper, we will show certain restrictions of System \mathbb{I} to have decidable typability.

REMARK 2.9. System \mathbb{I} does not have the subject reduction property. For example,

$$z : \alpha_2 \rightarrow \alpha_1 \rightarrow \alpha_3, w : \alpha_1 \wedge \alpha_2 \vdash_{\mathbb{I}} (\lambda x. (\lambda y. zyx)x)w : \alpha_3,$$

but

$$z : \alpha_2 \rightarrow \alpha_1 \rightarrow \alpha_3, w : \alpha_1 \wedge \alpha_2 \not\vdash_{\mathbb{I}} (\lambda x. zxx)w : \alpha_3.$$

By theorem 2.7, typability is preserved, so for example:

$$z : \alpha_2 \rightarrow \alpha_1 \rightarrow \alpha_3, w : \alpha_2 \wedge \alpha_1 \vdash_{\mathbb{I}} (\lambda x. zxx)w : \alpha_3.$$

The reason for the lack of subject reduction is that (1) “ \wedge ” is neither associative, commutative, nor idempotent, (2) the implicit \wedge -elimination done by the VAR rule and the way type environments are built together fix the component of an intersection type associated with a particular variable, and (3) there is no provision for weakening (i.e., introducing unneeded type assumptions). If subject reduction is viewed as a means to achieve other goals rather than as a goal by itself, then the lack of subject reduction is not a problem, because derivations of System \mathbb{I} can be easily translated into derivations of more permissive systems of intersection types (see [vB93] for a survey) for which numerous desirable properties have already been verified. The features of System \mathbb{I} which prevent subject reduction make the later analysis of principal typings and type inference much easier. \square

2.2 Substitution

The notion of substitution defined here will be used later in unification for type inference and in establishing a principal typing property for System \mathbb{I} .

Definition 2.10 (Type Contexts). The symbol \square denotes a “hole”. The set \mathbb{T}_\square of *type contexts* and its subset $\mathbb{T}_\square^\rightarrow$ as well as metavariables over these sets are given as follows:

$$\begin{aligned}\bar{\varphi} \in \mathbb{T}_\square^\rightarrow &::= \square \mid \alpha \mid (\varphi \rightarrow \bar{\varphi}) \\ \varphi \in \mathbb{T}_\square &::= \bar{\varphi} \mid (\varphi \wedge \varphi') \mid (F\varphi)\end{aligned}$$

Note that $\bar{\varphi}$ is only a metavariable over $\mathbb{T}_\square^\rightarrow$. If φ has n holes, we write $\#\square(\varphi) = n$ and use $\square^{(1)}, \dots, \square^{(n)}$ to denote these n holes in their order of occurrence in φ from left to right. Care must be applied when inserting $\tau_1, \dots, \tau_n \in \mathbb{T}$ in the holes of φ in order to obtain a type $\tau = \varphi[\tau_1, \dots, \tau_n]$ which is valid according to Definition 2.3. Specifically, if hole $\square^{(i)}$ in φ is to the immediate right of “ \rightarrow ”, then τ_i must be restricted to the subset \mathbb{T}^\rightarrow .

If $\varphi \in \mathbb{T}_\square$, we write $\text{EVar}(\varphi)$ for the set of E-variables occurring in φ , $\text{TVar}(\varphi)$ for the set of T-variables occurring in φ , and $\text{Var}(\varphi)$ for the disjoint union $\text{EVar}(\varphi) \cup \text{TVar}(\varphi)$. \square

Definition 2.11 (Expansions). The set \mathbb{E} of *expansions* is a proper subset of \mathbb{T}_\square , defined by the following grammar:

$$e \in \mathbb{E} ::= \square \mid (e \wedge e') \mid (Fe)$$

In words, an expansion is a type context which mentions no T-variable and no “ \rightarrow ”. \square

Definition 2.12 (Paths in Type Contexts). We define *path* as a partial function which determines the position of $\square^{(i)}$ in φ as a string in $\{\text{L}, \text{R}, 0, 1\}^*$. The definition goes as follows, using a “value” of \perp to indicate that the function is undefined on that input:

$$\begin{aligned}\text{path}(\square^{(i)}, \square) &= \begin{cases} \varepsilon & \text{if } i = 1, \\ \perp & \text{otherwise.} \end{cases} \\ \text{path}(\square^{(i)}, \alpha) &= \perp \\ \text{path}(\square^{(i)}, \varphi \rightarrow \bar{\varphi}) &= \begin{cases} \text{L} \cdot p & \text{if } p = \text{path}(\square^{(i)}, \varphi) \neq \perp, \\ \text{R} \cdot q & \text{if } q = \text{path}(\square^{(i-\#\square(\varphi))}, \bar{\varphi}) \neq \perp, \\ \perp & \text{otherwise.} \end{cases} \\ \text{path}(\square^{(i)}, \varphi \wedge \varphi') &= \begin{cases} 0 \cdot p & \text{if } p = \text{path}(\square^{(i)}, \varphi) \neq \perp, \\ 1 \cdot q & \text{if } q = \text{path}(\square^{(i-\#\square(\varphi))}, \varphi') \neq \perp, \\ \perp & \text{otherwise.} \end{cases} \\ \text{path}(\square^{(i)}, F\varphi) &= \text{path}(\square^{(i)}, \varphi)\end{aligned}$$

Let $\text{paths}(\varphi) = (\text{path}(\square^{(1)}, \varphi), \dots, \text{path}(\square^{(n)}, \varphi))$ where $n = \#\square(\varphi)$. \square

REMARK 2.13. Because an expansion $e \in \mathbb{E}$ is a type context that does not mention “ \rightarrow ”, a path in e is a string in $\{0, 1\}^*$ rather than in $\{\text{L}, \text{R}, 0, 1\}^*$. We thus use binary strings in $\{0, 1\}^*$ for a dual purpose: as paths in expansions and as offset labels to rename variables (see definition 2.2). The coincidence between the two is by design. \square

Definition 2.14 (Variable Renaming). For every $t \in \{0, 1\}^*$ (we do not need to consider the larger set $\{\text{L}, \text{R}, 0, 1\}$), we define a variable renaming $\langle \rangle^t$ from \mathbb{T}_\square to \mathbb{T}_\square , by induction:

1. $\langle \square \rangle^t = \square$.
2. $\langle a_i^s \rangle^t = a_i^{s \cdot t}$ for every $a_i^s \in \text{TVar}$.
3. $\langle \varphi \rightarrow \bar{\varphi} \rangle^t = \langle \varphi \rangle^t \rightarrow \langle \bar{\varphi} \rangle^t$.
4. $\langle \varphi_1 \wedge \varphi_2 \rangle^t = \langle \varphi_1 \rangle^t \wedge \langle \varphi_2 \rangle^t$.
5. $\langle F_i^s \varphi \rangle^t = F_i^{s \cdot t} \langle \varphi \rangle^t$ for every $F_i^s \in \text{EVar}$.

In words, $\langle \varphi \rangle^t$ is obtained from φ by appending t to every offset that is part of a variable name in φ . \square

Definition 2.15 (Substitution on Types and Type Contexts). Because every type is also a type context, it suffices to give the definition for the latter. A *substitution* is a total function $\mathbf{S} : \text{Var} \rightarrow (\mathbb{E} \cup \mathbb{T}^\rightarrow)$ which respects “sorts”, i.e., $\mathbf{S}F \in \mathbb{E}$ for every $F \in \text{EVar}$ and $\mathbf{S}\alpha \in \mathbb{T}^\rightarrow$ for every $\alpha \in \text{TVar}$. Note that $\mathbf{S}(\alpha)$ can not have an E-variable or “ \wedge ” in outermost position. We write $\mathbf{S}F$ instead of $\mathbf{S}(F)$ and $\mathbf{S}\alpha$ instead of $\mathbf{S}(\alpha)$, as long as no ambiguity is introduced. We lift a substitution \mathbf{S} to a function $\bar{\mathbf{S}}$ from \mathbb{T}_\square to \mathbb{T}_\square , by induction:

1. $\bar{\mathbf{S}}\square = \square$.
2. $\bar{\mathbf{S}}\alpha = \mathbf{S}\alpha$.
3. $\bar{\mathbf{S}}(\varphi \rightarrow \bar{\varphi}) = (\bar{\mathbf{S}}\varphi) \rightarrow (\bar{\mathbf{S}}\bar{\varphi})$.
4. $\bar{\mathbf{S}}(\varphi_1 \wedge \varphi_2) = (\bar{\mathbf{S}}\varphi_1) \wedge (\bar{\mathbf{S}}\varphi_2)$.
5. $\bar{\mathbf{S}}(F\varphi) = e[\bar{\mathbf{S}}(\langle\varphi\rangle^{s_1}), \dots, \bar{\mathbf{S}}(\langle\varphi\rangle^{s_n})]$, where $\mathbf{S}F = e$ and $\text{paths}(e) = (s_1, \dots, s_n)$.

When no ambiguity is possible, we write \mathbf{S} for $\bar{\mathbf{S}}$ and $\mathbf{S}\varphi$ for $\bar{\mathbf{S}}(\varphi)$. □

Definition 2.16 (Support of Substitutions). Let $\mathbf{S} : \text{Var} \rightarrow (\mathbb{E} \cup \mathbb{T}^\rightarrow)$ be a substitution. The *non-trivial E-domain* and *non-trivial T-domain* of \mathbf{S} are

$$\text{EDom}(\mathbf{S}) = \{ F \in \text{EVar} \mid \mathbf{S}F \neq F\square \} \text{ and } \text{TDom}(\mathbf{S}) = \{ \alpha \in \text{TVar} \mid \mathbf{S}\alpha \neq \alpha \},$$

respectively. The *non-trivial domain* of \mathbf{S} (or the *support* of \mathbf{S}) is

$$\text{Dom}(\mathbf{S}) = \text{EDom}(\mathbf{S}) \cup \text{TDom}(\mathbf{S}).$$

The notation

$$\{\{ F_1 := e_1, \dots, F_m := e_m, \alpha_1 := \bar{\tau}_1, \dots, \alpha_n := \bar{\tau}_n \}\}$$

denotes a substitution \mathbf{S} with the indicated mappings where $\text{EDom}(\mathbf{S}) = \{F_1, \dots, F_m\}$ and $\text{TDom}(\mathbf{S}) = \{\alpha_1, \dots, \alpha_n\}$. We use the enclosing pair, “ $\{\{$ ” and “ $\}\}$ ” instead of “ $\{$ ” and “ $\}$ ”, as visual help to distinguish \mathbf{S} from constraint sets to which it is applied. Consistent with the preceding notation, $\{\{ \}$ is the substitution such that:

$$\{\{ \}\}(v) = \begin{cases} v\square & \text{if } v \in \text{EVar}, \\ v & \text{if } v \in \text{TVar}. \end{cases} \quad \square$$

Definition 2.17 (Operations on Judgements and Skeletons). The notion of renaming from definition 2.14 is lifted to type environments, judgements, rule names, and skeletons in the obvious way.

The \wedge operator and the operation of applying an E-variable are lifted to skeletons as follows:

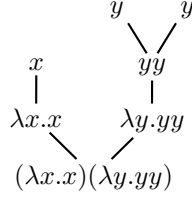
1. $\mathcal{S} \wedge \mathcal{S}' = \langle \wedge, A_1 \wedge A_2 \vdash_e M : \tau_1 \wedge \tau_2, \mathcal{S} \mathcal{S}' \rangle$ if $\mathcal{S} = \langle R_1, A_1 \vdash_{?} M : \tau_1, \bar{\mathcal{S}} \rangle$ and $\mathcal{S}' = \langle R_2, A_2 \vdash_{?} M : \tau_2, \bar{\mathcal{S}}' \rangle$.
2. $F\mathcal{S} = \langle F, F A \vdash_e M : F\tau, \mathcal{S} \rangle$ if $\mathcal{S} = \langle R, A \vdash_{?} M : \tau, \bar{\mathcal{S}} \rangle$.

Using the preceding, the notation for “expansion filling” is lifted to skeletons as follows:

1. $(e \wedge e')[\mathcal{S}_1, \dots, \mathcal{S}_n] = \mathcal{S} \wedge \mathcal{S}'$ if $\#_\square(e) = j$, $e[\mathcal{S}_1, \dots, \mathcal{S}_j] = \mathcal{S}$, and $e'[\mathcal{S}_{j+1}, \dots, \mathcal{S}_n] = \mathcal{S}'$.
2. $(Fe)[\mathcal{S}_1, \dots, \mathcal{S}_n] = F(e[\mathcal{S}_1, \dots, \mathcal{S}_n])$.
3. $\square[\mathcal{S}] = \mathcal{S}$.

The notion of substitution is lifted to type environments so that $\mathbf{S}A$ is the function such that $(\mathbf{S}A)(x) = \mathbf{S}(A(x))$. Substitution is lifted to judgements so that $\mathbf{S}(A \vdash_{?} M : \tau) = \mathbf{S}(A) \vdash_{?} M : \mathbf{S}(\tau)$. Substitution is lifted to skeletons as follows:

i.e., proportional to the size of M_1 :



Applying an arbitrary substitution to \mathcal{S}_1 , we can obtain another skeleton for M_1 . Thus, \mathcal{S}_1 is a scheme for infinitely many skeletons for M_1 .

Associated with \mathcal{S}_1 is a constraint set $\Delta_1 = \{\alpha_1 \rightarrow \alpha_1 \doteq G(\alpha_2 \wedge F\alpha_3 \rightarrow \beta_1) \rightarrow \beta_2, G\alpha_2 \doteq G(F\alpha_3 \rightarrow \beta_1)\}$, produced from M_1 by the Γ algorithm of section 6. (“Constraint sets” and restrictions on them are defined precisely in section 4.) Note that there is one constraint in Δ_1 for each use of the APP rule in \mathcal{S}_1 . A particular substitution \mathbf{S}_1 , obtained from Δ_1 using the Unify algorithm of section 5, is given by:

$$\mathbf{S}_1 = \{[G := \square, \alpha_1 := \tau, \alpha_2 := F\alpha \rightarrow \beta, \beta_2 := \tau]\}$$

where $\tau = ((F\alpha \rightarrow \beta) \wedge F\alpha) \rightarrow \beta$ with $\alpha = \alpha_3$ and $\beta = \beta_1$. Applying substitution \mathbf{S}_1 to skeleton \mathcal{S}_1 , we obtain another skeleton which is now a derivation $\mathcal{D}_1 = \mathbf{S}_1(\mathcal{S}_1)$, depicted in figure 2.

A consequence of the analysis in sections 5 and 6 is that \mathcal{D}_1 is a principal typing for M_1 , i.e., every typing \mathcal{D}' for M_1 is of the form $\mathcal{D}' = \mathbf{S}'(\mathcal{D}_1)$ for some substitution \mathbf{S}' . \square

The skeleton $\mathcal{S}_2 = \text{Skel}(M_2)$ is:

$$\begin{array}{c}
 \frac{\frac{\frac{\text{VAR}}{x : \alpha_1 \vdash x : \alpha_1} \quad \frac{\frac{\text{VAR}}{y : \alpha_2 \vdash y : \alpha_2} \quad F}{y : F\alpha_2 \vdash_e y : F\alpha_2}}{\text{APP}} \quad \frac{\frac{\text{VAR}}{z : \alpha_3 \vdash z : \alpha_3} \quad \frac{\frac{\text{VAR}}{z : \alpha_4 \vdash z : \alpha_4} \quad G}{z : G\alpha_4 \vdash_e z : G\alpha_4}}{\text{APP}}}{\frac{\frac{\frac{x : \alpha_1, y : F\alpha_2 \vdash xy : \beta_1}{\text{ABS-I}} \quad \frac{x : \alpha_1 \vdash \lambda y.xy : F\alpha_2 \rightarrow \beta_1}{\text{ABS-I}}}{\vdash \lambda x.\lambda y.xy : \alpha_1 \rightarrow F\alpha_2 \rightarrow \beta_1}}{\text{ABS-I}} \quad \frac{\frac{\frac{z : \alpha_3 \wedge G\alpha_4 \vdash zz : \beta_2}{\text{ABS-I}} \quad \frac{\vdash \lambda z.zz : \alpha_3 \wedge G\alpha_4 \rightarrow \beta_2}{H}}{\vdash_e \lambda z.zz : H(\alpha_3 \wedge G\alpha_4 \rightarrow \beta_2)}}{\text{APP}}}{\vdash (\lambda x.\lambda y.xy)(\lambda z.zz) : \beta_3}}
 \end{array}$$

Letting $\Delta_2 = \Gamma(M_2)$ and $\mathbf{S}_2 = \text{Unify}(\Delta_2)$, the derivation $\mathcal{D}_2 = \mathbf{S}_2(\mathcal{S}_2)$ is:

$$\begin{array}{c}
 \frac{\frac{\frac{\text{VAR}}{x : \tau_3 \vdash x : \tau_3} \quad \frac{\frac{\frac{\text{VAR}}{y : \tau_1 \vdash y : \tau_1} \quad \frac{\frac{\text{VAR}}{y : \alpha \vdash y : \alpha} \quad G}{y : G\alpha \vdash_e y : G\alpha}}{\wedge}}{\text{APP}}}{\frac{x : \tau_3, y : \tau_2 \vdash xy : \beta}{\text{ABS-I}} \quad \frac{x : \tau_3 \vdash \lambda y.xy : \tau_3}{\text{ABS-I}}}{\vdash \lambda x.\lambda y.xy : \tau_3 \rightarrow \tau_3}}{\text{ABS-I}} \quad \frac{\frac{\frac{\text{VAR}}{z : \alpha \vdash z : \alpha} \quad G}{z : \tau_1 \vdash z : \tau_1} \quad \frac{\frac{\text{VAR}}{z : \alpha \vdash z : \alpha} \quad G}{z : G\alpha \vdash_e z : G\alpha}}{\text{APP}}}{\frac{z : \tau_2 \vdash zz : \beta}{\text{ABS-I}} \quad \frac{\vdash \lambda z.zz : \tau_3}{\text{APP}}}{\vdash (\lambda x.\lambda y.xy)(\lambda z.zz) : \tau_3}}
 \end{array}$$

where τ_1, τ_2 and τ_3 abbreviate the following types:

$$\tau_1 = G\alpha \rightarrow \beta, \quad \tau_2 = \tau_1 \wedge G\alpha = (G\alpha \rightarrow \beta) \wedge G\alpha, \quad \tau_3 = \tau_2 \rightarrow \beta = (G\alpha \rightarrow \beta) \wedge G\alpha \rightarrow \beta.$$

Figure 3: Skeleton \mathcal{S}_2 and derivation \mathcal{D}_2 for $M_2 = (\lambda x.\lambda y.xy)(\lambda z.zz)$.

EXAMPLE 2.22 (A PRINCIPAL TYPING FOR $(\lambda x.\lambda y.xy)(\lambda z.zz)$). Let M_2 denote the λ -term $(\lambda x.\lambda y.xy)(\lambda z.zz)$. Depicted in figure 3 is a skeleton \mathcal{S}_2 for M_2 .

As in example 2.21, the skeleton \mathcal{S}_2 is a particular one, produced from M_2 by the Skel algorithm. Associated with \mathcal{S}_2 is the following constraint set, produced from M_2 by the Γ algorithm of section 6:

$$\Delta_2 = \{\alpha_1 \doteq F\alpha_2 \rightarrow \beta_1, \alpha_1 \rightarrow F\alpha_2 \rightarrow \beta_1 \doteq H(\alpha_3 \wedge G\alpha_4 \rightarrow \beta_2) \rightarrow \beta_3, H\alpha_3 \doteq H(G\alpha_4 \rightarrow \beta_2)\}$$

A particular substitution \mathbf{S}_2 , obtained from Δ_2 using the Unify algorithm of section 5, is given by:

$$\mathbf{S}_2 = \{\{F := \square \wedge G\square, H := \square, \alpha_1 := \tau_3, \alpha_2^0 := \tau_1, \alpha_2^1 := \alpha, \alpha_3 := \tau_1, \beta_1 := \beta, \beta_3 := \tau_3\}$$

where we use the abbreviations $\tau_1 = G\alpha \rightarrow \beta$ and $\tau_3 = ((G\alpha \rightarrow \beta) \wedge G\alpha) \rightarrow \beta$, with $\alpha = \alpha_4$ and $\beta = \beta_2$. Observe that, in this example, \mathbf{S}_2 assigns values to the offsprings α_2^0 and α_2^1 of α_2 , but does not need to assign any particular value to α_2 itself. This follows from the way substitutions are applied “outside-in”, and becomes clear when we consider the action of \mathbf{S}_2 on $F\alpha_2$:

$$\mathbf{S}_2(F\alpha_2) = (\square \wedge G\square)[\mathbf{S}\langle\alpha_2\rangle^0, \mathbf{S}\langle\alpha_2\rangle^1] = (\square \wedge G\square)[\mathbf{S}\alpha_2^0, \mathbf{S}\alpha_2^1] = \mathbf{S}\alpha_2^0 \wedge G(\mathbf{S}\alpha_2^1) = (G\alpha \rightarrow \beta) \wedge G\alpha.$$

Applying substitution \mathbf{S}_2 to skeleton \mathcal{S}_2 , we obtain a new skeleton which is also a derivation $\mathcal{D}_2 = \mathbf{S}_2(\mathcal{S}_2)$, as depicted in figure 3.

A consequence of the analysis in sections 5 and 6 is that \mathcal{D}_2 is a principal typing for M_2 , i.e., every typing \mathcal{D}' for M_2 is of the form $\mathcal{D}' = \mathbf{S}'(\mathcal{D}_2)$ for some substitution \mathbf{S}' . \square

REMARK 2.23. The typings identified as *principal* in this paper are not identical to the things which would be defined to be principal typings following the general definition of Wells [Wel02]. A minor difference is that a *typing* in this paper is an entire derivation of a judgement rather than a pair (A, τ) of the type environment and result type in the final judgement of a derivation. This difference can be bridged by simply using the (A, τ) pair from the final judgement of a typing in this paper. Then every principal typing of this paper is also a principal typing following the general definition. A slightly larger difference is that the word *typing* in this paper is (somewhat arbitrarily) restricted to the case of a derivation where the final type belongs to the restricted set \mathbb{T}^\rightarrow rather than the set \mathbb{T} of all types. So $(\{x : \alpha\}, \alpha)$ is in the final judgement of a principal typing for x according to this paper’s definition, but not $(\{x : F\alpha\}, F\alpha)$. Because each typable term has at least one principal typing, this difference does not cause a problem. \square

3 Properties of Substitutions

The mechanism of substitution in this paper is new. It comes with several peculiarities that set it apart from other forms of substitution in the literature.

3.1 On Types in General

We start with a very simple but fundamental result about substitutions. If \mathbf{S}_1 and \mathbf{S}_2 are substitutions in first-order unification, then $\mathbf{S}_1 = \mathbf{S}_2$ iff $\overline{\mathbf{S}}_1 = \overline{\mathbf{S}}_2$, where $\overline{\mathbf{S}}$ is the lifting of \mathbf{S} (definition 2.15). This is a basic fact, which is completely obvious in first-order unification but requires a little proof in β -unification.

Proposition 3.1 is nowhere invoked directly. But it is used implicitly throughout, because it allows us to use the same symbol \mathbf{S} to denote both a substitution and its lifting, unambiguously.

Proposition 3.1 (Lifting is Injective). *Let $\mathbf{S}_1 : \text{Var} \rightarrow (\mathbb{E} \cup \mathbb{T}^\rightarrow)$ and $\mathbf{S}_2 : \text{Var} \rightarrow (\mathbb{E} \cup \mathbb{T}^\rightarrow)$ be arbitrary substitutions. Then $\mathbf{S}_1 = \mathbf{S}_2$ iff $\overline{\mathbf{S}}_1 = \overline{\mathbf{S}}_2$.* \square

Proof. The implication from left to right is immediate, i.e., lifting \mathbf{S} to $\overline{\mathbf{S}}$ is a uniquely defined operation. For the converse (“lifting is injective”), we assume that $\overline{\mathbf{S}}_1 = \overline{\mathbf{S}}_2$ and we prove that $\mathbf{S}_1 = \mathbf{S}_2$.

Let α be an arbitrary \mathbb{T} -variable. We want to show $\mathbf{S}_1\alpha = \mathbf{S}_2\alpha$. By definition, $\overline{\mathbf{S}}\alpha = \mathbf{S}\alpha$. Hence, if $\overline{\mathbf{S}}_1 = \overline{\mathbf{S}}_2$, then $\mathbf{S}_1\alpha = \mathbf{S}_2\alpha$, as desired.

Let F be an arbitrary \mathbb{E} -variable. We want to prove that $\mathbf{S}_1F = \mathbf{S}_2F$. Take an arbitrary $\alpha \in \text{TVar}$ and consider the action of $\overline{\mathbf{S}}_1$ and $\overline{\mathbf{S}}_2$ on the type $(F\alpha)$. By definition 2.15:

$$\begin{aligned} \overline{\mathbf{S}}_1(F\alpha) &= e_1[\overline{\mathbf{S}}_1\alpha^{s_1}, \dots, \overline{\mathbf{S}}_1\alpha^{s_m}] = e_1[\mathbf{S}_1\alpha^{s_1}, \dots, \mathbf{S}_1\alpha^{s_m}], \\ \text{where } e_1 &= \mathbf{S}_1F \text{ and } (s_1, \dots, s_m) = \text{paths}(e_1). \end{aligned}$$

$$\overline{\mathbf{S}}_2(F\alpha) = e_2[\overline{\mathbf{S}}_2\alpha^{t_1}, \dots, \overline{\mathbf{S}}_2\alpha^{t_n}] = e_2[\mathbf{S}_2\alpha^{t_1}, \dots, \mathbf{S}_2\alpha^{t_n}],$$

where $e_2 = \mathbf{S}_2F$ and $(t_1, \dots, t_n) = \text{paths}(e_2)$.

By hypothesis, $\overline{\mathbf{S}}_1(F\alpha) = \overline{\mathbf{S}}_2(F\alpha)$. Together with the fact that $\mathbf{S}_1\alpha^{s_i} \in \mathbb{T}^\rightarrow$ for every $1 \leq i \leq m$ and $\mathbf{S}_2\alpha^{t_j} \in \mathbb{T}^\rightarrow$ for every $1 \leq j \leq n$, this implies that $e_1 = e_2$, i.e., $\mathbf{S}_1F = \mathbf{S}_2F$ as desired. \square

Much of the difficulty in dealing with substitutions in β -unification results from the distinctive way in which “composition” and “ground composition” of substitutions behave. We next give precise definitions for these two operations.

Definition 3.2 (Composition of Substitutions). Lifting substitutions to functions from \mathbb{T}_\square to \mathbb{T}_\square , as in definition 2.15, allows us to compose them (as functions from \mathbb{T}_\square to \mathbb{T}_\square) and we use the standard symbol “ \circ ”.² Specifically, if the substitutions \mathbf{S}_1 and \mathbf{S}_2 are lifted to functions from \mathbb{T}_\square to \mathbb{T}_\square , then their *composition* is defined by:

$$\mathbf{S}_2 \circ \mathbf{S}_1 = \{ \varphi \mapsto \mathbf{S}_2(\mathbf{S}_1\varphi) \mid \varphi \in \mathbb{T}_\square \}.$$

If \mathbf{S}_1 , \mathbf{S}_2 and \mathbf{S}_3 are arbitrary substitutions, then it is always the case that:

$$\mathbf{S}_3 \circ (\mathbf{S}_2 \circ \mathbf{S}_1) = (\mathbf{S}_3 \circ \mathbf{S}_2) \circ \mathbf{S}_1.$$

This really means $\overline{\mathbf{S}}_3 \circ (\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1) = (\overline{\mathbf{S}}_3 \circ \overline{\mathbf{S}}_2) \circ \overline{\mathbf{S}}_1$, which is the usual associativity of function composition. \square

Definition 3.3 (Ground Composition). If \mathbf{S}_1 and \mathbf{S}_2 are arbitrary substitutions, we define a new substitution $\mathbf{S}_2 \diamond \mathbf{S}_1$, the *ground composition* of \mathbf{S}_2 and \mathbf{S}_1 , by:

$$\mathbf{S}_2 \diamond \mathbf{S}_1 = \{ v \mapsto \mathbf{S}_2(\mathbf{S}_1v) \mid v \in \text{Var} \},$$

which of course can be lifted to a function $\overline{\mathbf{S}_2 \diamond \mathbf{S}_1}$ from \mathbb{T}_\square to \mathbb{T}_\square as in definition 2.15. \square

If $\overline{\mathbf{S}}_1$ and $\overline{\mathbf{S}}_2$ are the functions resulting from lifting \mathbf{S}_1 and \mathbf{S}_2 , it may be expected as elsewhere in unification theory that $\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1 = \overline{\mathbf{S}_2 \diamond \mathbf{S}_1}$, but it is not! This and other subtle issues are illustrated by examples.

EXAMPLE 3.4 (COMPOSITION \neq GROUND COMPOSITION). Consider the substitutions $\mathbf{S}_1 = \{\{\alpha := \beta\}\}$, where $\alpha \neq \beta$, and $\mathbf{S}_2 = \{\{F := \square \wedge \square\}\}$. It is clear that $\mathbf{S}_2 \diamond \mathbf{S}_1 = \{\{F := \square \wedge \square, \alpha := \beta\}\}$. Applying $\mathbf{S}_2 \diamond \mathbf{S}_1$ to the type $F\alpha$, we obtain:

$$(\mathbf{S}_2 \diamond \mathbf{S}_1)(F\alpha) = \alpha^0 \wedge \alpha^1.$$

Applying $\mathbf{S}_2 \circ \mathbf{S}_1$ to the same type $F\alpha$, we obtain:

$$(\mathbf{S}_2 \circ \mathbf{S}_1)(F\alpha) = \mathbf{S}_2(\mathbf{S}_1(F\alpha)) = \beta^0 \wedge \beta^1.$$

Hence, the two operations, “ \diamond ” and “ \circ ”, are not the same. To be explicit about the lifting operation, this says that $\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1 \neq \overline{\mathbf{S}_2 \diamond \mathbf{S}_1}$. \square

EXAMPLE 3.5 (GROUND COMPOSITION NOT ASSOCIATIVE). Consider the following substitutions:

$$\mathbf{S}_1 = \{\{F := GH\square\}\}, \quad \mathbf{S}_2 = \{\{H := \square \wedge \square\}\} \quad \text{and} \quad \mathbf{S}_3 = \{\{G := \square \wedge \square\}\}.$$

A straightforward calculation shows that:

$$\begin{aligned} \mathbf{S}_3 \diamond (\mathbf{S}_2 \diamond \mathbf{S}_1) &= \{\{F := (\square \wedge \square) \wedge (\square \wedge \square), G := \square \wedge \square, H := \square \wedge \square\}\}, \\ (\mathbf{S}_3 \diamond \mathbf{S}_2) \diamond \mathbf{S}_1 &= \{\{F := H^0\square \wedge H^1\square, G := \square \wedge \square, H := \square \wedge \square\}\}. \end{aligned}$$

Clearly, $\mathbf{S}_3 \diamond (\mathbf{S}_2 \diamond \mathbf{S}_1) \neq (\mathbf{S}_3 \diamond \mathbf{S}_2) \diamond \mathbf{S}_1$. \square

²The “composition” of substitutions as such, from Var to $\mathbb{E}\cup\mathbb{T}^\rightarrow$, is actually meaningless, because their domain and codomain are not the same.

EXAMPLE 3.6 (SUBSTITUTIONS NOT CLOSED UNDER COMPOSITION). We give two examples, each illustrating a different point.

1. Consider the substitutions:

$$\mathbf{S}_1 = \{[F := \square \wedge \square, \alpha^1 := \beta]\} \quad \text{and} \quad \mathbf{S}_2 = \{[G := \square \wedge \square, \alpha^1 := \gamma]\},$$

where $\beta \neq \gamma$. While $\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1$ is well-defined as a function from \mathbb{T}_\square to \mathbb{T}_\square , it is not a substitution, i.e., the lifting of a substitution. To see this, consider the action of $\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1$ on the types $F\alpha$ and $G\alpha$:

$$(\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1)(F\alpha) = \overline{\mathbf{S}}_2(\overline{\mathbf{S}}_1(F\alpha)) = \alpha^0 \wedge \beta \quad \text{and} \quad (\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1)(G\alpha) = \overline{\mathbf{S}}_2(\overline{\mathbf{S}}_1(G\alpha)) = \alpha^0 \wedge \gamma.$$

There is no substitution \mathbf{S} which maps both $F\alpha$ to $\alpha^0 \wedge \beta$ and $G\alpha$ to $\alpha^0 \wedge \gamma$. If such a substitution \mathbf{S} existed, it would map two distinct E-variables F and G to the same expansion $\square \wedge \square$ (which is possible) and the same T-variable α^1 to two distinct types β and γ (which is not possible). It follows also that there is no substitution which maps the single type $F\alpha \wedge G\alpha$ to $(\alpha^0 \wedge \beta) \wedge (\alpha^0 \wedge \gamma)$. Thus, posing $\tau = (F\alpha \wedge G\alpha)$, we have $\overline{\mathbf{S}}\tau \neq (\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1)\tau$ for every substitution \mathbf{S} .

2. Let now:

$$\mathbf{S}'_1 = \{[F := \square \wedge (\square \wedge \square), \alpha^{11} := \beta]\} \quad \text{and} \quad \mathbf{S}'_2 = \{[G := \square \wedge \square, \alpha^{11} := \gamma]\},$$

where $\beta \neq \gamma$. Consider the action of $\overline{\mathbf{S}}'_2 \circ \overline{\mathbf{S}}'_1$ on the types $F\alpha$ and $G\alpha^1$:

$$(\overline{\mathbf{S}}'_2 \circ \overline{\mathbf{S}}'_1)(F\alpha) = \overline{\mathbf{S}}'_2(\overline{\mathbf{S}}'_1(F\alpha)) = \alpha^0 \wedge (\alpha^{10} \wedge \beta) \quad \text{and} \quad (\overline{\mathbf{S}}'_2 \circ \overline{\mathbf{S}}'_1)(G\alpha^1) = \overline{\mathbf{S}}'_2(\overline{\mathbf{S}}'_1(G\alpha^1)) = \alpha^{10} \wedge \gamma.$$

By the reasoning used above, there is no substitution which maps $F\alpha \wedge G\alpha^1$ to $(\alpha^0 \wedge (\alpha^{10} \wedge \beta)) \wedge (\alpha^{10} \wedge \gamma)$. Thus, posing $\tau' = (F\alpha \wedge G\alpha^1)$, we have $\overline{\mathbf{S}}\tau' \neq (\overline{\mathbf{S}}'_2 \circ \overline{\mathbf{S}}'_1)\tau'$ for every substitution \mathbf{S} .

We have purposely given two types, τ and τ' , over which the composition of two substitutions is not equivalent to a single substitution. Looking ahead, τ and τ' violate condition 1 and condition 2, respectively, of “well-named” types (definition 3.9). In lemma 3.18, we show that if τ is well-named and $\overline{\mathbf{S}}_1\tau$ is well-named, then there is indeed a substitution \mathbf{S} such that $\overline{\mathbf{S}}\tau = (\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1)\tau$. \square

REMARK 3.7. It is possible to impose restrictions on variable names and the use of substitutions in order to recover the usual properties encountered in other forms of unification, including the following desirable property:

1. $\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1 = \overline{\mathbf{S}_2 \diamond \mathbf{S}_1}$,

for all appropriately restricted substitutions \mathbf{S}_1 and \mathbf{S}_2 , which would imply two other desirable properties:

2. $\mathbf{S}_3 \diamond (\mathbf{S}_2 \diamond \mathbf{S}_1) = (\mathbf{S}_3 \diamond \mathbf{S}_2) \diamond \mathbf{S}_1$, and
3. the function $\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1$ is a substitution, i.e., the lifting of a substitution.

For property 2, the associativity of “ \diamond ” (not always guaranteed, by example 3.5) would follow from the associativity of “ \circ ” (always guaranteed, by definition 3.2). For property 3, as $\mathbf{S}_2 \diamond \mathbf{S}_1$ is a substitution (by definition 3.3), it would follow that $\overline{\mathbf{S}}_2 \circ \overline{\mathbf{S}}_1$ is (the lifting of) a substitution.

However, it does not seem to be worth the effort to guarantee the preceding three properties at some reasonable level of generality. Instead, we are careful in restricting the use of substitutions in order to achieve, at a minimum, the third property above. \square

3.2 On Well-Named Types Only

Several properties of substitutions are true provided they are restricted to “well-named” types. The definition requires the preliminary notion of “E-path”, which also plays an important role in the analysis of later sections.

Definition 3.8 (E-paths). The set EVar^* of all finite sequences of E-variables is also called the *set of E-paths*. We define a function E-path from $\text{Var} \times \mathbb{T}_\square$ to finite subsets of EVar^* . By induction:

1. $\text{E-path}(v, \square) = \emptyset$.
2. $\text{E-path}(v, \alpha) = \begin{cases} \{\varepsilon\} & \text{if } v = \alpha, \\ \emptyset & \text{if } v \neq \alpha. \end{cases}$
3. $\text{E-path}(v, \varphi \rightarrow \bar{\varphi}) = \text{E-path}(v, \varphi) \cup \text{E-path}(v, \bar{\varphi})$.
4. $\text{E-path}(v, \varphi \wedge \varphi') = \text{E-path}(v, \varphi) \cup \text{E-path}(v, \varphi')$.
5. $\text{E-path}(v, F\varphi) = \begin{cases} \{F\vec{G} \mid \vec{G} \in \text{E-path}(v, \varphi)\} & \text{if } v \neq F, \\ \{\varepsilon\} \cup \{F\vec{G} \mid \vec{G} \in \text{E-path}(v, \varphi)\} & \text{if } v = F. \end{cases}$

Let φ be a type context with $n \geq 1$ holes $\square^{(1)}, \dots, \square^{(n)}$ and let $\tau = \varphi[\alpha_1, \dots, \alpha_n]$ where $\alpha_1, \dots, \alpha_n$ are n fresh and distinct T-variables. We define $\text{E-path}(\square^{(i)}, \varphi) = \text{E-path}(\alpha_i, \tau)$ for every $1 \leq i \leq n$. \square

Definition 3.9 (Well Named Types and Well Named Type Contexts). As every type is also a type context, it suffices to write the definition for the latter. We say that a type context $\varphi \in \mathbb{T}_\square$ is *well named* iff both of the following statements hold:

1. For every $v \in \text{Var}(\varphi)$, it holds that $\text{E-path}(v, \varphi) = \{\vec{G}\}$ (a singleton set) where v does not occur in \vec{G} .
2. For all $v^s, v^t \in \text{Var}(\varphi)$ with v basic and $s, t \in \{0, 1\}^*$, if $s \leq t$ then $s = t$.

Informally, the first condition says that, for every (type or expansion) variable v , the sequence of E-variables encountered as we go from the root of φ (viewed as a tree) to any occurrence of v is always the same. Furthermore, E-variables do not nest themselves. If $\text{E-path}(v, \varphi)$ is the singleton set $\{\vec{F}\}$, we can write $\text{E-path}(v, \varphi) = \vec{F}$ without ambiguity.

The second condition says that if a variable v occurs in φ , then no proper offspring of v occurs in φ , where a variable $v^{s \cdot t}$ is called an *offspring* of v^s . Note that types that mention only basic variables automatically satisfy the second condition. \square

REMARK 3.10. Condition 1 in definition 3.9 is the important one and will be recalled repeatedly in proofs later. Condition 2 will be automatically satisfied in the way we set up constraints and in the way we apply substitutions to them, and will play no significant role in the interesting part of our analysis.

When we derive a constraint set $\Gamma(M)$ from a λ -term M in section 6, we will be careful to restrict $\Gamma(M)$ to types over *basic* variables, thus automatically satisfying condition 2. Types over variables that are *not basic* will be introduced only as a result of applying substitutions. In β -unification, if a substitution \mathbf{S} is applied to a type τ , the resulting type $\mathbf{S}\tau$ may mention several distinct renamings v_1, \dots, v_n (“offspring”) of the same variable v in τ . Although we do not do it in this report, it is possible to choose v_1, \dots, v_n to be fresh basic variables, obviating the need to impose condition 2. To simplify the bookkeeping, we follow a different approach, whereby all offsprings in $\mathbf{S}\tau$ of the same variable v are obtained by attaching distinct and incomparable offset labels to v , as in definition 2.15. (The offset labels are *incomparable* as strings in $\{0, 1\}^*$.) In this way we guarantee that condition 2 is satisfied again.

There are various technical implications of condition 2 in definition 3.9 in relation to variable naming. These may be ignored without affecting the reader’s understanding through most of the analysis. Condition 2 in definition 3.9 matters and makes an important difference only in a few places in this section only. \square

In general, the standard composition of two substitutions using “ \circ ” does not produce a substitution, i.e., for substitutions \mathbf{S}_1 and \mathbf{S}_2 , there does *not* necessarily exist a substitution \mathbf{S}_3 such that $\overline{\mathbf{S}_3} = (\overline{\mathbf{S}_2} \circ \overline{\mathbf{S}_1})$. (See example 3.6). To work around this difficulty, we use “ $\otimes_{\mathcal{E}}$ ”, a new binary operation on substitutions which we call “safe composition relative to \mathcal{E} ”, where \mathcal{E} is an environment expressing certain naming constraints. Lemma 3.18 makes the new notion precise. We need a few preliminary definitions and related facts first.

Definition 3.11 (E-Path Environment). Given a well-named type context φ , we form its *E-path environment* as follows:

$$(\mathbf{E}\text{-env}(\varphi))(v) = \begin{cases} \vec{F} & \text{if } \mathbf{E}\text{-path}(v, \varphi) = \{\vec{F}\}, \\ \text{undefined} & \text{if } \mathbf{E}\text{-path}(v, \varphi) = \emptyset. \end{cases}$$

An E-path environment is a partial function $\mathcal{E} : \mathbf{Var} \rightarrow \mathbf{EVar}^*$ that is the result of applying the E-env function to a well-named type context φ . Let \mathcal{E} be a metavariable over E-path environments.

Let \mathcal{E} be an E-path environment, which implies there is a well-named type context φ inducing it, i.e., $\mathcal{E} = \mathbf{E}\text{-env}(\varphi)$. Let \mathbf{S} be a substitution such that $\mathbf{S}\varphi$ is a well-named type context. We define $\mathbf{S}\mathcal{E}$ to be another E-path environment, by setting

$$(\mathbf{S}\mathcal{E})(v) = \begin{cases} \vec{F} & \text{if } \mathbf{E}\text{-path}(v, \mathbf{S}\varphi) = \{\vec{F}\}, \\ \text{undefined} & \text{if } \mathbf{E}\text{-path}(v, \mathbf{S}\varphi) = \emptyset. \end{cases} \quad \square$$

Lemma 3.12 (E-Path Environments Are Prefix-Closed). *Let \mathcal{E} be an E-path environment and let $v \in \mathbf{Var}$. If $\mathcal{E}(v)$ is defined with $\mathcal{E}(v) = \vec{F}G$, then $\mathcal{E}(G)$ is defined with $\mathcal{E}(G) = \vec{F}$.* □

Proof. Immediate from the definitions. □

We give two equivalent definitions of “safe composition”, in 3.13 and 3.14. The first is more compact and convenient to use in some proofs. The second is more constructive and makes explicit that $\mathbf{S}_2 \otimes_{\mathcal{E}} \mathbf{S}_1$, the safe composition of substitutions \mathbf{S}_1 and \mathbf{S}_2 relative to \mathcal{E} , is well-defined as a function.

Definition 3.13 (Safe Composition). Let \mathbf{S}_1 and \mathbf{S}_2 be substitutions, and \mathcal{E} an E-path environment. The *safe composition of \mathbf{S}_1 and \mathbf{S}_2 relative to \mathcal{E}* is a new substitution, written $\mathbf{S}_2 \otimes_{\mathcal{E}} \mathbf{S}_1$ and defined by:

$$(\mathbf{S}_2 \otimes_{\mathcal{E}} \mathbf{S}_1)(v) = \begin{cases} \varphi_j & \text{if } v = \bar{v}^r, \mathcal{E}(\bar{v}) = \vec{F}, e = \mathbf{S}_2(\mathbf{S}_1(\vec{F}\square)) \text{ with } \#_{\square}(e) = n, \\ & r = \text{path}(\square^{(j)}, e) \text{ and } e[\varphi_1, \dots, \varphi_n] = \mathbf{S}_2(\mathbf{S}_1(\vec{F}\bar{v})), \\ \llbracket \cdot \rrbracket(v) & \text{otherwise.} \end{cases}$$

which of course can be lifted to a function $\overline{\mathbf{S}_2 \otimes_{\mathcal{E}} \mathbf{S}_1}$ from \mathbb{T}_{\square} to \mathbb{T}_{\square} as in definition 2.15. □

Definition 3.14 (Algorithmic Definition of Safe Composition). First, define auxiliary functions env-offset and path-to-hole as follows.

$$\text{env-offset}(\mathcal{E})(b, s) = \begin{cases} (s', t) & \text{if } b^{s'} \text{ in } \text{dom}(\mathcal{E}) \text{ and } s' \cdot t = s, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Observe that $\text{env-offset}(\mathcal{E})$ is a well defined function, because for each b and s there is at most one $s' \leq s$ such that $b^{s'} \in \text{dom}(\mathcal{E})$, because \mathcal{E} is generated from some well-named type context φ .

$$\text{path-to-hole}(\varphi)(s) = \begin{cases} i & \text{if } \text{path}(\square^{(i)}, \varphi) = s, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Observe that $\text{path-to-hole}(\varphi)$ is a well defined function, because for any φ and s there is at most one i such that $\text{path}(\square^{(i)}, \varphi) = s$.

Now, define the safe composition operator as follows.

$$\begin{aligned}
(\mathbf{S}_1 \otimes_{\mathcal{E}} \mathbf{S}_2)(v) &= \text{let } b^s = v \\
&\quad \text{in if } \text{env-offset}(\mathcal{E})(b, s) \text{ is defined} \\
&\quad \quad \text{then let } (s', t) = \text{env-offset}(\mathcal{E})(b, s) \\
&\quad \quad \quad v' = b^{s'} \\
&\quad \quad \quad \vec{F} = \mathcal{E}(v') \\
&\quad \quad \quad e = \mathbf{S}_2(\mathbf{S}_1(\vec{F}\square)) \\
&\quad \quad \quad n = \#_{\square}(e) \\
&\quad \quad \text{in if } \text{path-to-hole}(e)(t) \text{ is defined} \\
&\quad \quad \quad \text{then let } i &= \text{path-to-hole}(e)(t) \\
&\quad \quad \quad \quad e[\varphi_1, \dots, \varphi_n] &= \mathbf{S}_2(\mathbf{S}_1(\vec{F}v')) \\
&\quad \quad \quad \quad \quad \text{in } \varphi_i \\
&\quad \quad \quad \quad \text{else } \llbracket \cdot \rrbracket(v) \\
&\quad \quad \text{else } \llbracket \cdot \rrbracket(v)
\end{aligned}$$

□

We next prove three technical results. Lemma 3.15 is used to establish lemmas 3.16 and 3.17, and the latter two are used in the proof of lemma 3.18 and the proof of lemma 5.11.

Lemma 3.15. *Let \mathbf{S} be a substitution and let e be an expansion. Then:*

1. $\mathbf{S}e$ is an expansion.
2. Let $\#_{\square}(e) = k \geq 1$ and $\#_{\square}(\mathbf{S}e) = n \geq k$. Then there are maps a, b and r solely depending on e and \mathbf{S} :

$$\begin{aligned}
a &: \{1, \dots, n\} \rightarrow \{1, \dots, k\}, \\
b &: \{1, \dots, n\} \rightarrow \{1, \dots, n\}, \quad \text{and} \\
r &: \{1, \dots, k\} \times \{1, \dots, n\} \rightarrow \{0, 1\}^*
\end{aligned}$$

such that for all $\varphi_1, \dots, \varphi_k \in \mathbb{T}_{\square}$ it holds that:

$$\mathbf{S}(e[\varphi_1, \dots, \varphi_k]) = (\mathbf{S}e)[\mathbf{S}\langle \varphi_{a(1)} \rangle^{r(a(1), b(1))}, \dots, \mathbf{S}\langle \varphi_{a(n)} \rangle^{r(a(n), b(n))}]$$

where

- $\{\varphi_{a(1)}, \dots, \varphi_{a(n)}\} = \{\varphi_1, \dots, \varphi_k\}$, i.e., a is a total surjective map,
- if $\mathbf{E}\text{-path}(\square^{(i)}, e) = \vec{F}$ for $1 \leq i \leq k$,
then $\text{paths}(\mathbf{S}(\vec{F}\square)) = (r(i, 1), \dots, r(i, n_i))$ for some $n_i \geq 1$,
- $n = n_1 + \dots + n_k$,
- $\{b(j) \mid a(j) = i \text{ and } 1 \leq j \leq n\} = \{1, \dots, n_i\}$ for every $1 \leq i \leq k$. □

Proof. Both parts are by structural induction on e . Part 1 is easy and left to the reader. Consider part 2 only. For the base case $e = \square$ of the induction, the result is straightforward.

Proceeding inductively, suppose the result is true for expansion e and for every expansion whose size does not exceed $\text{size}(e)$. Let $F \in \mathbf{EVar}$ and consider the action of \mathbf{S} on $F e[\varphi_1, \dots, \varphi_k]$ for some arbitrary $\varphi_1, \dots, \varphi_k \in \mathbb{T}_{\square}$. We make a record of what we need to push the argument through:

1. Let $\#_{\square}(\mathbf{S}F) = m \geq 1$ and $\text{paths}(\mathbf{S}F) = (p_1, \dots, p_m)$.
2. For every $\ell \in \{1, \dots, m\}$, let $\#_{\square}(\mathbf{S}\langle e \rangle^{p_{\ell}}) = n_{\ell} \geq k$.

3. By the induction hypothesis, for every $\ell \in \{1, \dots, m\}$ there are maps a_ℓ , b_ℓ and r_ℓ solely depending on $\langle e \rangle^{p_\ell}$ and \mathbf{S} :

$$\begin{aligned} a_\ell &: \{1, \dots, n_\ell\} \rightarrow \{1, \dots, k\}, \\ b_\ell &: \{1, \dots, n_\ell\} \rightarrow \{1, \dots, n_\ell\}, \quad \text{and} \\ r_\ell &: \{1, \dots, k\} \times \{1, \dots, n_\ell\} \rightarrow \{0, 1\}^* \end{aligned}$$

such that for all $\varphi'_1, \dots, \varphi'_k \in \mathbb{T}_\square$ it holds that:

$$\mathbf{S}(\langle e \rangle^{p_\ell} [\varphi'_1, \dots, \varphi'_k]) = (\mathbf{S}(e)^{p_\ell})[\mathbf{S}\langle \varphi'_{a_\ell(1)} \rangle^{r_\ell(a_\ell(1), b_\ell(1))}, \dots, \mathbf{S}\langle \varphi'_{a_\ell(n_\ell)} \rangle^{r_\ell(a_\ell(n_\ell), b_\ell(n_\ell))}]$$

where

- $\{\varphi'_{a_\ell(1)}, \dots, \varphi'_{a_\ell(n_\ell)}\} = \{\varphi'_1, \dots, \varphi'_k\}$,
 - if $\mathbf{E}\text{-path}(\square^{(i)}, \langle e \rangle^{p_\ell}) = \vec{F}$ for $1 \leq i \leq k$,
then $\text{paths}(\mathbf{S}(\vec{F}\square)) = (r_\ell(i, 1), \dots, r_\ell(i, n_{\ell,i}))$ for some $n_{\ell,i} \geq 1$,
 - $n_\ell = n_{\ell,1} + n_{\ell,2} + \dots + n_{\ell,k}$,
 - $\{b_\ell(j) \mid a_\ell(j) = i \text{ and } 1 \leq j \leq n_\ell\} = \{1, \dots, n_{\ell,i}\}$ for every $1 \leq i \leq k$.
4. We also need a general fact about associativity of the operation of placing types in the holes of expansions, namely,

$$\begin{aligned} e_0[e_1[\varphi_{1,1}, \dots, \varphi_{1,n_1}], \dots, e_m[\varphi_{m,1}, \dots, \varphi_{m,n_m}]] &= \\ (e_0[e_1, \dots, e_m])[\varphi_{1,1}, \dots, \varphi_{1,n_1}, \dots, \varphi_{m,1}, \dots, \varphi_{m,n_m}] &\cdot \end{aligned}$$

Based on the preceding observations, it is now straightforward to check the following sequence of equalities, for arbitrary $\varphi_1, \dots, \varphi_k \in \mathbb{T}_\square$:

$$\begin{aligned} \mathbf{S}(Fe[\varphi_1, \dots, \varphi_k]) &= (\mathbf{S}F)[\mathbf{S}\langle e[\varphi_1, \dots, \varphi_k] \rangle^{p_1}, \dots, \mathbf{S}\langle e[\varphi_1, \dots, \varphi_k] \rangle^{p_m}] \\ &= (\mathbf{S}F)[\mathbf{S}(\langle e \rangle^{p_1} [\langle \varphi_1 \rangle^{p_1}, \dots, \langle \varphi_k \rangle^{p_1}]), \dots, \mathbf{S}(\langle e \rangle^{p_m} [\langle \varphi_1 \rangle^{p_m}, \dots, \langle \varphi_k \rangle^{p_m}])] \\ &= (\mathbf{S}F)[\mathbf{S}\langle e \rangle^{p_1} [\mathbf{S}\langle \varphi_{a_1(1)} \rangle^{p_1 \cdot r_1(a_1(1), b_1(1))}, \dots, \mathbf{S}\langle \varphi_{a_1(n_1)} \rangle^{p_1 \cdot r_1(a_1(n_1), b_1(n_1))}], \\ &\quad \vdots \\ &\quad \mathbf{S}\langle e \rangle^{p_m} [\mathbf{S}\langle \varphi_{a_m(1)} \rangle^{p_m \cdot r_m(a_m(1), b_m(1))}, \dots, \mathbf{S}\langle \varphi_{a_m(n_m)} \rangle^{p_m \cdot r_m(a_m(n_m), b_m(n_m))}]] \\ &= (\mathbf{S}(Fe))[\mathbf{S}\langle \varphi_{a_1(1)} \rangle^{p_1 \cdot r_1(a_1(1), b_1(1))}, \dots, \mathbf{S}\langle \varphi_{a_1(n_1)} \rangle^{p_1 \cdot r_1(a_1(n_1), b_1(n_1))}], \\ &\quad \vdots \\ &\quad \mathbf{S}\langle \varphi_{a_m(1)} \rangle^{p_m \cdot r_m(a_m(1), b_m(1))}, \dots, \mathbf{S}\langle \varphi_{a_m(n_m)} \rangle^{p_m \cdot r_m(a_m(n_m), b_m(n_m))}] \end{aligned}$$

The first and second equalities follow from the definitions, the third equality uses the facts collected in 3 above based on the induction hypothesis, and the fourth equality follows from 4. From the right-hand side of the last equality, it is now easy to extract maps a , b and r that satisfy the conclusion of part 2 for the expansion Fe .

The remaining case of the induction is $e = e_1 \wedge e_2$, and we assume the conclusion of part 2 is true for every expansion whose size is strictly smaller than $\text{size}(e)$. This case is straightforward (easier than the preceding case Fe) and left to the reader. \square

Lemma 3.16. *Let \mathbf{S} be an arbitrary substitution and let $\vec{F} \in \mathbf{EVar}^*$. For all $\varphi \in \mathbb{T}_\square$, it is the case that:*

$$\mathbf{S}(\vec{F}\varphi) = (\mathbf{S}(\vec{F}\square))[\mathbf{S}\langle \varphi \rangle^{r_1}, \dots, \mathbf{S}\langle \varphi \rangle^{r_n}]$$

where $\text{paths}(\mathbf{S}(\vec{F}\square)) = (r_1, \dots, r_n)$. \square

Proof. This is a special case of lemma 3.15, obtained by making $k = 1$, $e = \vec{F}\square$, and $\varphi_1 = \varphi$. \square

Lemma 3.17. *Let \mathbf{S}_1 and \mathbf{S}_2 be arbitrary substitutions and let $\vec{F} \in \text{EVar}^*$. By part 1 of lemma 3.15, $\mathbf{S}_1(\vec{F}\square)$ and $\mathbf{S}_2(\mathbf{S}_1(\vec{F}\square))$ are expansions. Let $e_1 = \mathbf{S}_1(\vec{F}\square)$ and $e_2 = \mathbf{S}_2(\mathbf{S}_1(\vec{F}\square))$, with $\#\square(e_1) = k \geq 1$ and $\#\square(e_2) = n \geq k$. Then for all $\varphi \in \mathbb{T}_\square$, it holds that:*

$$\mathbf{S}_2(\mathbf{S}_1(\vec{F}\varphi)) = e_2[\mathbf{S}_2\langle \mathbf{S}_1\langle \varphi \rangle^{q_1} \rangle^{r_1}, \dots, \mathbf{S}_2\langle \mathbf{S}_1\langle \varphi \rangle^{q_n} \rangle^{r_n}]$$

for appropriately defined $q_1, r_1, q_2, r_2, \dots, q_n, r_n \in \{0, 1\}^*$ depending solely on the E-path \vec{F} and the substitutions \mathbf{S}_1 and \mathbf{S}_2 . \square

Proof. We have the following sequence of equalities, using the maps a , b and r as defined in part 2 of lemma 3.15:

$$\begin{aligned} \mathbf{S}_2(\mathbf{S}_1(\vec{F}\varphi)) &= \mathbf{S}_2(e_1[\mathbf{S}_1\langle \varphi \rangle^{p_1}, \dots, \mathbf{S}_1\langle \varphi \rangle^{p_k}]) \\ &= e_2[\mathbf{S}_2\langle \mathbf{S}_1\langle \varphi \rangle^{p_{a(1)}} \rangle^{r(a(1), b(1))}, \dots, \mathbf{S}_2\langle \mathbf{S}_1\langle \varphi \rangle^{p_{a(n)}} \rangle^{r(a(n), b(n))}] \end{aligned}$$

where $\text{paths}(e_1) = (p_1, \dots, p_k)$ and, if $\text{E-path}(\square^{(i)}, e_1) = \vec{F}_i$ for every $1 \leq i \leq k$, then $\text{paths}(\mathbf{S}_2(\vec{F}_i\square)) = (r(i, 1), r(i, 2), \dots, r(i, n_i))$ for some $n_i \geq 1$. The first equality above follows from part 2 of lemma 3.15, posing $e = \vec{F}\square$; more directly, it also follows from lemma 3.16. The second equality above follows from part 2 of lemma 3.15, posing $e = e_1$. The desired conclusion now follows. \square

Lemma 3.18 (Sufficient Condition for Safe Composition). *Let \mathbf{S}_1 and \mathbf{S}_2 be substitutions, φ a type context, and $\mathbf{S} = \mathbf{S}_2 \otimes_{\mathcal{E}} \mathbf{S}_1$ for some E-path environment \mathcal{E} . If $\mathcal{E} \supseteq \text{E-env}(\varphi)$, then $\mathbf{S}(\varphi) = \mathbf{S}_2(\mathbf{S}_1(\varphi))$. \square*

Note that the condition $\mathcal{E} \supseteq \text{E-env}(\varphi)$ implies that φ is well-named.

Proof. Fix the E-path environment \mathcal{E} throughout the proof. The appropriate induction here is on the number $n \geq 0$ of occurrences of “ \rightarrow ” and “ \wedge ” in well-named type contexts φ such that $\mathcal{E} \supseteq \text{E-env}(\varphi)$. The base case of the induction on n involves another nested induction on the length $\ell \geq 0$ of E-paths.

Base case: Type contexts φ for this case have $n = 0$ occurrences of “ \rightarrow ” and “ \wedge ”, and therefore are of the form \square or $\vec{F}G\square$ or $\vec{F}\alpha$, with $\mathcal{E}(G) = \vec{F}$ and $\mathcal{E}(\alpha) = \vec{F}$. If $\varphi = \square$, the desired result is immediate. Consider the case $\varphi = \vec{F}G\square$ only, and omit the entirely similar case $\varphi = \vec{F}\alpha$.

We proceed by a nested induction, on the length $\ell \geq 0$ of \vec{F} . The base case of the nested induction is $\ell = 0$, for which $\varphi = G\square$ and $\mathcal{E}(G) = \varepsilon$. Posing $\bar{v} = G$ in definition 3.13, we obtain in the same definition: $e = \mathbf{S}_2(\mathbf{S}_1\square) = \square$ and $r_j = \text{path}(\square^{(j)}, e) = \varepsilon$, implying that $v = \bar{v}^{r_j} = G$ and also that

$$\mathbf{S}_2(\mathbf{S}_1G) = \mathbf{S}_2(\mathbf{S}_1\bar{v}) = (\mathbf{S}_2 \otimes_{\mathcal{E}} \mathbf{S}_1)(v) = (\mathbf{S}_2 \otimes_{\mathcal{E}} \mathbf{S}_1)(G) = \mathbf{S}(G),$$

which is the desired result.

Consider next the case $\varphi = \vec{F}G\square$ where the length of \vec{F} is $\ell + 1$. If $\mathcal{E} \supseteq \text{E-env}(\varphi)$, then $\mathcal{E}(G) = \vec{F}$. Let $e = \mathbf{S}_2(\mathbf{S}_1(\vec{F}\square))$ with $\#\square(e) = n$ and $\text{paths}(e) = (r_1, \dots, r_n)$. By lemma 3.17,

$$\mathbf{S}_2(\mathbf{S}_1(\vec{F}G\square)) = e[\mathbf{S}_2\langle \mathbf{S}_1\langle G\square \rangle^{q_1} \rangle^{r'_1}, \dots, \mathbf{S}_2\langle \mathbf{S}_1\langle G\square \rangle^{q_n} \rangle^{r'_n}]$$

for appropriately defined $q_1, r'_1, \dots, q_n, r'_n \in \{0, 1\}^*$. Defining $\varphi_j = \mathbf{S}_2\langle \mathbf{S}_1\langle G\square \rangle^{q_j} \rangle^{r'_j}$ for every $1 \leq j \leq n$, this means that $\mathbf{S}_2(\mathbf{S}_1(\vec{F}G\square)) = e[\varphi_1, \dots, \varphi_n]$. For a fixed $j \in \{1, \dots, n\}$, if we pose $v = G^{r_j}$ and $\bar{v} = G$ in definition 3.13, we get:

$$\mathbf{S}G^{r_j} = (\mathbf{S}_2 \otimes_{\mathcal{E}} \mathbf{S}_1)(G^{r_j}) = \varphi_j.$$

The preceding observations imply the following sequence of equalities:

$$\begin{aligned} \mathbf{S}_2(\mathbf{S}_1(\vec{F}G\square)) &= e[\varphi_1, \dots, \varphi_n] && \text{as shown above} \\ &= e[\mathbf{S}G^{r_1}, \dots, \mathbf{S}G^{r_n}] && \text{as shown above} \\ &= (\mathbf{S}(\vec{F}\square))[\mathbf{S}G^{r_1}, \dots, \mathbf{S}G^{r_n}] && \text{because } \mathbf{S}(\vec{F}\square) = \mathbf{S}_2(\mathbf{S}_1(\vec{F}\square)) \text{ by induction hypothesis} \\ &= \mathbf{S}(\vec{F}G\square) && \text{by lemma 3.16} \end{aligned}$$

which is the desired conclusion.

Induction case: Type contexts φ for this case have $n + 1$ occurrences of “ \rightarrow ” and “ \wedge ”, and therefore are of the form $\vec{F}(\varphi_1 \rightarrow \varphi_2)$ or $\vec{F}(\varphi_1 \wedge \varphi_2)$ where each of φ_1 and φ_2 has at most n such occurrences. Consider the case $\vec{F}(\varphi_1 \rightarrow \varphi_2)$ only, and omit the entirely similar case $\vec{F}(\varphi_1 \wedge \varphi_2)$.

Let $\mathbf{S}_1(\vec{F}\square) = e_1$ and $\mathbf{S}_2(\mathbf{S}_1(\vec{F}\square)) = e_2$, with $\text{paths}(e_2) = (r_1, \dots, r_n)$. By the base case of the induction, $\mathbf{S}(\vec{F}\square) = \mathbf{S}_2(\mathbf{S}_1(\vec{F}\square)) = e_2$. For $a \in \{1, 2\}$, by lemma 3.17, we have:

$$\mathbf{S}_2(\mathbf{S}_1(\vec{F}\varphi_a)) = e_2[\mathbf{S}_2\langle \mathbf{S}_1\langle \varphi_a \rangle^{q_1} \rangle^{r'_1}, \dots, \mathbf{S}_2\langle \mathbf{S}_1\langle \varphi_a \rangle^{q_n} \rangle^{r'_n}]$$

for appropriate $q_1, r'_1, \dots, q_n, r'_n \in \{0, 1\}^*$ that depend solely on the E-path \vec{F} and the substitutions \mathbf{S}_1 and \mathbf{S}_2 . Defining $\varphi_{a,j} = \mathbf{S}_2\langle \mathbf{S}_1\langle \varphi_a \rangle^{q_j} \rangle^{r'_j}$ for every $1 \leq j \leq n$, we can write $\mathbf{S}_2(\mathbf{S}_1(\vec{F}\varphi_a)) = e_2[\varphi_{a,1}, \dots, \varphi_{a,n}]$. Hence, we have the following sequence of equalities:

$$\begin{aligned} e_2[\mathbf{S}\langle \varphi_a \rangle^{r_1}, \dots, \mathbf{S}\langle \varphi_a \rangle^{r_n}] &= \mathbf{S}(\vec{F}\varphi_a) && \text{by lemma 3.16} \\ &= \mathbf{S}_2(\mathbf{S}_1(\vec{F}\varphi_a)) && \text{by induction hypothesis} \\ &= e_2[\varphi_{a,1}, \dots, \varphi_{a,n}] && \text{as shown above} \end{aligned}$$

This implies $\mathbf{S}\langle \varphi_a \rangle^{r_j} = \varphi_{a,j}$ for $a = 1, 2$ and $j = 1, \dots, n$. By lemma 3.17, we have:

$$\begin{aligned} \mathbf{S}_2(\mathbf{S}_1(\vec{F}(\varphi_1 \rightarrow \varphi_2))) &= e_2[\mathbf{S}_2\langle \mathbf{S}_1\langle \varphi_1 \rightarrow \varphi_2 \rangle^{q_1} \rangle^{r'_1}, \dots, \mathbf{S}_2\langle \mathbf{S}_1\langle \varphi_1 \rightarrow \varphi_2 \rangle^{q_n} \rangle^{r'_n}] \\ &= e_2[\mathbf{S}_2\langle \mathbf{S}_1\langle \varphi_1 \rangle^{q_1} \rangle^{r'_1} \rightarrow \mathbf{S}_2\langle \mathbf{S}_1\langle \varphi_2 \rangle^{q_1} \rangle^{r'_1}, \dots, \mathbf{S}_2\langle \mathbf{S}_1\langle \varphi_1 \rangle^{q_n} \rangle^{r'_n} \rightarrow \mathbf{S}_2\langle \mathbf{S}_1\langle \varphi_2 \rangle^{q_n} \rangle^{r'_n}] \end{aligned}$$

Finally, the preceding implies the following sequence of equalities:

$$\begin{aligned} \mathbf{S}_2(\mathbf{S}_1(\vec{F}(\varphi_1 \rightarrow \varphi_2))) &= e_2[\varphi_{1,1} \rightarrow \varphi_{2,1}, \dots, \varphi_{1,n} \rightarrow \varphi_{2,n}] && \text{because } \mathbf{S}_2\langle \mathbf{S}_1\langle \varphi_a \rangle^{q_j} \rangle^{r'_j} = \varphi_{a,j} \\ &= e_2[\mathbf{S}\langle \varphi_1 \rangle^{r_1} \rightarrow \mathbf{S}\langle \varphi_2 \rangle^{r_1}, \dots, \mathbf{S}\langle \varphi_1 \rangle^{r_n} \rightarrow \mathbf{S}\langle \varphi_2 \rangle^{r_n}] && \text{because } \varphi_{a,j} = \mathbf{S}\langle \varphi_a \rangle^{r_j} \\ &= (\mathbf{S}(\vec{F}\square))[\mathbf{S}\langle \varphi_1 \rightarrow \varphi_2 \rangle^{r_1}, \dots, \mathbf{S}\langle \varphi_1 \rightarrow \varphi_2 \rangle^{r_n}] && \text{because } e_2 = \mathbf{S}(\vec{F}\square) \\ &= \mathbf{S}(\vec{F}(\varphi_1 \rightarrow \varphi_2)) && \text{by lemma 3.16} \end{aligned}$$

which is the desired conclusion. \square

3.3 Finite-Support Substitutions

When we deal with termination properties of our algorithms in section 5 and 8 we restrict attention to substitutions with finite support (definition 2.16). This is justified by the following lemma.

Lemma 3.19 (Finite-Support Substitutions Suffice). *Let \mathbf{S} be an arbitrary substitution and φ a type context. Then, from the given \mathbf{S} and φ , we can construct a substitution \mathbf{S}_0 such that $\overline{\mathbf{S}_0}\varphi = \overline{\mathbf{S}}\varphi$ with $\text{Dom}(\mathbf{S}_0)$ finite.* \square

Proof. We first define the set $\text{Var}(\varphi, \ell)$ of variables in φ at level ℓ , by induction on $\ell \geq 0$:

$$\begin{aligned} \text{Var}(\varphi, 0) &= \{v \in \text{Var}(\varphi) \mid \text{E-path}(v, \varphi) = \varepsilon\} \\ &\vdots \\ \text{Var}(\varphi, \ell + 1) &= \{v \in \text{Var}(\varphi) \mid \text{E-path}(v, \varphi) \in \text{Var}(\varphi, 0) \cdots \text{Var}(\varphi, \ell)\} \end{aligned}$$

Clearly $\text{Var}(\varphi, \ell)$ is finite for every ℓ . Moreover, there is a least $m \geq 0$ such that:

$$\text{Var}(\varphi) = \text{Var}(\varphi, 0) \cup \text{Var}(\varphi, 1) \cup \cdots \cup \text{Var}(\varphi, m) \quad \text{and} \quad \text{Var}(\varphi, \ell) \neq \emptyset \quad \text{for every } \ell \leq m.$$

The desired substitution \mathbf{S}_0 is defined by:

$$\mathbf{S}_0(v) = \begin{cases} \mathbf{S}(v) & \text{if } v = \bar{v}^p \text{ and } \bar{v} \in \text{Var}(\varphi, \ell) \\ & \text{for some } 0 \leq \ell \leq m \text{ with } p \in \text{paths}(\mathbf{S}(\text{E-path}(\bar{v}, \varphi)\square)), \\ \{\!\! \{ \} \!\!\} (v) & \text{otherwise.} \end{cases}$$

It is clear that $\text{Dom}(\mathbf{S}_0)$ is finite. Moreover, for every $v \in \text{Var}(\varphi)$ with $\text{E-path}(v, \varphi) = \vec{F}$, an easy induction on the length $\ell \geq 0$ of \vec{F} shows that:

$$\begin{aligned} \mathbf{S}(\vec{F}v\Box) &= \mathbf{S}_0(\vec{F}v\Box) & \text{if } v \in \text{EVar}, \\ \mathbf{S}(\vec{F}v) &= \mathbf{S}_0(\vec{F}v) & \text{if } v \in \text{TVar}. \end{aligned}$$

This implies that for every type context $\varphi' \in \mathbb{T}_\Box$ such that $\varphi'[\varphi_1, \dots, \varphi_n] = \varphi$ for some $\varphi_1, \dots, \varphi_n \in \mathbb{T}_\Box$ where $n = \#\Box(\varphi') \geq 0$, it is the case that $\mathbf{S}\varphi' = \mathbf{S}_0\varphi'$. This last assertion is established by a straightforward induction on the size of the “initial fragment” φ' of φ (details of the induction omitted). A special case of φ' is $\varphi' = \varphi$ with $\varphi_1 = \dots = \varphi_n = \Box$, which implies $\mathbf{S}\varphi = \mathbf{S}_0\varphi$. \square

4 Lambda-Compatible Beta-Unification

The problem of β -unification was introduced and shown undecidable by Kfoury in [Kfo99]. This section introduces λ -compatible β -unification, a restriction of β -unification, in order to develop a principality property and in preparation for a unification algorithm presented in section 5.

Definition 4.1 (Positive and Negative Types). We identify two proper subsets \mathbb{R} and \mathbb{S} of \mathbb{T} , which we call the “positive types” and the “negative types”, respectively. We first define \mathbb{R} and \mathbb{S} with polarities inserted, as \mathbb{R}^\rightarrow and \mathbb{S}^\rightarrow , defined simultaneously with \mathbb{R}^\rightarrow and \mathbb{S}^\rightarrow , together with metavariables over these sets, as follows:

$$\begin{aligned} \bar{\rho} \in \widetilde{\mathbb{R}}^\rightarrow &::= +\alpha \mid (\sigma \rightarrow \bar{\rho}) \\ \rho \in \widetilde{\mathbb{R}} &::= \bar{\rho} \mid (+F\bar{\rho}) \\ \bar{\sigma} \in \widetilde{\mathbb{S}}^\rightarrow &::= -\alpha \mid (+F\bar{\rho} \rightarrow \bar{\sigma}) \\ \sigma \in \widetilde{\mathbb{S}} &::= \bar{\sigma} \mid (\sigma \wedge \sigma') \mid (-F\sigma) \end{aligned}$$

We obtain \mathbb{R}^\rightarrow and \mathbb{R} from $\widetilde{\mathbb{R}}^\rightarrow$ and $\widetilde{\mathbb{R}}$, respectively, by omitting all polarities. Similarly we obtain \mathbb{S}^\rightarrow and \mathbb{S} from $\widetilde{\mathbb{S}}^\rightarrow$ and $\widetilde{\mathbb{S}}$. Let $\bar{\rho}$, ρ , $\bar{\sigma}$, and σ also range over \mathbb{R}^\rightarrow , \mathbb{R} , \mathbb{S}^\rightarrow , and \mathbb{S} , respectively.

Note that there is a restriction that exactly one E-variable occurs in each positive position to the left of “ \rightarrow ”, and “ \wedge ” occurs only in negative positions. Note also that the metavariables $\bar{\rho}$ and $\bar{\sigma}$ are restricted to the subsets \mathbb{R}^\rightarrow and \mathbb{S}^\rightarrow , respectively.

If $\rho \in \mathbb{R}$ (resp. $\sigma \in \mathbb{S}$), there is exactly one way of inserting polarities in ρ (resp. σ) so that the resulting type ρ' (resp. σ') with polarities is in $\widetilde{\mathbb{R}}$ (resp. $\widetilde{\mathbb{S}}$). Let $(\rho)^+ \in \widetilde{\mathbb{R}}$ (resp. $(\sigma)^- \in \widetilde{\mathbb{S}}$) be the uniquely defined expression obtained by inserting polarities in $\rho \in \mathbb{R}$ (resp. $\sigma \in \mathbb{S}$). We thus have two well-defined functions: $()^+$ from \mathbb{R} to $\widetilde{\mathbb{R}}$ and $()^-$ from \mathbb{S} to $\widetilde{\mathbb{S}}$. \square

Definition 4.2 (Well-Named Constraint Sets). A *constraint* is an equation of the form $\tau \doteq \tau'$ where $\tau, \tau' \in \mathbb{T}$. An *instance* Δ of β -unification is a finite set of constraints, i.e.,

$$\Delta = \{\tau_1 \doteq \tau'_1, \tau_2 \doteq \tau'_2, \dots, \tau_n \doteq \tau'_n\}$$

We write $\text{EVar}(\Delta)$ for the set $\text{EVar}(\tau_1 \wedge \dots \wedge \tau'_n)$, $\text{TVar}(\Delta)$ for the set $\text{TVar}(\tau_1 \wedge \dots \wedge \tau'_n)$ and $\text{Var}(\Delta)$ for their disjoint union $\text{EVar}(\Delta) \cup \text{TVar}(\Delta)$.

The above constraint set Δ is said to be *well named* iff the type $\tau_1 \wedge \tau'_1 \wedge \dots \wedge \tau_n \wedge \tau'_n$ is well named. Given an arbitrary sequence of E-variables $\vec{F} \in \text{EVar}^*$, we write $\vec{F}\Delta$ to denote the constraint set:

$$\vec{F}\Delta = \{ \vec{F}\tau \doteq \vec{F}\tau' \mid \tau \doteq \tau' \text{ is a constraint in } \Delta \}.$$

We write $\vec{F}(\tau \doteq \tau')$ to stand for the constraint $\vec{F}\tau \doteq \vec{F}\tau'$. \square

Definition 4.3 (Good Constraints). From now on, all generated constraints will be in one of 3 forms:

- (a) $\vec{F}(\bar{\rho} \doteq \bar{\sigma})$ where $\text{Var}(\bar{\rho}) \cap \text{Var}(\bar{\sigma}) = \emptyset$.
- (b) $\vec{F}(G\bar{\rho} \doteq \sigma)$ where $\text{Var}(G\bar{\rho}) \cap \text{Var}(\sigma) = \emptyset$.
- (c) $\vec{F}(e[\bar{\rho}_1, \dots, \bar{\rho}_n] \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n])$ where $e = e_1 \wedge e_2$ and $\text{Var}(\bar{\rho}_1 \wedge \dots \wedge \bar{\rho}_n) \cap \text{Var}(\bar{\sigma}_1 \wedge \dots \wedge \bar{\sigma}_n) = \emptyset$.

In form (c), we impose the restriction $e = e_1 \wedge e_2$ in order to make (c) disjoint from (a). In this way, the 3 forms are mutually exclusive.

We say a constraint is a *good constraint* if it is in one of the 3 forms above. Observe the polarities of types in good constraints: always positive to the left, always negative to the right, of the symbol “ \doteq ”.

For precise statement of concepts and results later, it is convenient to consider an additional form, of which (a) and (b), but not (c), are special cases. This additional form is:

- (d) $\vec{F}(\rho \doteq \sigma)$ where there is no $G \in \text{EVar}$ such that $\rho = G\bar{\rho}$ and $\sigma = G\sigma'$.

In form (d), there are constraints which are not good; this happens when $\text{Var}(\rho) \cap \text{Var}(\sigma) \neq \emptyset$ and/or when $\rho \in \mathbb{R}^-$ and $\sigma \in \mathbb{S} - \mathbb{S}^-$. \square

Definition 4.4 (Outer and Inner Variable Occurrences). In a constraint in one of the forms specified in definition 4.3, an E-variable H is said to have an *outer occurrence* if it occurs in \vec{F} in form (a), in form (b) and in form (d), or if it occurs in $\vec{F}e$ in form (c). An occurrence of H which is not outer is said to be an *inner occurrence*. In words, an “outer” occurrence appears on both sides of the constraint and at the top level. Occurrences of T-variables are always inner; only occurrences of E-variables are differentiated between outer and inner.

Let Δ be a finite set of such constraints. We say $H \in \text{EVar}$ has an *outer* (resp. *inner*) occurrence in Δ if H has an outer (resp. inner) occurrence in a constraint in Δ .

The definition of “outer” and “inner” occurrences of E-variables carries over, in the obvious way, when polarities are inserted in constraints. Thus, in each of the forms in definition 4.3:

- (a) $\vec{F}((\bar{\rho})^+ \doteq (\bar{\sigma})^-)$.
- (b) $\vec{F}(+G(\bar{\rho})^+ \doteq (\sigma)^-)$.
- (c) $\vec{F}(e[(\bar{\rho}_1)^+, \dots, (\bar{\rho}_n)^+] \doteq e[(\bar{\sigma}_1)^-, \dots, (\bar{\sigma}_n)^-])$.
- (d) $\vec{F}((\rho)^+ \doteq (\sigma)^-)$.

No polarities are inserted in the outer \vec{F} . Only inner occurrences are said to be positive or negative. \square

Applying `simplify()` to constraint sets:

- $\text{simplify}(\emptyset) = \emptyset$.
- $\text{simplify}(\{\tau \doteq \tau'\} \cup \Delta) = \text{simplify}(\tau \doteq \tau') \cup \text{simplify}(\Delta)$.
- $\text{simplify}(\tau \doteq \tau') = \begin{cases} F \text{simplify}(\tau_1 \doteq \tau'_1) & \text{if } \tau = F\tau_1 \text{ and } \tau' = F\tau'_1, \\ \text{simplify}(\tau'_1 \doteq \tau_1) \cup \text{simplify}(\tau_2 \doteq \tau'_2) & \text{if } \tau = \tau_1 \rightarrow \tau_2 \text{ and } \tau' = \tau'_1 \rightarrow \tau'_2, \\ \text{simplify}(\tau_1 \doteq \tau'_1) \cup \text{simplify}(\tau_2 \doteq \tau'_2) & \text{if } \tau = \tau_1 \wedge \tau_2 \text{ and } \tau' = \tau'_1 \wedge \tau'_2, \\ \emptyset & \text{if } \tau = \tau', \\ \{\tau \doteq \tau'\} & \text{otherwise.} \end{cases}$

Figure 4: The function `simplify()`.

Lemma 4.5 (Simplification Preserves Good Constraints). *Let Δ be a finite set of good constraints.*

1. Applying the function $\text{simplify}(\)$ defined in figure 4 to Δ , we obtain a finite set $\text{simplify}(\Delta)$ of good constraints, each of which is in one of the following forms:

$$\begin{aligned} (a.1) \quad & \vec{F}(\alpha \doteq \bar{\sigma}) \\ (a.2) \quad & \vec{F}(\bar{\rho} \doteq \alpha) \\ (b) \quad & \vec{F}(G\bar{\rho} \doteq \sigma) \end{aligned}$$

Form (b) here is identical to form (b) in definition 4.3; forms (a.1) and (a.2) are special cases of form (a) in definition 4.3.

2. The set of inner (resp. outer) variable occurrences in $\text{simplify}(\Delta)$ is a subset of the set of inner (resp. outer) variable occurrences in Δ . \square

It is worth noticing that part 1 of this lemma implies that *every* constraint in the result of simplifying a set of good constraints matches one of the rewrite rules in algorithm Unify given in figure 5.

Proof. Define the operation $\text{simplify}_1(\)$ on constraint sets by:

$$\begin{aligned} \text{simplify}_1(\emptyset) &= \emptyset, \\ \text{simplify}_1(\{\tau \doteq \tau'\} \cup \Delta) &= \text{simplify}_1(\tau \doteq \tau') \cup \text{simplify}_1(\Delta), \\ \text{simplify}_1(\tau \doteq \tau') &= \begin{cases} F \text{simplify}_1(\tau_1 \doteq \tau'_1) & \text{if } \tau = F\tau_1 \text{ and } \tau' = F\tau'_1, \\ \{\tau'_1 \doteq \tau_1, \tau_2 \doteq \tau'_2\} & \text{if } \tau = \tau_1 \rightarrow \tau_2 \text{ and } \tau' = \tau'_1 \rightarrow \tau'_2, \\ \{\tau_1 \doteq \tau'_1, \tau_2 \doteq \tau'_2\} & \text{if } \tau = \tau_1 \wedge \tau_2 \text{ and } \tau' = \tau'_1 \wedge \tau'_2, \\ \emptyset & \text{if } \tau = \tau' = \alpha, \\ \{\tau \doteq \tau'\} & \text{otherwise.} \end{cases} \end{aligned}$$

In contrast to $\text{simplify}(\)$, $\text{simplify}_1(\)$ takes apart the arguments of a topmost occurrence only of a type constructor, \rightarrow or \wedge , appearing symmetrically on both sides of the same constraint. This is why the side condition in the 4th case above is $\tau = \tau' = \alpha$ rather than $\tau = \tau'$. With the side condition $\tau = \tau'$ instead, $\text{simplify}_1(\)$ would not be well-defined as a function; for example, it would make $\text{simplify}_1(\tau_1 \rightarrow \tau_2 \doteq \tau_1 \rightarrow \tau_2)$ equal to both $\{\tau_1 \doteq \tau_1, \tau_2 \doteq \tau_2\}$ and \emptyset . This is not a problem with the definition of $\text{simplify}(\)$.

If Δ is a finite set of good constraints, $\text{simplify}(\Delta)$ is obtained by applying $\text{simplify}_1(\)$ to Δ repeatedly. Because Δ is a finite set and each of its types has finite size, this process is bound to terminate, producing a constraint set Δ' such that $\text{simplify}_1(\Delta') = \Delta'$.

Let Δ be a finite set of good constraints. Each constraint in Δ is in form (a) or form (b) or form (c), as described in definition 4.3. We further classify these 3 forms into 5 forms as follows:

$$\begin{aligned} (a.1) \quad & \vec{F}(\alpha \doteq \bar{\sigma}), \\ (a.2) \quad & \vec{F}(\bar{\rho} \doteq \alpha), \\ (a.3) \quad & \vec{F}(\sigma \rightarrow \bar{\rho} \doteq G\bar{\rho}' \rightarrow \bar{\sigma}), \\ (b) \quad & \vec{F}(G\bar{\rho} \doteq \sigma), \\ (c) \quad & \vec{F}(e[\bar{\rho}_1, \dots, \bar{\rho}_n] \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]), \quad \text{where } e = e_1 \wedge e_2. \end{aligned}$$

Forms (a.1), (a.2), and (a.3) completely classifies form (a); forms (a.1) and (a.2) have the special case $\vec{F}(\alpha \doteq \alpha')$ in common, where $\alpha, \alpha' \in \text{TVar}$. The action of $\text{simplify}_1(\)$ in the 5 cases is:

$$\begin{aligned} (a.1) \quad & \text{simplify}_1(\vec{F}(\alpha \doteq \bar{\sigma})) = \{\vec{F}(\alpha \doteq \bar{\sigma})\}, \\ (a.2) \quad & \text{simplify}_1(\vec{F}(\bar{\rho} \doteq \alpha)) = \{\vec{F}(\bar{\rho} \doteq \alpha)\}, \\ (a.3) \quad & \text{simplify}_1(\vec{F}(\sigma \rightarrow \bar{\rho} \doteq G\bar{\rho}' \rightarrow \bar{\sigma})) = \{\vec{F}(\bar{\rho} \doteq \bar{\sigma}), \vec{F}(G\bar{\rho}' \doteq \sigma)\}, \\ (b) \quad & \text{simplify}_1(\vec{F}(G\bar{\rho} \doteq \sigma)) = \{\vec{F}(G\bar{\rho} \doteq \sigma)\}, \\ (c) \quad & \text{simplify}_1(\vec{F}(e[\bar{\rho}_1, \dots, \bar{\rho}_n] \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n])) = \\ & \{\vec{F}(e_1[\bar{\rho}_1, \dots, \bar{\rho}_m] \doteq e_1[\bar{\sigma}_1, \dots, \bar{\sigma}_m]), \vec{F}(e_2[\bar{\rho}_{m+1}, \dots, \bar{\rho}_n] \doteq e_2[\bar{\sigma}_{m+1}, \dots, \bar{\sigma}_n])\}, \end{aligned}$$

where, in (c), we take $e = e_1 \wedge e_2$ with $1 \leq \#_{\square}(e_1) = m < n$ and $\#_{\square}(e_2) = n - m \geq 1$. Thus, only forms (a.3) and (c) are split by $\text{simplify}_1(\cdot)$ into two other smaller constraints. Hence, the process of applying $\text{simplify}_1(\cdot)$ repeatedly stops when there are no constraints left that are in form (a.3) or in form (c). Hence, in a good constraint set Δ' such that $\text{simplify}_1(\Delta') = \Delta'$, every constraint is in form (a.1) or form (a.2) or form (b). This proves part 1 of the lemma.

For part 2 of the lemma, it suffices to show that, given a finite set Δ of good constraints, the set of inner (resp. outer) variable occurrences in $\text{simplify}_1(\Delta)$ is a subset of the set of inner (resp. outer) variable occurrences in Δ . This is a straightforward case analysis, by inspecting each of the forms (a.1), (a.2), (a.3), (b), and (c). \square

Definition 4.6 (Links and Graphs). Let $v, w \in \text{Var}$. A *link* (from v to w) is the pair of v and w written in the form $v \rightsquigarrow w$. Let Δ be a set of good constraints. The *graph* of Δ is a finite set of links.

First consider a set Δ consisting of a single good constraint $\vec{F}(\bar{\rho} \doteq \bar{\sigma})$ of form (a) as specified in definition 4.3. In general, $\bar{\rho}$ and $\bar{\sigma}$ are of the following forms:

$$\bar{\rho} = \sigma_1 \rightarrow \cdots \rightarrow \sigma_m \rightarrow \alpha \quad \text{and} \quad \bar{\sigma} = \rho_1 \rightarrow \cdots \rightarrow \rho_n \rightarrow \beta$$

for some $\sigma_1, \dots, \sigma_m \in \mathbb{S}$ and $\rho_1, \dots, \rho_n \in \mathbb{R} - \mathbb{R}^{\rightarrow}$, with $m, n \geq 0$, and some $\alpha, \beta \in \text{TVar}$. We define:

$$\text{graph}(\Delta) = \begin{cases} \{\alpha \rightsquigarrow \beta\} & \text{if } m = n, \\ \{\alpha \rightsquigarrow w \mid w \in \text{Var}(\rho_{m+1} \rightarrow \cdots \rightarrow \rho_n \rightarrow \beta) \text{ and } \text{E-path}(w, \Delta) = \vec{F}\} & \text{if } m < n, \\ \{v \rightsquigarrow \beta \mid v \in \text{Var}(\sigma_{n+1} \rightarrow \cdots \rightarrow \sigma_m \rightarrow \alpha) \text{ and } \text{E-path}(v, \Delta) = \vec{F}\} & \text{if } m > n. \end{cases}$$

For an arbitrary set Δ of good constraints, we define the graph of Δ as:

$$\text{graph}(\Delta) = \bigcup \{ \text{graph}(\vec{F}\{\bar{\rho} \doteq \bar{\sigma}\}) \mid \vec{F}\{\bar{\rho} \doteq \bar{\sigma}\} \subseteq \Delta \}.$$

If there are no good constraints in Δ of form (a), then $\text{graph}(\Delta) = \emptyset$. (Strictly speaking, if we take $\text{Var}(\Delta)$ as the set of nodes in the graph of Δ , then $\text{graph}(\Delta) = \emptyset$ means that all the nodes in the graph of Δ are isolated, not that there are no nodes in the graph.)

It is important to note that if $v \rightsquigarrow w$ is a link in $\text{graph}(\Delta)$, then $\text{E-path}(v, \Delta) = \text{E-path}(w, \Delta)$: There are no links between variables that have different E-paths. For every $\vec{F} \in \text{EVar}^*$, we define $\text{graph}(\Delta)/\vec{F}$ as:

$$\text{graph}(\Delta)/\vec{F} = \{v \rightsquigarrow w \mid \text{E-path}(v, \Delta) = \text{E-path}(w, \Delta) = \vec{F} \text{ and } v \rightsquigarrow_{\Delta} w\},$$

where we write $v \rightsquigarrow_{\Delta} w$ as a shorthand for “ $v \rightsquigarrow w$ is a link in $\text{graph}(\Delta)$ ”. \square

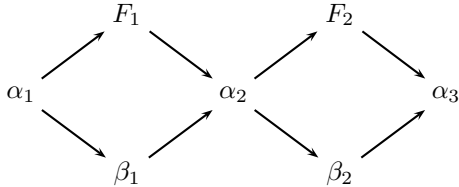
EXAMPLE 4.7 (\rightsquigarrow -CHAINS). Consider the following set Δ of good constraints, all of form (a) in definition 4.3:

$$\Delta = \{\alpha_1 \doteq F_1 \bar{\rho}_1 \rightarrow \beta_1, F_1 \sigma_1 \rightarrow \beta_1 \doteq \alpha_2, \alpha_2 \doteq F_2 \bar{\rho}_2 \rightarrow \beta_2, F_2 \sigma_2 \rightarrow \beta_2 \doteq \alpha_3\},$$

for some $\bar{\rho}_1, \bar{\rho}_2 \in \mathbb{R}^{\rightarrow}$ and $\sigma_1, \sigma_2 \in \mathbb{S}$. Suppose all the variables in $\{\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, F_1, F_2\}$ are pairwise distinct. There are exactly 8 links in $\text{graph}(\Delta)$, namely (separated by blank space for clarity):

$$\alpha_1 \rightsquigarrow F_1 \quad \alpha_1 \rightsquigarrow \beta_1 \quad F_1 \rightsquigarrow \alpha_2 \quad \beta_1 \rightsquigarrow \alpha_2 \quad \alpha_2 \rightsquigarrow F_2 \quad \alpha_2 \rightsquigarrow \beta_2 \quad F_2 \rightsquigarrow \alpha_3 \quad \beta_2 \rightsquigarrow \alpha_3$$

A graphic representation of $\text{graph}(\Delta)$ is:



For the Δ under consideration, $\text{graph}(\Delta)$ is acyclic. Changing, say, α_3 to α_1 introduces cycles, which violates condition (D) for λ -compatibility below. An instance of a \rightsquigarrow -chain in $\text{graph}(\Delta)$ (read as “link-chain” or as “chain in $\text{graph}(\Delta)$ ”, trying to avoid the overloaded word “path” in this paper) is:

$$\alpha_1 \rightsquigarrow F_1 \rightsquigarrow \alpha_2 \rightsquigarrow \beta_2 \rightsquigarrow \alpha_3$$

This is one of 4 possible \rightsquigarrow -chains in $\text{graph}(\Delta)$ of maximal length. When $\text{graph}(\Delta)$ is acyclic, all the variables appearing along a \rightsquigarrow -chain are guaranteed to be pairwise distinct. \square

Definition 4.8 (λ -Compatibility). A constraint set Δ is λ -compatible iff Δ is finite of the form:

$$\Delta = \{\vec{F}_1(\rho_1 \doteq \sigma_1), \dots, \vec{F}_n(\rho_n \doteq \sigma_n)\}$$

where $\vec{F}_i(\rho_i \doteq \sigma_i)$ is a good constraint of form (a) or form (b) as specified in definition 4.3, for every $1 \leq i \leq n$, and moreover Δ satisfies all of the following conditions:

- (A) Δ is well named.
- (B) Every expansion variable $F \in \text{EVar}$ has at most one inner positive occurrence in Δ , i.e., $+F$ occurs at most once in $\pm\Delta$, where $\pm\Delta$ is obtained by inserting polarities in Δ :

$$\pm\Delta = \{\vec{F}_1((\rho_1)^+ \doteq (\sigma_1)^-), \dots, \vec{F}_n((\rho_n)^+ \doteq (\sigma_n)^-)\}$$

- (C) Every type variable $\alpha \in \text{TVar}$ occurs at most twice in Δ . And if it occurs twice, it occurs once positively as $+\alpha$ and once negatively as $-\alpha$ in the constraint set $\pm\Delta$.
- (D) $\text{graph}(\Delta)$ is acyclic.

We use the name “ λ -compatible” because, as shown in lemma 6.2, every constraint set induced by a λ -term satisfies the conditions above. \square

REMARK 4.9. Condition (D) in definition 4.8 is the least transparent and plays a role in the proof of lemma 5.7 only: It imposes a technical restriction on a constraint set Δ , whose sole purpose is to guarantee “ $\text{Var}(\rho) \cap \text{Var}(\sigma) = \emptyset$ for every $\vec{F}(\rho \doteq \sigma) \in \Delta$ ” is an invariant of algorithm `Unify` developed in section 5. The requirement that $\text{Var}(\rho) \cap \text{Var}(\sigma) = \emptyset$ for every $\vec{F}(\rho \doteq \sigma) \in \Delta$ is not strong enough by itself to be preserved by every rewrite step of the algorithm. \square

EXAMPLE 4.10 (WHY CONDITION (D) IS INCLUDED). Consider the following set Δ of constraints:

$$\Delta = \{\alpha \doteq F\beta \rightarrow \bar{\sigma}, F\beta \rightarrow \bar{\rho} \doteq \alpha\},$$

where $\bar{\sigma} \in \mathbb{S}^{\rightarrow}$ and $\bar{\rho} \in \mathbb{R}^{\rightarrow}$, with $\text{Var}(\bar{\rho}) \cap \text{Var}(\bar{\sigma}) = \emptyset$ and $\text{Var}(\bar{\rho} \wedge \bar{\sigma}) \cap \{\alpha, \beta, F\} = \emptyset$. Then Δ is a set of good constraints, all of form (a) as specified in definition 4.3. We can choose $\bar{\sigma}$ and $\bar{\rho}$ so that conditions (A), (B) and (C) in definition 4.8 are satisfied. However, it is easy to see that no choice of $\bar{\sigma}$ and $\bar{\rho}$ will satisfy (D), because $\text{graph}(\Delta)$ always includes the \curvearrowright -chain $\alpha \curvearrowright F \curvearrowright \alpha$ which is a cycle.

Algorithm `Unify` in section 5 will substitute the type $F\beta \rightarrow \bar{\sigma}$ for α in the second constraint, or the type $F\beta \rightarrow \bar{\rho}$ for α in the first constraint, producing the same constraint set Δ' in both cases, namely:

$$\Delta' = \{F\beta \rightarrow \bar{\rho} \doteq F\beta \rightarrow \bar{\sigma}\},$$

which still satisfies (A), (B) and (C), but not (D) again, as $F \curvearrowright F$ is a cycle (a self-loop in this case). The resulting constraint in Δ' is not good. \square

Lemma 4.11 (Simplification Preserves λ -Compatibility). *If Δ is a λ -compatible constraint set, then so is $\text{simplify}(\Delta)$.* \square

Proof. Because $\Delta_0 = \Delta$ is λ -compatible, every constraint in Δ_0 is a good constraint of form (a) or form (b). By lemma 4.5, all the constraints in $\text{simplify}(\Delta_0)$ are good constraints of form (a.1) or (a.2) or (b). We next show that $\text{simplify}(\Delta_0)$ satisfies conditions (A), (B), (C) and (D), in definition 4.8. It suffices to show that $\text{simplify}_1(\Delta_0)$ satisfies (A), (B), (C) and (D), where $\text{simplify}_1(\cdot)$ is the function defined in the proof of lemma 4.5.

Following the case analysis in the proof of lemma 4.5, every constraint in Δ_0 is a good constraint of form (a.1), or form (a.2), or form (a.3), or form (b), but not of form (c), because Δ_0 is a λ -compatible (and not only good) constraint set.³ If Δ_0 satisfies conditions (A), (B) and (C), then it is immediate that $\text{simplify}_1(\Delta_0)$ also satisfies (A), (B) and (C).

³Starting from a λ -compatible constraint set Δ , which does not include good constraints of form (c), algorithm `Unify` developed in section 5 applied to Δ may generate good constraints of form (c), in addition to form (a) and form (b).

It remains to check that $\text{simplify}_1(\Delta_0)$ satisfies (D). Assume that $\text{simplify}_1(\Delta_0) \neq \Delta_0$, otherwise there is nothing to prove. There is exactly one constraint in Δ_0 , namely, a good constraint of form (a.3), which is split into two constraints, in order to produce $\text{simplify}_1(\Delta_0)$. Suppose $\text{simplify}_1(\Delta_0) = \Delta_1$ and:

$$\Delta_0 = \Delta \cup \{\vec{F}(\sigma \rightarrow \bar{\rho} \doteq G\bar{\rho}' \rightarrow \bar{\sigma})\} \quad \text{and} \quad \Delta_1 = \Delta \cup \{\vec{F}(\bar{\rho} \doteq \bar{\sigma}), \vec{F}(G\bar{\rho}' \doteq \sigma)\},$$

for some set Δ of good constraints. We need to check that $\text{graph}(\Delta_1)$ is acyclic. But this is an immediate consequence of two facts: $\text{graph}(\Delta_1) = \text{graph}(\Delta_0)$ and, by hypothesis, $\text{graph}(\Delta_0)$ is acyclic. \square

Definition 4.12 (Solutions). Let $\mathbf{S} : \text{Var} \rightarrow (\mathbb{E} \cup \mathbb{T}^\rightarrow)$ be a substitution and let $\Delta = \{\tau_1 \doteq \tau'_1, \dots, \tau_n \doteq \tau'_n\}$ be a λ -compatible constraint set. We say \mathbf{S} is a *solution* for Δ iff $\mathbf{S}\tau_i = \mathbf{S}\tau'_i$ for every $i \in \{1, \dots, n\}$. \square

Definition 4.13 (Principal Solutions). Let $\mathbf{S} : \text{Var} \rightarrow (\mathbb{E} \cup \mathbb{T}^\rightarrow)$ be a substitution and let Δ be a λ -compatible constraint set. The substitution \mathbf{S} is a *principal solution* for Δ iff \mathbf{S} is a solution for Δ and for every solution \mathbf{S}' for Δ , there is a substitution \mathbf{S}'' such that $\mathbf{S}'\Delta = \mathbf{S}''(\mathbf{S}\Delta)$.⁴ The *principality property* is the existence of a principal solution for every constraint set that has a solution. \square

5 Algorithm for Lambda-Compatible Beta-Unification

We design a non-deterministic algorithm `Unify` which takes a λ -compatible constraint set Δ as input, such that if Δ has a solution then every evaluation of `Unify`(Δ) terminates returning a principal solution for Δ , and if Δ has no solution then every evaluation of `Unify`(Δ) diverges.

Metavariable conventions:

- $\bar{\rho} \in \mathbb{R}^\rightarrow, \rho \in \mathbb{R}, \bar{\sigma}, \bar{\sigma}_i \in \mathbb{S}^\rightarrow, \sigma \in \mathbb{S}, \tau, \tau' \in \mathbb{T}, e \in \mathbb{E}, \alpha \in \text{TVar}, F \in \text{EVar}.$

Mode of operation:

- Initial call: `Unify`(Δ) \Rightarrow `Unify`(`simplify`(Δ), $\{\}\text{, E-env}(\Delta)$).
- Final call: `Unify`($\emptyset, \mathbf{S}, \mathcal{E}$) $\Rightarrow \mathbf{S}$.
- `Unify`($\Delta_0, \mathbf{S}_0, \mathcal{E}$) \Rightarrow `Unify`($\Delta_1, \mathbf{S}_1, \mathcal{E}$), provided:
 - $\Delta_0 = \Delta \cup \vec{F}\{\rho \doteq \sigma\}$ and $\rho \doteq \sigma \Rightarrow \mathbf{S}$ is an instance of one of the rewrite rules.
 - $\Delta_1 = \text{simplify}(\mathbf{S}\Delta_0)$ and $\mathbf{S}_1 = \mathbf{S} \otimes_{\mathcal{E}} \mathbf{S}_0$.

Rewrite rules:

$$\alpha \doteq \bar{\sigma} \quad \Rightarrow \quad \{\{\alpha := \bar{\sigma}\}\} \quad (\text{rule 1})$$

$$\bar{\rho} \doteq \alpha \quad \Rightarrow \quad \{\{\alpha := \bar{\rho}\}\} \quad (\text{rule 2})$$

$$F\bar{\rho} \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n] \quad \Rightarrow \quad \{\{F := e\}\} \quad (\text{rule 3})$$

Applying substitutions to constraint sets:

- $\mathbf{S}\emptyset = \emptyset.$
- $\mathbf{S}(\{\tau \doteq \tau'\} \cup \Delta) = \{\mathbf{S}\tau \doteq \mathbf{S}\tau'\} \cup \mathbf{S}\Delta.$

Figure 5: Algorithm `Unify` (the function `simplify`() is defined in figure 4).

Definition 5.1 (Unification Algorithm). The operation of `Unify` is based on the rewrite rules shown in figure 5. The presentation of `Unify` in figure 5 is self-contained — except for two parts in the “mode of operation”, namely, the definition of `E-env`(Δ) and the evaluation of $\mathbf{S}_1 = \mathbf{S} \otimes_{\mathcal{E}} \mathbf{S}_0$. If Δ is the well-named

⁴We purposely write $\mathbf{S}'\Delta = \mathbf{S}''(\mathbf{S}\Delta)$ instead of $\mathbf{S}' = \mathbf{S}'' \circ \mathbf{S}$ in order to avoid pitfalls associated with the composition of substitutions in β -unification. These are carefully examined in section 3.

constraint set $\{\tau_1 \doteq \tau'_1, \dots, \tau_n \doteq \tau'_n\}$, then the *E-path environment* of Δ is the E-path environment of the type $\tau_1 \wedge \tau'_1 \wedge \dots \wedge \tau_n \wedge \tau'_n$ (see definition 3.11), i.e.,

$$\mathbf{E}\text{-env}(\Delta) = \mathbf{E}\text{-env}(\tau_1 \wedge \tau'_1 \wedge \dots \wedge \tau_n \wedge \tau'_n) .$$

If \mathcal{E} is an E-path environment, the evaluation of $\mathbf{S}_1 = \mathbf{S} \otimes_{\mathcal{E}} \mathbf{S}_0$ is given in definition 3.13. \square

REMARK 5.2. A rewrite step induced by **rule 1** or **rule 2** in figure 5 produces, from a given λ -compatible Δ_0 , another λ -compatible constraint set $\text{simplify}(\mathbf{S}\Delta_0)$. In fact, ignoring trivial constraints of the form $\tau \doteq \tau$ (same type τ on both sides) and before applying $\text{simplify}(\)$ to it, $\mathbf{S}\Delta_0$ is already λ -compatible. These assertions follow from lemmas 4.11 and 5.7 and their proofs.

By contrast, a rewrite step induced by **rule 3** in figure 5 produces a constraint set $\text{simplify}(\mathbf{S}\Delta_0)$ which is λ -compatible provided Δ_0 is, according to lemma 5.7, whereas $\mathbf{S}\Delta_0$ is not in general. That $\mathbf{S}\Delta_0$ is not necessarily λ -compatible in this case results from the (non-interesting) fact that a constraint of the form $\{e[\bar{\rho}_1, \dots, \bar{\rho}_n] \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]\}$ where $n \geq 2$, which is good of form (c) in definition 4.3, cannot be λ -compatible yet; it must be first broken up into

$$\{\vec{F}_1(\bar{\rho}_1 \doteq \bar{\sigma}_1), \dots, \vec{F}_n(\bar{\rho}_n \doteq \bar{\sigma}_n)\}$$

using $\text{simplify}(\)$, where $\vec{F}_i = \mathbf{E}\text{-path}(\square^{(i)}, e)$ for $1 \leq i \leq n$. \square

We next prove several lemmas, culminating in the main result of this section (theorem 5.16). On the way, there are 4 key intermediary results: **Unify** preserves λ -compatibility (lemma 5.7); **Unify** preserves solvability (lemma 5.11); if there is a solution, **Unify** constructs a principal one (lemma 5.12); and if there is a solution, **Unify** terminates (lemma 5.15). The remaining lemmas and definitions provide the necessary supporting material.

The next two lemmas, 5.3 and 5.5, are used in the proof of the first key result in lemma 5.7: λ -compatibility is an invariant of algorithm **Unify**.

Lemma 5.3 (A Property of Positive and Negative Types). *Let $R = \{\rho_1, \dots, \rho_m\} \subset \mathbb{R} - \mathbb{R}^{\rightarrow}$ and let $S = \{\bar{\sigma}_1, \dots, \bar{\sigma}_n\} \subset \mathbb{S}^{\rightarrow}$, with $m \geq 1$ and $n \geq 1$. Define the type:*

$$\sigma = (\rho_1 \rightarrow \dots \rightarrow \rho_m \rightarrow \beta) \wedge \bar{\sigma}_1 \wedge \dots \wedge \bar{\sigma}_n ,$$

where β is a fresh T -variable.⁵ Suppose σ satisfies:

1. σ is well-named.
2. Every $F \in \mathbf{EVar}(\sigma)$ has at most one positive occurrence in σ .
3. Every $\alpha \in \mathbf{TVar}(\sigma)$ has at most one negative occurrence in σ .

Then, for every $\tau, \tau' \in R \cup S$ such that $\tau \neq \tau'$, it holds that $\mathbf{Var}(\tau) \cap \mathbf{Var}(\tau') = \emptyset$. \square

Observe the two first conditions above correspond to (A) and (B) in definition 4.8, while the third condition is a weaker version of (C) in definition 4.8.

Proof. This is a straightforward proof. The details can be found in the technical report [KW02]. \square

EXAMPLE 5.4 (LINKS WITH POLARITIES). Consider the constraint set Δ in example 4.7, now with polarities inserted:

$$\begin{aligned} \pm \Delta = \{ & +\alpha_1 \doteq +F_1(\bar{\rho}_1)^+ \longrightarrow -\beta_1, -F_1(\sigma_1)^- \longrightarrow +\beta_1 \doteq -\alpha_2, \\ & +\alpha_2 \doteq +F_2(\bar{\rho}_2)^+ \longrightarrow -\beta_2, -F_2(\sigma_2)^- \longrightarrow +\beta_2 \doteq -\alpha_3 \}. \end{aligned}$$

With polarities inserted, the 8 links of $\text{graph}(\pm\Delta)$ are:

$$\begin{array}{cccc} +\alpha_1 \curvearrowright +F_1 & +\alpha_1 \curvearrowright -\beta_1 & -F_1 \curvearrowright -\alpha_2 & +\beta_1 \curvearrowright -\alpha_2 \\ +\alpha_2 \curvearrowright +F_2 & +\alpha_2 \curvearrowright -\beta_2 & -F_2 \curvearrowright -\alpha_3 & +\beta_2 \curvearrowright -\alpha_3 \end{array}$$

⁵By definition 4.1, the type σ thus defined is indeed a negative type, allowing us to use the letter “ σ ” to denote it.

There is a pattern here, which is proved in general in the next lemma: To the right of “ \curvearrowright ” it is always either a positive occurrence of an E-variable or a negative occurrence of a T-variable.

It is possible to specify rules to connect together links with polarities, in order to produce \curvearrowright -chains with polarities. But this is a little awkward and there is no real need for it. \square

Lemma 5.5 (A Property of Links). *Let Δ be a λ -compatible constraint set, and $\pm\Delta$ the corresponding set with polarities inserted. Then:*

1. *Every link in $\text{graph}(\pm\Delta)$ is in one of 2 possible forms:*

- *either “ $p v \curvearrowright +G$ ” where $v \in \text{Var}(\Delta)$ with $p \in \{+, -\}$ and $G \in \text{EVar}(\Delta)$,*
- *or “ $p v \curvearrowright -\beta$ ” where $v \in \text{Var}(\Delta)$ with $p \in \{+, -\}$ and $\beta \in \text{TVar}(\Delta)$.*

In words, the right end-point of a link with polarities is either a positive occurrence of an E-variable or a negative occurrence of a T-variable.

2. *For every $w \in \text{Var}(\Delta)$ there is at most one good constraint $\vec{F}(\bar{\rho} \doteq \bar{\sigma})$ of form (a) in Δ such that*

$$\{v \curvearrowright w \mid v \curvearrowright_{\Delta} w\} = \{v \curvearrowright w \mid v \curvearrowright_{\vec{F}\{\bar{\rho} \doteq \bar{\sigma}\}} w\},$$

i.e., all the links into w are contributed by at most one good constraint of form (a) in Δ . \square

Proof. Links are induced by good constraints of form (a) as specified in definition 4.3. Consider such a constraint $\vec{F}(\bar{\rho} \doteq \bar{\sigma})$. In general, $\bar{\rho}$ and $\bar{\sigma}$ are of the following forms:

$$\bar{\rho} = \sigma_1 \rightarrow \cdots \rightarrow \sigma_m \rightarrow \alpha \quad \text{and} \quad \bar{\sigma} = G_1 \bar{\rho}_1 \rightarrow \cdots \rightarrow G_n \bar{\rho}_n \rightarrow \beta$$

where $\sigma_1, \dots, \sigma_m \in \mathbb{S}$ and $\bar{\rho}_1, \dots, \bar{\rho}_n \in \mathbb{R}^{\rightarrow}$, with $m, n \geq 0$, and $\alpha, \beta \in \text{TVar}$ and $G_1, \dots, G_n \in \text{EVar}$. Inserting polarities, we obtain:

$$\begin{aligned} (\bar{\rho})^+ &= (\sigma_1)^- \rightarrow \cdots \rightarrow (\sigma_m)^- \rightarrow +\alpha \quad \text{and} \\ (\bar{\sigma})^- &= +G_1(\bar{\rho}_1)^+ \rightarrow \cdots \rightarrow +G_n(\bar{\rho}_n)^+ \rightarrow -\beta. \end{aligned}$$

Following definition 4.6, the conclusion of part 1 of the lemma is now immediate.

Part 2 follows from the fact that, in a λ -compatible constraint set Δ , every $G \in \text{EVar}(\Delta)$ has at most one inner positive occurrence, and every $\beta \in \text{TVar}(\Delta)$ has at most one inner negative occurrence. \square

REMARK 5.6. The dual statement of part 2 in lemma 5.5, namely, for every $v \in \text{Var}(\Delta)$ all the links *out from* v are contributed by at most one good constraint of form (a) in Δ , is generally false. For a counter-example, consider the constraint set:

$$\Delta = \{F\sigma \rightarrow \alpha \doteq \alpha', F\sigma' \rightarrow \alpha' \doteq \alpha''\}$$

for some appropriate $\sigma, \sigma' \in \mathbb{S}$ that make Δ λ -compatible. The links $F \curvearrowright_{\Delta} \alpha'$ and $F \curvearrowright_{\Delta} \alpha''$, both out from F , are contributed by two different constraints in Δ . \square

Lemma 5.7 (‘Unify’ Preserves λ -Compatibility). *Let Δ_0 and Δ_1 be constraint sets such that*

$$\text{Unify}(\Delta_0, \mathbf{S}_0, \mathcal{E}) \Rightarrow \text{Unify}(\Delta_1, \mathbf{S}_1, \mathcal{E})$$

for some $\mathbf{S}_0, \mathbf{S}_1$ and \mathcal{E} (which do not matter here). If Δ_0 is λ -compatible such that $\text{simplify}(\Delta_0) = \Delta_0$, then Δ_1 is λ -compatible. In words, the properties listed in definition 4.8 are invariant relative to the rewrite rules of Unify. \square

Proof. Δ_1 is obtained from Δ_0 according to one of the 3 rules in figure 5. We consider each of the 3 rules separately. In each of the 3 cases, a formal proof requires an induction on types, which we avoid because it is a routine uninteresting induction that obscures the underlying argument.

rule 1: The constraint under consideration is $\vec{F}(\alpha \doteq \bar{\sigma})$. There are two subcases, depending on whether α occurs exactly twice or exactly once in Δ_0 . We omit the latter subcase, which is trivial, and assume that

α occurs exactly twice. Thus, given that α has a single positive occurrence in $\vec{F}(\alpha \doteq \bar{\sigma})$, we assume that α has a single negative occurrence in Δ_0 . Let $\vec{G}(\rho \doteq \sigma)$ be the only constraint other than $\vec{F}(\alpha \doteq \bar{\sigma})$ in Δ_0 that mentions α . We thus have:

$$\begin{aligned}\Delta_0 &= \Delta \cup \vec{F}\{\alpha \doteq \bar{\sigma}\} \cup \vec{G}\{\rho \doteq \sigma\}, \\ \Delta_1 &= \text{simplify}(\Delta \cup \{\{\alpha := \bar{\sigma}\} \{\vec{G}(\rho \doteq \sigma)\}\}) \\ &= \text{simplify}(\Delta) \cup \text{simplify}(\{\{\alpha := \bar{\sigma}\} \{\vec{G}(\rho \doteq \sigma)\}\}) \\ &= \Delta \cup \text{simplify}(\{\{\alpha := \bar{\sigma}\} \{\vec{G}(\rho \doteq \sigma)\}\}),\end{aligned}$$

where Δ is the subset of all constraints of Δ_0 that do not mention α . Let $\hat{\Delta}_1 = \Delta \cup \{\{\alpha := \bar{\sigma}\} \{\vec{G}(\rho \doteq \sigma)\}\}$, so that $\Delta_1 = \text{simplify}(\hat{\Delta}_1)$. By lemma 4.11, it suffices to show that $\hat{\Delta}_1$ is λ -compatible. This means we have to show that all the constraints in $\hat{\Delta}_1$ are good of form (a) or (b) and that $\hat{\Delta}_1$ satisfies conditions (A), (B), (C) and (D) in definition 4.8. It is convenient to organize the proof by first showing (1) and (2) below, simultaneously with the fact that $\hat{\Delta}_1$ satisfies (D):

- (1) If $\alpha \in \text{TVar}(\rho)$, then $\text{Var}(\bar{\sigma}) \cap \text{Var}(\sigma) = \emptyset$.
- (2) If $\alpha \in \text{TVar}(\sigma)$, then $\text{Var}(\bar{\sigma}) \cap \text{Var}(\rho) = \emptyset$.

Because Δ_0 satisfies condition (A) in definition 4.8, it must be that \vec{G} is a prefix of \vec{F} . There are two cases, depending on whether \vec{G} is a proper prefix of \vec{F} or not.

Case 1: \vec{G} is a proper prefix of \vec{F} , i.e., $\vec{F} = \vec{G}\vec{H}$ for some $\vec{H} \neq \varepsilon$. If $\alpha \in \text{TVar}(\rho)$, then every E-variable in \vec{H} has an inner occurrence in ρ (because Δ_0 is well-named), which implies every E-variable in \vec{H} has no inner occurrence in σ (because $\text{Var}(\rho) \cap \text{Var}(\sigma) = \emptyset$), which implies $\text{Var}(\bar{\sigma}) \cap \text{Var}(\sigma) = \emptyset$, thus proving (1) above. By a totally similar argument, if $\alpha \in \text{TVar}(\sigma)$ then $\text{Var}(\bar{\sigma}) \cap \text{Var}(\rho) = \emptyset$, thus proving (2).

We next relate $\text{graph}(\hat{\Delta}_1)$ and $\text{graph}(\Delta_0)$. The set of nodes of $\text{graph}(\hat{\Delta}_1)$ is $\text{Var}(\Delta_0) - \{\alpha\}$, and the set of its links is:

$$\text{graph}(\hat{\Delta}_1) = \{v \rightsquigarrow w \mid v \rightsquigarrow_{\Delta_0} w \text{ and } v \neq \alpha\},$$

i.e., $\text{graph}(\hat{\Delta}_1)$ is a proper subset of $\text{graph}(\Delta_0)$. Because $\text{graph}(\Delta_0)$ is acyclic, so is $\text{graph}(\hat{\Delta}_1)$, thus proving that $\hat{\Delta}_1$ satisfies (D).

Case 2: $\vec{F} = \vec{G}$. Then $\vec{F}\{\alpha \doteq \bar{\sigma}, \rho \doteq \sigma\} \subseteq \Delta_0$. In general, $\bar{\sigma}$ is of the form:

$$\bar{\sigma} = \rho_1 \rightarrow \dots \rightarrow \rho_n \rightarrow \beta$$

for some $\rho_1, \dots, \rho_n \in \mathbb{R} - \mathbb{R}^-$ and $\beta \in \text{TVar}$ with $n \geq 0$. We have $\alpha \rightsquigarrow v$ for every $v \in \text{Var}(\bar{\sigma})$ such that $\text{E-path}(v, \Delta_0) = \vec{F}$, according to definition 4.6. Because $\text{simplify}(\Delta_0) = \Delta_0$, every constraint in Δ_0 is a good constraint of form (a.1) or (a.2) or (b), by lemma 4.5. There are three subcases, depending on whether $\vec{F}(\rho \doteq \sigma)$ is of one of these 3 forms.

Subcase 2.1: $\vec{F}(\rho \doteq \sigma)$ is of form (a.1), i.e., $\rho = \alpha' \in \text{TVar}$ and $\sigma \in \mathbb{S}^-$. Because Δ_0 satisfies (C), it must be that $\alpha \neq \alpha'$, thus proving (1) vacuously, i.e., it cannot be that $\alpha \in \text{TVar}(\rho)$. By definition 4.6, we have $\alpha' \rightsquigarrow v'$ for every $v' \in \text{Var}(\sigma)$ such that $\text{E-path}(v', \Delta_0) = \vec{F}$. If $\alpha \in \text{TVar}(\sigma)$, we have $\alpha' \rightsquigarrow \alpha$ and, as noted in the preceding paragraph, $\alpha \rightsquigarrow v$ for every $v \in \text{Var}(\bar{\sigma})$ such that $\text{E-path}(v, \Delta_0) = \vec{F}$. Because Δ_0 satisfies (D), it must be that $\alpha' \neq v$ for every $v \in \text{TVar}(\bar{\sigma})$, thus proving (2).

We relate $\text{graph}(\hat{\Delta}_1)$ and $\text{graph}(\Delta_0)$ in this subcase as follows. The set of nodes of $\text{graph}(\hat{\Delta}_1)$ are all the variables in $\text{Var}(\Delta_0) - \{\alpha\}$, and the set of its links is:

$$\text{graph}(\hat{\Delta}_1) = \{v \rightsquigarrow w \mid v \rightsquigarrow_{\Delta_0} w \text{ and } v \neq \alpha\} \cup \{\alpha' \rightsquigarrow w \mid \alpha \rightsquigarrow_{\Delta_0} w\}.$$

In the second part of this union, we replace a link $\alpha \rightsquigarrow w$ in $\text{graph}(\Delta_0)$ by the link $\alpha' \rightsquigarrow w$, for every $w \in \text{Var}(\bar{\sigma})$ with $\text{E-path}(w, \Delta_0) = \vec{F}$. Because $\text{graph}(\Delta_0)$ is acyclic and $\alpha' \rightsquigarrow_{\Delta_0} \alpha$, it now follows that $\text{graph}(\hat{\Delta}_1)$ is acyclic, thus proving that $\hat{\Delta}_1$ satisfies (D).

Subcase 2.2: $\vec{F}(\rho \doteq \sigma)$ is of form (a.2), i.e., $\sigma = \alpha' \in \text{TVar}$ and $\rho \in \mathbb{R}^-$. Because Δ_0 satisfies (C), it must be that $\alpha' \neq \beta$ (no two negative occurrences of the same T-variable). Because Δ_0 satisfies (A) and is

therefore well-named, it must be that $\alpha' \notin \text{TVar}(\rho_i)$ for every $1 \leq i \leq n$. Hence, $\alpha' \notin \text{TVar}(\bar{\sigma})$ which is the same as $\text{Var}(\sigma) \cap \text{Var}(\bar{\sigma}) = \emptyset$, thus proving (1). If $\alpha \in \text{TVar}(\sigma)$, then necessarily $\alpha = \alpha'$. Hence $v' \curvearrowright \alpha' = \alpha$ for every $v' \in \text{Var}(\rho)$ such that $\text{E-path}(v', \Delta_0) = \vec{F}$, by definition 4.6. Because Δ_0 satisfies (D), together with the fact that $\alpha \curvearrowright v$ for every $v \in \text{Var}(\bar{\sigma})$ such that $\text{E-path}(v, \Delta_0) = \vec{F}$, we have $\text{Var}(\rho) \cap \text{Var}(\bar{\sigma}) = \emptyset$, thus proving (2).

We relate $\text{graph}(\hat{\Delta}_1)$ and $\text{graph}(\Delta_0)$ in this subcase as follows. The set of nodes of $\text{graph}(\hat{\Delta}_1)$ are all the variables in $\text{Var}(\Delta_0) - \{\alpha\}$. If $\alpha \in \text{TVar}(\rho)$ and $\alpha' \neq \alpha$, then

$$\text{graph}(\hat{\Delta}_1) = \{v \curvearrowright w \mid v \curvearrowright_{\Delta_0} w \text{ and } v \neq \alpha\} \cup \{w \curvearrowright \alpha' \mid \alpha \curvearrowright_{\Delta_0} w\}.$$

In the second part of this union, we replace a link $\alpha \curvearrowright w$ in $\text{graph}(\Delta_0)$ by the link $w \curvearrowright \alpha'$, for every $w \in \text{Var}(\bar{\sigma})$ with $\text{E-path}(w, \Delta_0) = \vec{F}$. By part 2 of lemma 5.5, there are no links of the form $v \curvearrowright w$ where $v \neq \alpha$ and $w \in \text{Var}(\bar{\sigma})$, i.e., deleting the node α in $\text{graph}(\Delta_0)$ turns every such w into an isolated node or a ‘‘source’’ node in this case. Hence, although the direction of the link $\alpha \curvearrowright w$ is reversed to $w \curvearrowright \alpha'$, no cycle is introduced as a result. Together with the acyclicity of $\text{graph}(\Delta_0)$, this implies that $\text{graph}(\hat{\Delta}_1)$ is acyclic, thus proving that $\hat{\Delta}_1$ satisfies (D). If $\alpha \notin \text{TVar}(\rho)$ and $\alpha' = \alpha$, then

$$\text{graph}(\hat{\Delta}_1) = \{v \curvearrowright w \mid v \curvearrowright_{\Delta_0} w \text{ with } v \neq \alpha \text{ and } w \neq \alpha\} \cup \{v \curvearrowright w \mid v \curvearrowright_{\Delta_0} \alpha \text{ and } \alpha \curvearrowright_{\Delta_0} w\}.$$

This immediately implies that, if $\text{graph}(\Delta_0)$ is acyclic, then so is $\text{graph}(\hat{\Delta}_1)$, thus proving that $\hat{\Delta}_1$ satisfies (D) again.

Subcase 2.3: $\vec{F}(\rho \doteq \sigma)$ is of form (b), i.e., $\rho = H\bar{\rho}'$ for some $H \in \text{EVar}$ and $\bar{\rho}' \in \mathbb{R}^-$, and $\sigma = e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]$ for some $e \in \mathbb{E}$ and $\bar{\sigma}_1, \dots, \bar{\sigma}_n \in \mathbb{S}^-$ with $n = \#\square(e)$. Because Δ_0 is well-named, it must be $\alpha \notin \text{TVar}(\rho)$ and, therefore, $\alpha \in \text{TVar}(\sigma)$. By lemma 5.3, it must be $\text{Var}(\rho) \cap \text{Var}(\bar{\sigma}) = \emptyset$, thus proving both (1) and (2).

We relate $\text{graph}(\hat{\Delta}_1)$ and $\text{graph}(\Delta_0)$ as follows. The set of nodes of $\text{graph}(\hat{\Delta}_1)$ is $\text{Var}(\Delta_0) - \{\alpha\}$, and the set of its links is:

$$\text{graph}(\hat{\Delta}_1) = \{v \curvearrowright w \mid v \curvearrowright_{\Delta_0} w \text{ and } v \neq \alpha\}.$$

In words, to obtain $\text{graph}(\hat{\Delta}_1)$, we eliminate all links in $\text{graph}(\Delta_0)$ of the form $\alpha \curvearrowright w$. Because $\text{graph}(\Delta_0)$ is acyclic, so is $\text{graph}(\hat{\Delta}_1)$, thus proving that $\hat{\Delta}_1$ satisfies (D).

We have thus completed the proof that (1) and (2) are true, and that $\hat{\Delta}_1$ satisfies condition (D), in all cases and subcases.⁶

Using the fact that α occurs either in ρ or in σ , we conclude that $\text{Var}(\{\alpha := \bar{\sigma}\}\rho) \cap \text{Var}(\{\alpha := \bar{\sigma}\}\sigma) = \emptyset$, from (1.1) and (1.2). This in turn implies $\{\alpha := \bar{\sigma}\}\{\vec{G}(\rho \doteq \sigma)\}$ is a good constraint of form (a) or (b), depending on whether $\vec{G}(\rho \doteq \sigma)$ is good of form (a) or (b), respectively. It follows that every constraint in $\hat{\Delta}_1$ is a good constraint of form (a) or (b).

We next proceed to show that $\hat{\Delta}_1$ satisfies conditions (A), (B), and (C) in definition 4.8. Because $\text{E-path}(\alpha, \Delta_0) = \text{E-path}(\alpha, \hat{\Delta}_1)$, it is readily checked that if Δ_0 satisfies conditions (A) and (C), then so does $\hat{\Delta}_1$. What makes things work as expected is that we substitute a negative type $\bar{\sigma}$ for a negative occurrence of α . If Δ_0 satisfies condition (B), then so does $\hat{\Delta}_1$ trivially.

rule 2: This is symmetric to **rule 1**, i.e., the difference between **rule 1** and **rule 2** are the reversed polarities, and is therefore omitted. (In fact, the proof for **rule 2** is somewhat easier, because we can here merge the counterparts of subcase 2.1 and subcase 2.2 into a single subcase, which occurs when $\vec{F}\{\bar{\rho} \doteq \alpha, \rho \doteq \sigma\} \subseteq \Delta_0$ and $(\rho \doteq \sigma)$ is of form (a), with no need to deal with forms (a.1) and (a.2) separately. This is because the positive occurrence of α here always occurs in ρ , never in σ ; by contrast, for **rule 1**, when $\vec{F}\{\alpha \doteq \bar{\sigma}, \rho \doteq \sigma\} \subseteq \Delta_0$, the negative occurrence of α may occur in ρ just as in σ . As a result also, in the proof for **rule 2**, we do not need to invoke lemma 5.5.)

⁶In only two places so far, do we need to invoke condition (D), i.e., that $\text{graph}(\Delta_0)$ is acyclic, in order to prove (1) and (2), which are next used to show good constraints are preserved by the rewrite rules; these two places are subcase 2.1 and subcase 2.2. There is one more place, in the part of the proof for **rule 2** below corresponding to subcase 2.1 and subcase 2.2, where we need to invoke (D) again. Nowhere else, in the present proof or elsewhere in this paper, do we need to invoke condition (D) in an induction to prove a property other than itself. But, of course, we also have to show that (D) itself is initially satisfied, which is shown in the proof of lemma 6.2, and preserved by all the rewrite rules, which is shown in the present proof.

rule 3: Let $\Delta_0 = \Delta \cup \vec{G}\{F\bar{\rho} \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]\}$, where the constraint $F\bar{\rho} \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]$ induces a rewrite step according to **rule 3**. Δ_0 is transformed to $\Delta_1 = \text{simplify}(\hat{\Delta}_1)$ where

$$\hat{\Delta}_1 = \{[F := e]\Delta \cup \vec{G}\{e[\langle \bar{\rho} \rangle^{s_1}, \dots, \langle \bar{\rho} \rangle^{s_n}] \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]\}$$

where $s_i = \text{path}(\square^{(i)}, e)$ for every $1 \leq i \leq n$. Note that $\hat{\Delta}_1$ is not necessarily in the form required by definition 4.8, because $\hat{\Delta}_1$ may contain good constraints of form (c) as specified in definition 4.3. However, by lemma 4.5, $\Delta_1 = \text{simplify}(\hat{\Delta}_1)$ is in the required form for λ -compatibility, i.e., it consists of good constraints all of form (a) or form (b).

Consider a T-variable or E-variable $v \in \text{Var}(\Delta_0)$ such that F occurs in $\text{E-path}(v, \Delta_0)$. Because Δ_0 is well-named, it must be that $\text{E-path}(v, \Delta_0) = \vec{G}F\vec{H}$ for some $\vec{H} \in \text{EVar}^*$. Because every constraint in Δ_0 is good, it must be that $F \notin \text{EVar}(e)$. It follows that $\text{EVar}(\vec{H}) \cap \text{EVar}(e) = \emptyset$. Hence, substituting e for F produces a well-named $\hat{\Delta}_1$. Hence, Δ_1 is also well-named and, thus, satisfies condition (A) in definition 4.8.

Because Δ_0 satisfies condition (B) and (C) in definition 4.8, then so does $\hat{\Delta}_1$ trivially, again using the fact that Δ_0 is well-named (use, in particular, condition 2 in definition 3.9). Thus, Δ_1 satisfies conditions (B) and (C) in definition 4.8.

It remains to show that Δ_1 satisfies condition (D). For every $1 \leq i \leq n$, let $\text{E-path}(\square^{(i)}, e) = \vec{H}_i$ for some $\vec{H}_i \in \text{EVar}^*$. The following equality follows from the pertinent definitions:

$$\begin{aligned} \text{graph}(\Delta_1) &= \{v \curvearrowright w \mid v \curvearrowright_{\Delta_0} w \text{ with } F \notin \text{E-path}(v, \Delta_0) = \text{E-path}(w, \Delta_0)\} \cup \\ &\quad \{v^s \curvearrowright w^s \mid v \curvearrowright_{\Delta_0} w \text{ with } F \in \text{E-path}(v, \Delta_0) = \text{E-path}(w, \Delta_0) \text{ and } s \in \{s_1, \dots, s_n\}\} \cup \\ &\quad \left(\bigcup \{ \text{graph}(\vec{G}\vec{H}_i\{\langle \bar{\rho} \rangle^{s_i} \doteq \bar{\sigma}_i\}) \mid 1 \leq i \leq n \} \right). \end{aligned}$$

We have to show that $\text{graph}(\Delta_1)$ is acyclic. Given a \curvearrowright -chain $C = v_1 \curvearrowright v_2 \curvearrowright \dots \curvearrowright v_m$ in $\text{graph}(\Delta)$ for some λ -compatible constraint set Δ , it is meaningful to write $\text{E-path}(C, \Delta)$ because all the entries in C have the same E-path in Δ . Given an offset $s \in \{0, 1\}^*$, we write C^s to denote $v_1^s \curvearrowright v_2^s \curvearrowright \dots \curvearrowright v_m^s$, which may or may not be a valid \curvearrowright -chain in $\text{graph}(\Delta)$.

Consider an arbitrary \curvearrowright -chain C in $\text{graph}(\Delta_1)$. If $\text{E-path}(C, \Delta_1) = \vec{L} \notin \{\vec{G}\vec{H}_1, \dots, \vec{G}\vec{H}_n\}$, then C is acyclic, because *either* C is already a \curvearrowright -chain in $\text{graph}(\Delta_0)$ with $\text{E-path}(C, \Delta_0) = \vec{L}$, *or* $C = C_0^s$ for some \curvearrowright -chain C_0 in $\text{graph}(\Delta_0)$ and $s \in \{0, 1\}^*$. In the latter situation, we have $\vec{L} = \vec{G}\vec{H}_i\langle \vec{K} \rangle^s$ with $\vec{K} \neq \varepsilon$ and $s = s_i$ for some $i \in \{1, \dots, n\}$, and $\text{E-path}(C_0, \Delta_0) = \vec{G}F\vec{K}$ which is mapped to $\text{E-path}(C, \Delta_1) = \vec{G}\vec{H}_i\langle \vec{K} \rangle^s$ after applying the substitution $\{[F := e]\}$.

The non-trivial case occurs when $C = w_1 \curvearrowright_{\Delta_1} w_2 \curvearrowright_{\Delta_1} \dots \curvearrowright_{\Delta_1} w_m$ such that $\text{E-path}(C, \Delta_1) = \vec{G}\vec{H}_i$ for some $1 \leq i \leq n$. There are two cases, $n = 1$ and $n \geq 2$. Consider the case $n = 1$ first. In this case, $e = \vec{H}\square$ for some $\vec{H} \in \text{EVar}^*$. By definition 4.6, $X = \text{graph}(\Delta_0)/\vec{G}F$ and $Y = \text{graph}(\Delta_0)/\vec{G}\vec{H}$ are disconnected components of $\text{graph}(\Delta_0)$, i.e., for every $v, w \in \text{Var}(\Delta_0)$ such that $\text{E-path}(v, \Delta_0) = \vec{G}F$ and $\text{E-path}(w, \Delta_0) = \vec{G}\vec{H}$, we have $v \not\curvearrowright w$ and $w \not\curvearrowright v$. In the case $n = 1$, after substituting \vec{H} for F , the new constraint $\vec{G}\vec{H}(\bar{\rho} \doteq \bar{\sigma})$ introduces links of the form $v \curvearrowright w$, all directed from component X to Y . Hence, X and Y are now part of the same component in $\text{graph}(\Delta_1)$, but still acyclic; as all the other components of $\text{graph}(\Delta_1)$ are components of $\text{graph}(\Delta_0)$ and therefore acyclic, $\text{graph}(\Delta_1)$ is acyclic.

Consider the case $n \geq 2$ next. New links of the form $v^{s_i} \curvearrowright_{\Delta_1} w$ are introduced by every new constraint $\vec{G}\vec{H}_i(\langle \bar{\rho} \rangle^{s_i} \doteq \bar{\sigma}_i)$ where $s_i \neq \varepsilon$, and $v \in \text{Var}(\bar{\rho})$ and $w \in \text{Var}(\bar{\sigma})$. In general, there is a fixed $i \in \{1, \dots, n\}$ such that:

$$\begin{aligned} C &= w_1 \curvearrowright_{\Delta_1} \dots \curvearrowright_{\Delta_1} w_k \curvearrowright_{\Delta_1} w_{k+1} \curvearrowright_{\Delta_1} \dots \curvearrowright_{\Delta_1} w_{k+\ell} \quad \text{where} \\ &- 0 \leq k \leq m, \quad 0 \leq \ell \leq m \quad \text{and} \quad k + \ell = m, \\ &- w_1 = v_1^s, \dots, w_k = v_k^s \text{ for some } v_1, \dots, v_k \in \text{Var}(\Delta_0), \text{ with } s = s_i \neq \varepsilon, \\ &- w_{k+1}, \dots, w_{k+\ell} \in \text{Var}(\Delta_0) \cap \text{Var}(\Delta_1). \end{aligned}$$

If $\ell = 0$, i.e., $C = v_1^s \curvearrowright_{\Delta_1} \dots \curvearrowright_{\Delta_1} v_k^s$, then by stripping the offset s we obtain a \curvearrowright -chain C_0 in $\text{graph}(\Delta_0)$, namely, $C_0 = v_1 \curvearrowright_{\Delta_0} \dots \curvearrowright_{\Delta_0} v_k$. Because C_0 is acyclic, so is C acyclic.

If $k = 0$, i.e., $C = w_{k+1} \curvearrowright_{\Delta_1} \dots \curvearrowright_{\Delta_1} w_{k+\ell}$, then C is already a \curvearrowright -chain in $\text{graph}(\Delta_0)$. Because $\text{graph}(\Delta_0)$ is acyclic, C is also acyclic.

Suppose now $k \neq 0 \neq \ell$. Neither of the two \curvearrowright -chains, $v_1^s \curvearrowright_{\Delta_1} \cdots \curvearrowright_{\Delta_1} v_k^s$ and $w_{k+1} \curvearrowright_{\Delta_1} \cdots \curvearrowright_{\Delta_1} w_{k+\ell}$, contains a cycle, for the same reasons given in the case $\ell = 0$ and $k = 0$, respectively. The two \curvearrowright -chains are connected by the link $v_k^s \curvearrowright_{\Delta_1} w_{k+1}$ introduced by the constraint $\tilde{G}\tilde{H}_i(\langle \bar{\rho} \rangle^s \doteq \bar{\sigma}_i)$. Because Δ_0 is well-named, condition 1 in definition 3.9 implies $\{v_1, \dots, v_k\} \cap \{w_1, \dots, w_{k+\ell}\} = \emptyset$, and condition 2 in definition 3.9 implies $\{v_1^s, \dots, v_k^s\} \cap \{w_1, \dots, w_{k+\ell}\} = \emptyset$. Hence, C is acyclic. \square

Lemma 5.8 (Progress). *Let Δ_0 be a λ -compatible constraint set such that $\text{simplify}(\Delta_0) = \Delta_0$. If Δ_0 is not empty, then for every \mathbf{S} and \mathcal{E}*

$$1. \text{Unify}(\Delta_0, \mathbf{S}, \mathcal{E}) \Rightarrow \text{Unify}(\Delta_1, \tilde{\mathbf{S}} \otimes_{\mathcal{E}} \mathbf{S}, \mathcal{E}) ,$$

for some Δ_1 and $\tilde{\mathbf{S}}$ (which do not matter here). Moreover, whenever part 1 holds for some Δ_1 and $\tilde{\mathbf{S}}$, it also holds that:

$$2. \text{If } \mathbf{S}\mathcal{E} \supseteq \text{E-env}(\Delta_0) \text{ then } (\tilde{\mathbf{S}} \otimes_{\mathcal{E}} \mathbf{S})\mathcal{E} \supseteq \text{E-env}(\Delta_1) .$$

In words, part 1 says that Δ_0 always contains a constraint that can be processed by one of the rewrite rules of **Unify**. Part 2 is another invariant property of **Unify**, which says that E-path environments are preserved in the process of rewriting constraint sets (how a substitution \mathbf{S} is applied to an E-path environment \mathcal{E} to obtain another E-path environment $\mathbf{S}\mathcal{E}$ is given in definition 3.11). \square

Proof. For part 1, we have in fact a stronger result: Every constraint in a λ -compatible constraint set Δ such that $\text{simplify}(\Delta) = \Delta$ can be processed by one of the rewrite rules. This is a consequence of lemma 4.5: A good constraint of form (a.1) is processed by **rule 1**, a good constraint of form (a.2) is processed by **rule 2**, and a good constraint of form (b) is processed by **rule 3**.

Part 2 follows from: the fact that $\Delta_1 = \text{simplify}(\tilde{\mathbf{S}}\Delta_0)$ which implies $\text{E-env}(\Delta_1) \subseteq \text{E-env}(\tilde{\mathbf{S}}\Delta_0)$, the fact that $(\tilde{\mathbf{S}} \otimes_{\mathcal{E}} \mathbf{S})\mathcal{E} = \tilde{\mathbf{S}}(\mathbf{S}\mathcal{E})$ by lemma 3.18, the hypothesis $\mathbf{S}\mathcal{E} \supseteq \text{E-env}(\Delta_0)$, and the definition of E-path environments (definition 3.11) – producing the following sequence of equalities and containments:

$$(\tilde{\mathbf{S}} \otimes_{\mathcal{E}} \mathbf{S})\mathcal{E} = \tilde{\mathbf{S}}(\mathbf{S}\mathcal{E}) \supseteq \tilde{\mathbf{S}}(\text{E-env}(\Delta_0)) = \text{E-env}(\tilde{\mathbf{S}}\Delta_0) \supseteq \text{E-env}(\Delta_1)$$

which is the desired conclusion. \square

Lemma 5.9 (‘Simplify’ Preserves Solvability). *Let Δ be a λ -compatible constraint set and let \mathbf{S} be a substitution. Then:*

$$1. \mathbf{S} \text{ is a solution for } \Delta \text{ iff } \mathbf{S} \text{ is a solution for } \text{simplify}(\Delta).$$

$$2. \mathbf{S} \text{ is a principal solution for } \Delta \text{ iff } \mathbf{S} \text{ is a principal solution for } \text{simplify}(\Delta). \quad \square$$

Proof. For part 1 of the lemma, it suffices to consider the case when Δ consists of a single constraint $\tau \doteq \tau'$. Moreover, it suffices to show that \mathbf{S} is a solution for Δ iff \mathbf{S} is a solution for $\text{simplify}_1(\Delta)$, where $\text{simplify}_1(\cdot)$ is defined in the proof of lemma 4.5.

There are 5 cases in the definition of $\text{simplify}_1(\cdot)$. In the 4th case (when $\tau = \tau'$) and 5th case (when $\text{simplify}_1(\Delta) = \Delta$), the desired conclusion is immediate.

In the 2nd case (when $\tau = \tau_1 \rightarrow \tau_2$ and $\tau' = \tau'_1 \rightarrow \tau'_2$) and 3rd case (when $\tau = \tau_1 \wedge \tau_2$ and $\tau' = \tau'_1 \wedge \tau'_2$), the desired conclusion follows from the way substitutions are lifted to types (definition 2.15).

For the 1st case, suppose $\tau = \vec{F}\rho$ and $\tau' = \vec{F}\sigma$ with $\vec{F} \neq \varepsilon$ and $\text{Var}(\rho) \cap \text{Var}(\sigma) = \emptyset$. A straightforward computation shows that:

$$\text{simplify}_1(\tau \doteq \tau') = \text{simplify}_1(\vec{F}(\rho \doteq \sigma)) = \vec{F} \text{simplify}_1(\rho \doteq \sigma) .$$

There are different subcases depending on the forms of ρ and σ . We consider only one of these subcases, namely, when $\rho = \sigma_1 \rightarrow \rho_1$ and $\sigma = \rho_2 \rightarrow \sigma_2$; all other subcases are treated similarly. For this subcase, the desired conclusion follows from the fact that \mathbf{S} is a solution for $\vec{F}(\rho \doteq \sigma)$ iff \mathbf{S} is a solution for $\{\vec{F}(\rho_1 \doteq \sigma_2), \vec{F}(\rho_2 \doteq \sigma_1)\}$. Remaining details omitted.

For part 2 of the lemma, let \mathbf{S} be a *principal* solution for Δ which, by part 1, is also a solution for $\text{simplify}(\Delta)$. It suffices to show that \mathbf{S} is principal for $\text{simplify}_1(\Delta)$, where $\text{simplify}_1(\cdot)$ is defined in the proof of lemma 4.5. This is a straightforward consequence of principality (definition 4.13) and the way substitutions are lifted to types (definition 2.15). This proves the left-to-right implication of part 2. The converse implication is readily proved in the same way. Remaining details omitted. \square

The next lemma is used in the proof of lemma 5.11.

Lemma 5.10 (A Property of Substitutions). *Let $\tau \in \mathbb{T}$ be well-named, $\alpha \in \text{TVar}(\tau)$ and $\bar{\tau} \in \mathbb{T}^\rightarrow$. If $\text{E-path}(\alpha, \tau) = \vec{F}$ and \mathbf{S} is a substitution such that $\mathbf{S}(\vec{F}\alpha) = \mathbf{S}(\vec{F}\bar{\tau})$, then $\mathbf{S}(\{\alpha := \bar{\tau}\} \tau) = \mathbf{S}\tau$. \square*

Proof. By induction on well-named $\tau \in \mathbb{T}$ we prove the stronger conclusion: For every offset $p \in \{0, 1\}^*$ we have $\mathbf{S}\langle\{\alpha := \bar{\tau}\} \tau\rangle^p = \mathbf{S}\langle\tau\rangle^p$. The base case of the induction is $\tau = \alpha$, in which case $\vec{F} = \varepsilon$. Because $\{\alpha := \bar{\tau}\} \tau = \bar{\tau}$ here, the desired conclusion follows.

Proceeding inductively, the two cases $\tau = \tau_1 \rightarrow \tau_2$ and $\tau = \tau_1 \wedge \tau_2$, are treated similarly. Consider the first only. Suppose $\alpha \in \text{TVar}(\tau)$ with $\text{E-path}(\alpha, \tau) = \vec{F}$. With no loss of generality, let $\alpha \in \text{TVar}(\tau_1) - \text{TVar}(\tau_2)$. (The two other subcases, $\alpha \in \text{TVar}(\tau_2) - \text{TVar}(\tau_1)$ and $\alpha \in \text{TVar}(\tau_1) \cap \text{TVar}(\tau_2)$, are similar and therefore omitted.) Then $\text{E-path}(\alpha, \tau_1) = \vec{F}$. By the induction hypothesis, $\mathbf{S}\langle\{\alpha := \bar{\tau}\} \tau_1\rangle^p = \mathbf{S}\langle\tau_1\rangle^p$ for every $p \in \{0, 1\}^*$. Moreover, $\alpha \notin \text{TVar}(\tau_2)$ implies $\{\alpha := \bar{\tau}\} \tau_2 = \tau_2$, which implies $\mathbf{S}\langle\{\alpha := \bar{\tau}\} \tau_2\rangle^p = \mathbf{S}\langle\tau_2\rangle^p$ for every $p \in \{0, 1\}^*$. Hence, $\mathbf{S}\langle\{\alpha := \bar{\tau}\} \tau\rangle^p = \mathbf{S}\langle\tau\rangle^p$ for every $p \in \{0, 1\}^*$, which is the desired conclusion.

The last case of the induction is when $\tau = G\tau'$ for some $G \in \text{EVar}$ and $\tau' \in \mathbb{T}$. If τ is well-named, so is τ' , and if $\alpha \in \text{TVar}(\tau)$ then $\alpha \in \text{TVar}(\tau')$. Let $\vec{F} = \text{E-path}(\alpha, \tau')$, so that $G\vec{F} = \text{E-path}(\alpha, \tau)$. Consider an arbitrary $p \in \{0, 1\}^*$ and let $\mathbf{S}G^p = e$ where $\#\square(e) = n$ and $\text{paths}(e) = (s_1, \dots, s_n)$. We then have the following sequence of equalities:

$$\begin{aligned} \mathbf{S}\langle\{\alpha := \bar{\tau}\} \tau\rangle^p &= \mathbf{S}\langle G(\{\alpha := \bar{\tau}\} \tau')\rangle^p && \text{because } \tau = G\tau', \\ &= e[\mathbf{S}\langle\{\alpha := \bar{\tau}\} \tau'\rangle^{p \cdot s_1}, \dots, \mathbf{S}\langle\{\alpha := \bar{\tau}\} \tau'\rangle^{p \cdot s_n}] && \text{because } \mathbf{S}G^p = e, \\ &= e[\mathbf{S}\langle\tau'\rangle^{p \cdot s_1}, \dots, \mathbf{S}\langle\tau'\rangle^{p \cdot s_n}] && \text{by the induction hypothesis} \\ &= \mathbf{S}\langle G\tau'\rangle^p && \text{because } \mathbf{S}G^p = e, \\ &= \mathbf{S}\langle\tau\rangle^p && \text{because } \tau = G\tau', \end{aligned}$$

which is the desired conclusion. This completes the induction and the proof. \square

Lemma 5.11 ('Unify' Preserves Solvability). *Let Δ_0 be a λ -compatible constraint set, let Δ_1 be a constraint set, and let:*

$$\text{Unify}(\Delta_0, \mathbf{S}, \mathcal{E}) \Rightarrow \text{Unify}(\Delta_1, \tilde{\mathbf{S}} \otimes_{\mathcal{E}} \mathbf{S}, \mathcal{E})$$

for some \mathbf{S} , $\tilde{\mathbf{S}}$ and \mathcal{E} (which do not matter here). Then, from a solution \mathbf{S}_0 for Δ_0 , we can construct a solution \mathbf{S}_1 for Δ_1 . \square

Proof. Δ_1 is obtained from Δ_0 according to one of the 3 rewrite rules in figure 5. In each of the 3 cases, $\Delta_1 = \text{simplify}(\Delta'_1)$ where $\Delta'_1 = \tilde{\mathbf{S}}\Delta_0$ for some appropriately defined substitution $\tilde{\mathbf{S}}$. We consider each case separately. In all 3 cases, from a solution \mathbf{S}_0 for Δ_0 , we show the existence of a solution \mathbf{S}_1 for Δ'_1 (as well as Δ_1).

rule 1: The given constraint set $\Delta_0 = \Delta \cup \vec{F}\{\alpha \doteq \bar{\sigma}\}$ is transformed to $\Delta'_1 = \{\alpha := \bar{\sigma}\} \Delta \cup \vec{F}\{\bar{\sigma} \doteq \bar{\sigma}\}$. There are two subcases, depending on whether α occurs exactly twice or exactly once in Δ_0 . We omit the latter subcase, which is trivial, and assume that α occurs exactly twice. If α occurs exactly twice in Δ_0 , there is exactly one constraint in Δ_0 , say $\vec{G}(\rho \doteq \sigma) \in \Delta$, that mentions α and it mentions α exactly once. With no loss of generality, assume $\alpha \in \text{TVar}(\rho)$ and $\alpha \notin \text{TVar}(\sigma)$; in particular, because Δ_0 is well-named, $\text{E-path}(\alpha, \vec{G}\rho) = \vec{F}$. Because \mathbf{S}_0 is a solution for Δ_0 , we have $\mathbf{S}_0(\vec{F}\alpha) = \mathbf{S}_0(\vec{F}\bar{\sigma})$. This implies, by lemma 5.10, that $\mathbf{S}_0(\{\alpha := \bar{\sigma}\} \rho) = \mathbf{S}_0 \rho$. Hence, the desired solution \mathbf{S}_1 for Δ'_1 is simply the given solution \mathbf{S}_0 for Δ_0 .

rule 2: Identical to **rule 1**, except for the reversed polarities.

rule 3: Let $\Delta_0 = \Delta \cup \vec{G}\{F\bar{\rho} \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]\}$, where the constraint $F\bar{\rho} \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]$ induces a rewrite step according to **rule 3**. Δ_0 is transformed to $\Delta_1 = \text{simplify}(\Delta'_1)$ where

$$\Delta'_1 = \{[F := e]\Delta \cup \vec{G}\{e[\langle\bar{\rho}\rangle^{s_1}, \dots, \langle\bar{\rho}\rangle^{s_n}] \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]\}$$

where $\text{paths}(e) = (s_1, \dots, s_n)$. Because \mathbf{S}_0 is a solution for Δ_0 , we have that $\mathbf{S}_0(\vec{G}F\bar{\rho}) = \mathbf{S}_0(\vec{G}e)$. It suffices to construct a substitution \mathbf{S}_1 such that $\mathbf{S}_0 = \mathbf{S}_1 \otimes_{\mathcal{E}'}$ where $\mathcal{E}' = \text{E-env}(\Delta_0)$. We keep the action of \mathbf{S}_1 on every variable v (and its offsprings) for which $F \notin \text{E-path}(v, \Delta_0)$ identical to the action of \mathbf{S}_0 ; in

particular, $\mathbf{S}_1(\vec{G}e) = \mathbf{S}_0(\vec{G}e)$. For every v such that $\mathbf{E}\text{-path}(v, \Delta_0) = \vec{G}F\vec{H}$ for some $\vec{H} \in \mathbf{EVar}^*$, it therefore suffices to construct \mathbf{S}_1 so that

$$\begin{aligned} \mathbf{S}_0(\vec{G}F\vec{H}v) &= (\mathbf{S}_1 \otimes_{\mathcal{E}} \{\{F := e\}\})(\vec{G}F\vec{H}v) \\ &= \mathbf{S}_1(\{\{F := e\}\})(\vec{G}F\vec{H}v) \\ &= \mathbf{S}_1(\vec{G}e[\langle \vec{H}v \rangle^{s_1}, \dots, \langle \vec{H}v \rangle^{s_n}]) \end{aligned}$$

Note that the action of \mathbf{S}_1 on $\vec{G}e$ is already determined, because $F \notin \mathbf{EVar}(\vec{G}e)$, but not on $\langle \vec{H}v \rangle^{s_i}$ and its offsprings. By part 1 of lemma 3.15, $\mathbf{S}_1(\vec{G}e)$ is an expansion, say e' . We then have

$$e' = \mathbf{S}_1(\vec{G}e) = \mathbf{S}_0(\vec{G}e) = \mathbf{S}_0(\vec{G}F\Box)$$

Let $\#\Box(e') = m \geq n$ and $\text{paths}(e') = (p_1, \dots, p_m)$. By lemma 3.16,

$$\mathbf{S}_0(\vec{G}F\vec{H}v) = e'[\mathbf{S}_0\langle \vec{H}v \rangle^{p_1}, \dots, \mathbf{S}_0\langle \vec{H}v \rangle^{p_m}]$$

By part 2 of lemma 3.15,

$$\mathbf{S}_1(\vec{G}e[\langle \vec{H}v \rangle^{s_1}, \dots, \langle \vec{H}v \rangle^{s_n}]) = e'[\mathbf{S}_1\langle \vec{H}v \rangle^{s_{a(1)}r(a(1), b(1))}, \dots, \mathbf{S}_1\langle \vec{H}v \rangle^{s_{a(m)}r(a(m), b(m))}]$$

for appropriately defined maps:

$$\begin{aligned} a &: \{1, \dots, m\} \rightarrow \{1, \dots, n\} \\ b &: \{1, \dots, m\} \rightarrow \{1, \dots, m\} \\ r &: \{1, \dots, n\} \times \{1, \dots, m\} \rightarrow \{0, 1\}^* \end{aligned}$$

such that

- $\{a(1), \dots, a(m)\} = \{1, \dots, n\}$,
- if $\mathbf{E}\text{-path}(\Box^{(i)}, \vec{G}e) = \vec{G}\vec{G}_i$ for $1 \leq i \leq n$,
then $\text{paths}(\mathbf{S}_1(\vec{G}\vec{G}_i\Box)) = \text{paths}(\mathbf{S}_0(\vec{G}\vec{G}_i\Box)) = (r(i, 1), \dots, r(i, m_i))$ for some $m_i \geq 1$,
- $m = m_1 + \dots + m_n$,
- $\{b(j) \mid a(j) = i \text{ and } 1 \leq j \leq m\} = \{1, \dots, m_i\}$ for every $1 \leq i \leq n$.

Hence, to complete the definition of \mathbf{S}_1 , we need to satisfy the equality $\mathbf{S}_1\langle \vec{H}v \rangle^{s_{a(i)}r(a(i), b(i))} = \mathbf{S}_0\langle \vec{H}v \rangle^{p_i}$ for every $1 \leq i \leq m$. If $\vec{H} = H_1H_2 \cdots H_\ell$ for some $\ell \geq 0$ and $s_{a(i)}r(a(i), b(i)) = q_i$, this equality is satisfied by setting \mathbf{S}_1 according to the following values:

$$\mathbf{S}_1H_j^{q_i t} = \mathbf{S}_0H_j^{p_i t} \quad \text{and} \quad \mathbf{S}_1v^{q_i t} = \mathbf{S}_0v^{p_i t}$$

for every $1 \leq j \leq \ell$ and every $t \in \{0, 1\}^*$. □

Lemma 5.12 (Principal Solution Constructed). ; Let Δ_0 and Δ_1 be λ -compatible constraint sets, let \mathbf{S} be a substitution and \mathcal{E} an E-path environment such that $\mathbf{S}\mathcal{E} \supseteq \mathbf{E}\text{-env}(\Delta_0)$, and let:

$$\text{Unify}(\Delta_0, \mathbf{S}, \mathcal{E}) \implies \text{Unify}(\Delta_1, \tilde{\mathbf{S}} \otimes_{\mathcal{E}} \mathbf{S}, \mathcal{E})$$

for some substitution $\tilde{\mathbf{S}}$. If \mathbf{S}_1 is a principal solution for Δ_1 , then so is $\mathbf{S}_1 \otimes_{\mathbf{S}\mathcal{E}} \tilde{\mathbf{S}}$ for Δ_0 . □

Proof. Δ_1 is obtained from Δ_0 according to one of the 3 rewrite rules in figure 5. For each of the 3 rules, we have $\Delta_0 = \Delta \cup \vec{F}\{\rho \doteq \sigma\}$ and $\Delta_1 = \text{simplify}(\Delta'_1)$, where $\Delta'_1 = \tilde{\mathbf{S}}\Delta_0$ and $\tilde{\mathbf{S}}$ depends on the form of $\rho \doteq \sigma$ (see figure 5). Suppose \mathbf{S}_1 is a principal solution for Δ_1 .

First, we check that $\mathbf{S}_0 = \mathbf{S}_1 \otimes_{\mathbf{S}\mathcal{E}} \tilde{\mathbf{S}}$ is a solution for Δ_0 . Consider a constraint $\vec{G}\rho' \doteq \vec{G}\sigma'$ in Δ_0 , which implies that $\tilde{\mathbf{S}}(\vec{G}\rho') \doteq \tilde{\mathbf{S}}(\vec{G}\sigma')$ is a constraint in Δ'_1 . We have the following sequence of equalities:

$$\mathbf{S}_0(\vec{G}\rho') \underset{1}{=} (\mathbf{S}_1 \otimes_{\mathbf{S}\mathcal{E}} \tilde{\mathbf{S}})(\vec{G}\rho') \underset{2}{=} \mathbf{S}_1(\tilde{\mathbf{S}}(\vec{G}\rho')) \underset{3}{=} \mathbf{S}_1(\tilde{\mathbf{S}}(\vec{G}\sigma')) \underset{4}{=} (\mathbf{S}_1 \otimes_{\mathbf{S}\mathcal{E}} \tilde{\mathbf{S}})(\vec{G}\sigma') \underset{5}{=} \mathbf{S}_0(\vec{G}\sigma')$$

as desired. Equalities 1 and 5 follow from the definition of \mathbf{S}_0 , equalities 2 and 4 follow from lemma 3.18, and equality 3 follows from the fact that \mathbf{S}_1 is a solution for Δ_1 together with part 1 in lemma 5.9. The argument works whether or not $\vec{G}\rho' \doteq \vec{G}\sigma'$ is $\vec{F}\rho \doteq \vec{F}\sigma$.

Second, we check that \mathbf{S}_0 is principal for Δ_0 . Let \mathbf{S}'_0 be an arbitrary solution for Δ_0 . Consider an arbitrary constraint $\vec{G}\rho' \doteq \vec{G}\sigma'$ in Δ_0 . The corresponding constraint in Δ'_1 is $\tilde{\mathbf{S}}(\vec{G}\rho') \doteq \tilde{\mathbf{S}}(\vec{G}\sigma')$. By lemma 5.11, there exists a solution \mathbf{S}'_1 for Δ_1 from \mathbf{S}'_0 such that:

$$(\#) \quad \mathbf{S}'_0(\vec{G}\rho') = \mathbf{S}'_1(\tilde{\mathbf{S}}(\vec{G}\rho')) = \mathbf{S}'_1(\tilde{\mathbf{S}}(\vec{G}\sigma')) = \mathbf{S}'_0(\vec{G}\sigma') .$$

By hypothesis, \mathbf{S}_1 is a principal solution for Δ_1 and, by part 2 in lemma 5.9, a principal solution for Δ'_1 . Hence, there is a substitution $\hat{\mathbf{S}}$ such that:

$$\begin{aligned} \mathbf{S}'_1(\tilde{\mathbf{S}}(\vec{G}\rho')) &= \hat{\mathbf{S}}(\mathbf{S}_1(\tilde{\mathbf{S}}(\vec{G}\rho'))) = \hat{\mathbf{S}}(\mathbf{S}_1 \otimes_{\mathbf{S}\mathcal{E}} \tilde{\mathbf{S}}(\vec{G}\rho')) = \hat{\mathbf{S}}(\mathbf{S}_0(\vec{G}\rho')) \quad \text{and} \\ \mathbf{S}'_1(\tilde{\mathbf{S}}(\vec{G}\sigma')) &= \hat{\mathbf{S}}(\mathbf{S}_1(\tilde{\mathbf{S}}(\vec{G}\sigma'))) = \hat{\mathbf{S}}(\mathbf{S}_1 \otimes_{\mathbf{S}\mathcal{E}} \tilde{\mathbf{S}}(\vec{G}\sigma')) = \hat{\mathbf{S}}(\mathbf{S}_0(\vec{G}\sigma')) . \end{aligned}$$

Together with (#), this implies that \mathbf{S}_0 is a principal solution for Δ_0 . □

To show that $\text{Unify}(\Delta)$ always terminates if Δ has a solution, we need to define a strictly decreasing measure on solutions \mathbf{S} , which we here take as $\text{size}(\mathbf{S})$.

Definition 5.13 (Size). We define the function $\text{size} : \mathbb{T}_{\square} \rightarrow \mathcal{N}$ by induction on \mathbb{T}_{\square} :

1. $\text{size}(\square) = \text{size}(\alpha) = 1$.
2. $\text{size}(\varphi \rightarrow \bar{\varphi}) = \text{size}(\varphi) + \text{size}(\bar{\varphi}) + 1$.
3. $\text{size}(\varphi \wedge \varphi') = \text{size}(\varphi) + \text{size}(\varphi') + 1$.
4. $\text{size}(F\varphi) = \text{size}(\varphi)$.

In words, $\text{size}(\varphi)$ is the number of nodes in the tree representation of φ . Occurrences of E-variables are not included in the count for $\text{size}(\varphi)$. Proper subsets of \mathbb{T}_{\square} are \mathbb{T} and \mathbb{E} ; it is therefore meaningful to use $\text{size}(\tau)$ and $\text{size}(e)$ for $\tau \in \mathbb{T}$ and $e \in \mathbb{E}$.

We extend the function $\text{size}(\)$ to an arbitrary substitution $\mathbf{S} : \text{Var} \rightarrow (\mathbb{E} \cup \mathbb{T}^{\rightarrow})$ with finite $\text{Dom}(\mathbf{S})$ as follows. Let:

$$\mathbf{S} = \{ \{ F_1 := e_1, \dots, F_m := e_m, \alpha_1 := \bar{\tau}_1, \dots, \alpha_n := \bar{\tau}_n \} \} ,$$

where $m + n \neq 0$. We define $\text{size}(\mathbf{S})$ by:

$$\text{size}(\mathbf{S}) = \text{size}(e_1) + \dots + \text{size}(e_m) + \text{size}(\bar{\tau}_1) + \dots + \text{size}(\bar{\tau}_n)$$

If $\text{Dom}(\mathbf{S}) = \{ \}$, we define $\text{size}(\mathbf{S}) = 0$. If $\text{Dom}(\mathbf{S})$ is infinite, we leave $\text{size}(\mathbf{S})$ undefined. □

Lemma 5.14 (Solutions with Finite Support Suffice). *Let Δ be a λ -compatible constraint set and let \mathbf{S} be a substitution. If \mathbf{S} is a solution for Δ , then we can construct a substitution \mathbf{S}' from \mathbf{S} such that:*

1. $\text{Dom}(\mathbf{S}')$ is finite.
2. \mathbf{S}' is a solution for Δ . □

Proof. This is an immediate consequence of lemma 3.19. □

Lemma 5.15 ('Unify' Decreases Solution Size). *Let Δ_0 be a λ -compatible constraint set, let Δ_1 be a constraint set, and let:*

$$\text{Unify}(\Delta_0, \mathbf{S}, \mathcal{E}) \Rightarrow \text{Unify}(\Delta_1, \tilde{\mathbf{S}} \otimes_{\mathcal{E}} \mathbf{S}, \mathcal{E})$$

for some \mathbf{S} , $\tilde{\mathbf{S}}$ and \mathcal{E} (which do not matter here). If \mathbf{S}_0 is a solution for Δ_0 with finite $\text{Dom}(\mathbf{S}_0)$, then we can construct a solution \mathbf{S}_1 for Δ_1 with finite $\text{Dom}(\mathbf{S}_1)$ such that $\text{size}(\mathbf{S}_1) < \text{size}(\mathbf{S}_0)$. □

Proof. This is a slight adjustment of the proof for lemma 5.11. The given solution \mathbf{S}_0 in the proof for lemma 5.11 can be assumed to have finite $\text{Dom}(\mathbf{S}_0)$, by lemma 5.14. Call $\hat{\mathbf{S}}_1$ the solution constructed for Δ_1 in lemma 5.11, to distinguish it from the solution \mathbf{S}_1 constructed here. It is clear that $\text{size}(\mathbf{S}_0) = \text{size}(\hat{\mathbf{S}}_1)$. We consider each of the 3 rewrite rules separately.

rule 1: The given constraint set $\Delta_0 = \Delta \cup \vec{F}\{\alpha \doteq \bar{\sigma}\}$ is first transformed to $\Delta'_1 = \{\{\alpha := \bar{\sigma}\} \Delta \cup \vec{F}\{\bar{\sigma} \doteq \bar{\sigma}\}\}$ and then to $\Delta_1 = \text{simplify}(\Delta'_1)$. Suppose the given solution \mathbf{S}_0 is such that $\mathbf{S}_0(\vec{F}\square) = e$. Then

$$\mathbf{S}_0(\vec{F}\alpha) = \hat{\mathbf{S}}_1(\vec{F}\alpha) = e[\alpha^{s_1}, \dots, \alpha^{s_n}]$$

where $\text{paths}(e) = (s_1, \dots, s_n)$. As the variables $\alpha^{s_1}, \dots, \alpha^{s_n}$ do not occur in Δ'_1 , we can define \mathbf{S}_1 by:

$$\mathbf{S}_1 v = \begin{cases} v & \text{if } v \in \{\alpha^{s_1}, \dots, \alpha^{s_n}\}, \\ \hat{\mathbf{S}}_1 v & \text{otherwise.} \end{cases}$$

Because $\{\alpha^{s_1}, \dots, \alpha^{s_n}\} \subseteq \text{Dom}(\mathbf{S}_0)$ while $\{\alpha^{s_1}, \dots, \alpha^{s_n}\} \cap \text{Dom}(\mathbf{S}_1) = \emptyset$, we conclude $\text{size}(\mathbf{S}_1) < \text{size}(\mathbf{S}_0)$.

rule 2: Identical to **rule 1**, except for the reversed polarities.

rule 3: Let $\Delta_0 = \Delta \cup \vec{G}\{F\bar{\rho} \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]\}$, where the constraint $F\bar{\rho} \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]$ induces a rewrite step according to **rule 3**. Δ_0 is transformed to $\Delta_1 = \text{simplify}(\Delta'_1)$ where

$$\Delta'_1 = \{[F := e]\Delta \cup \vec{G}\{e[\langle \bar{\rho} \rangle^{s_1}, \dots, \langle \bar{\rho} \rangle^{s_n}] \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n]\}\}$$

where $\text{paths}(e) = (s_1, \dots, s_n)$. Let $\text{paths}(\mathbf{S}_0(\vec{G}\square)) = (t_1, \dots, t_m)$. Because the variables F^{t_1}, \dots, F^{t_m} do not occur in Δ'_1 , we can define \mathbf{S}_1 by:

$$\mathbf{S}_1 v = \begin{cases} v\square & \text{if } v \in \{F^{t_1}, \dots, F^{t_m}\}, \\ \hat{\mathbf{S}}_1 v & \text{otherwise.} \end{cases}$$

Because $\{F^{t_1}, \dots, F^{t_m}\} \subseteq \text{Dom}(\mathbf{S}_0)$ while $\{F^{t_1}, \dots, F^{t_m}\} \cap \text{Dom}(\mathbf{S}_1) = \emptyset$, it follows $\text{size}(\mathbf{S}_1) < \text{size}(\mathbf{S}_0)$. \square

The following theorem shows that the algorithm is sound (i.e., the substitutions Unify produces when it terminates are in fact solutions) and complete (i.e., Unify produces a solution if there is one), as well as showing it produces principal solutions. We write $\xRightarrow{*}$ for the reflexive transitive closure of \Rightarrow .

Theorem 5.16 (Soundness, Completeness, & Principality). *Let Δ be a λ -compatible constraint set and let $\mathcal{E} = \text{E-env}(\Delta)$. Then:*

1. Δ has a solution if and only if $\text{Unify}(\text{simplify}(\Delta), \{\square\}, \mathcal{E}) \xRightarrow{*} \text{Unify}(\emptyset, \mathbf{S}, \mathcal{E})$ for some \mathbf{S} .

2. If $\text{Unify}(\text{simplify}(\Delta), \{\square\}, \mathcal{E}) \xRightarrow{*} \text{Unify}(\emptyset, \mathbf{S}, \mathcal{E})$, then \mathbf{S} is a principal solution for Δ . \square

Proof. Part 1 follows from lemmas 5.7, 5.8, 5.11, 5.12 and 5.15. For part 2, first note that if Unify terminates after $n \geq 1$ steps beyond the initial call, then the returned substitution \mathbf{S} applied to Δ produces a constraint set of the form:

$$\begin{aligned} \mathbf{S}\Delta &= (\mathbf{S}_n \otimes_{\mathcal{E}} (\mathbf{S}_{n-1} \otimes_{\mathcal{E}} (\mathbf{S}_{n-2} \otimes_{\mathcal{E}} \dots (\mathbf{S}_1 \otimes_{\mathcal{E}} \mathbf{S}_0) \dots)))\Delta \\ &= \mathbf{S}_n((\mathbf{S}_{n-1} \otimes_{\mathcal{E}} (\mathbf{S}_{n-2} \otimes_{\mathcal{E}} \dots (\mathbf{S}_1 \otimes_{\mathcal{E}} \mathbf{S}_0) \dots))\Delta) \\ &= \mathbf{S}_n(\mathbf{S}_{n-1}((\mathbf{S}_{n-2} \otimes_{\mathcal{E}} \dots (\mathbf{S}_1 \otimes_{\mathcal{E}} \mathbf{S}_0) \dots)\Delta)) \\ &\quad \vdots \\ &= \mathbf{S}_n(\mathbf{S}_{n-1}(\mathbf{S}_{n-2}(\dots ((\mathbf{S}_1 \otimes_{\mathcal{E}} \mathbf{S}_0)\Delta) \dots))) \\ &= \mathbf{S}_n(\mathbf{S}_{n-1}(\mathbf{S}_{n-2}(\dots (\mathbf{S}_1(\mathbf{S}_0\Delta)) \dots))) \end{aligned}$$

where $\mathbf{S}_0 = \{\square\}$ and $\mathbf{S}_1, \dots, \mathbf{S}_n$ are the n successive substitutions produced by the n rewrite steps. The first equality above is by the definition of \mathbf{S} , while all remaining equalities are by lemma 3.18. Starting from the

last equality above, we can also write the following sequence of equalities:

$$\begin{aligned}
\mathbf{S}\Delta &= \mathbf{S}_n(\mathbf{S}_{n-1}(\mathbf{S}_{n-2}(\mathbf{S}_{n-3}(\cdots(\mathbf{S}_1(\mathbf{S}_0\Delta))\cdots)))) \\
&= (\mathbf{S}_n \otimes_{\mathcal{E}_{n-1}} \mathbf{S}_{n-1})(\mathbf{S}_{n-2}(\mathbf{S}_{n-3}(\cdots(\mathbf{S}_1(\mathbf{S}_0\Delta))\cdots))) \\
&= ((\mathbf{S}_n \otimes_{\mathcal{E}_{n-1}} \mathbf{S}_{n-1}) \otimes_{\mathcal{E}_{n-2}} \mathbf{S}_{n-2})(\mathbf{S}_{n-3}(\cdots(\mathbf{S}_1(\mathbf{S}_0\Delta))\cdots)) \\
&\quad \vdots \\
&= (((\cdots(((\mathbf{S}_n \otimes_{\mathcal{E}_{n-1}} \mathbf{S}_{n-1}) \otimes_{\mathcal{E}_{n-2}} \mathbf{S}_{n-2}) \otimes_{\mathcal{E}_{n-3}} \mathbf{S}_{n-3}) \cdots) \otimes_{\mathcal{E}_1} \mathbf{S}_1)(\mathbf{S}_0\Delta) \\
&= (((\cdots(((\mathbf{S}_n \otimes_{\mathcal{E}_{n-1}} \mathbf{S}_{n-1}) \otimes_{\mathcal{E}_{n-2}} \mathbf{S}_{n-2}) \otimes_{\mathcal{E}_{n-3}} \mathbf{S}_{n-3}) \cdots) \otimes_{\mathcal{E}_1} \mathbf{S}_1) \otimes_{\mathcal{E}_0} \mathbf{S}_0)\Delta
\end{aligned}$$

where $\mathcal{E}_0 = \mathcal{E}$ and $\mathcal{E}_i = \mathbf{S}_{i-1}(\mathcal{E}_{i-1})$ for every $1 \leq i \leq n-1$. Invoking lemma 5.7, part 2 of lemma 5.8 and lemma 5.12, it is now readily checked that the substitution

$$(((\cdots(((\mathbf{S}_n \otimes_{\mathcal{E}_{n-1}} \mathbf{S}_{n-1}) \otimes_{\mathcal{E}_{n-2}} \mathbf{S}_{n-2}) \otimes_{\mathcal{E}_{n-3}} \mathbf{S}_{n-3}) \cdots) \otimes_{\mathcal{E}_1} \mathbf{S}_1) \otimes_{\mathcal{E}_0} \mathbf{S}_0)$$

is a principal solution for Δ . Hence, \mathbf{S} is a principal solution for Δ . \square

Note that Unify diverges exactly when there is no solution. The evaluation strategy does not matter, because lemmas 5.11 and 5.15 imply termination when there is a solution and lemma 5.12 implies divergence when no solution exists.

6 Type Inference Algorithm

This section defines a procedure which, given a λ -term M , generates a finite set $\Gamma(M)$ of constraints, the solvability of which is equivalent to the typability of the term M . We use this to prove the principality property for System II and to define a complete type-inference algorithm. An example of a run of the type-inference algorithm is presented in appendix A.

If $M = x$, for fresh $\alpha \in \text{TVar}_b$:	$ \begin{aligned} \text{Typ}(M) &= \alpha, \\ \text{Env}(M) &= \{x \mapsto \alpha\}, \\ \Gamma(M) &= \emptyset, \\ \text{Skel}(M) &= \langle \text{VAR}, \text{Env}(M) \vdash M : \alpha, \rangle. \end{aligned} $
If $M = (N_1 N_2)$, for fresh $F \in \text{EVar}_b$, $\beta \in \text{TVar}_b$:	$ \begin{aligned} \text{Typ}(M) &= \beta, \\ \text{Env}(M) &= \text{Env}(N_1) \wedge F \text{Env}(N_2), \\ \Gamma(M) &= \Gamma(N_1) \cup F \Gamma(N_2) \cup \{\text{Typ}(N_1) \doteq F \text{Typ}(N_2) \rightarrow \beta\}, \\ \text{Skel}(M) &= \langle \text{APP}, \text{Env}(M) \vdash M : \beta, \text{Skel}(N_1) (F \text{Skel}(N_2)) \rangle. \end{aligned} $
If $M = (\lambda x.N)$, for fresh $\alpha \in \text{TVar}_b$:	$ \begin{aligned} \text{Typ}(M) &= \begin{cases} \text{Env}(N)(x) \rightarrow \text{Typ}(N) & \text{if } \text{Env}(N)(x) \text{ defined,} \\ \alpha \rightarrow \text{Typ}(N) & \text{otherwise,} \end{cases} \\ \text{Env}(M) &= \text{Env}(N) \setminus x, \\ \Gamma(M) &= \Gamma(N), \\ \text{Skel}(M) &= \langle R, \text{Env}(M) \vdash M : \text{Typ}(M), \text{Skel}(N) \rangle \text{ where} \\ &\quad \text{if } x \in \text{FV}(N) \text{ then } R = \text{ABS-I} \text{ else } R = \text{ABS-K}. \end{aligned} $

Figure 6: Definition of $\Gamma(M)$, $\text{Skel}(M)$, $\text{Typ}(M)$, and $\text{Env}(M)$.

Definition 6.1 (Algorithm Generating Constraints and Skeleton). For every λ -term M , figure 6 gives an inductive definition of a set of constraints $\Gamma(M)$ and a derivation skeleton $\text{Skel}(M)$, defined simultaneously with a type $\text{Typ}(M)$ and a type environment $\text{Env}(M)$. In this definition, for a given subterm occurrence N , when a fresh variable is chosen, the same fresh variable must be used in $\text{Env}(N)$, $\text{Typ}(N)$, $\Gamma(N)$, and $\text{Skel}(N)$. The process of going from M to $\Gamma(M)$ and $\text{Skel}(M)$ is uniquely determined up to the choice of expansion variables and type variables. \square

Lemma 6.2 (Constraint Set is λ -Compatible). *Let M be an arbitrary λ -term. The constraint set $\Gamma(M)$ induced by M is λ -compatible.* \square

Proof. This is by induction on M . An appropriate induction hypothesis is stated using the functions $\text{Typ}(M)$, $\text{Env}(M)$ and $\Gamma(M)$ in figure 6 with polarities inserted. Accordingly, define:

- If $M = x$, then for fresh $\alpha \in \text{TVar}_b$:

$$\begin{aligned}\pm\text{Typ}(M) &= +\alpha , \\ \pm\text{Env}(M) &= \{x \mapsto -\alpha\} , \\ \pm\Gamma(M) &= \emptyset .\end{aligned}$$

- If $M = (NP)$, then for fresh $F \in \text{EVar}_b$ and fresh $\beta \in \text{TVar}_b$:

$$\begin{aligned}\pm\text{Typ}(M) &= +\beta , \\ \pm\text{Env}(M) &= \pm\text{Env}(N) \wedge -F \pm\text{Env}(P) , \\ \pm\Gamma(M) &= \pm\Gamma(N) \cup F \pm\Gamma(P) \cup \{\pm\text{Typ}(N) \doteq +F \pm\text{Typ}(P) \rightarrow -\beta\} .\end{aligned}$$

Note that an outer occurrence of F is inserted without a polarity in $\pm\text{Typ}(M)$, consistent with the use of polarities in definitions 4.4 and 4.8.

- If $M = (\lambda x.N)$, then for fresh $\alpha \in \text{TVar}_b$:

$$\begin{aligned}\pm\text{Typ}(M) &= \begin{cases} \pm\text{Env}(N)(x) \rightarrow \pm\text{Typ}(N) & \text{if } \pm\text{Env}(N)(x) \text{ defined ,} \\ -\alpha \rightarrow \pm\text{Typ}(N) & \text{otherwise ,} \end{cases} \\ \pm\text{Env}(M) &= \pm\text{Env}(N) \setminus x , \\ \pm\Gamma(M) &= \pm\Gamma(N) .\end{aligned}$$

By omitting all polarities in $\pm\text{Typ}(M)$, $\pm\text{Env}(M)$ and $\pm\Gamma(M)$, we get precisely $\text{Typ}(M)$, $\text{Env}(M)$ and $\Gamma(M)$, and we can also go from the latter to the former without ambiguity.

For each λ -term M , we define $\text{Triple}(M) = (\pm\text{Typ}(M), \pm\text{Env}(M), \pm\Gamma(M))$. The induction hypothesis specifies 6 properties of $\text{Triple}(M)$:

- (0) $\text{Triple}(M)$ is of the form

$$\begin{aligned}\pm\text{Typ}(M) &= (\bar{\rho})^+ , \\ \pm\text{Env}(M) &= \{x_1 : (\sigma_1)^-, \dots, x_m : (\sigma_m)^-\} , \\ \pm\Gamma(M) &= \{\vec{F}_1(\bar{\rho}_1)^+ \doteq \vec{F}_1(\bar{\sigma}_1)^-, \dots, \vec{F}_n(\bar{\rho}_n)^+ \doteq \vec{F}_n(\bar{\sigma}_n)^-\} ,\end{aligned}$$

for some $\bar{\rho}, \bar{\rho}_i \in \mathbb{R}^-$, $\bar{\sigma}_i \in \mathbb{S}^-$ and $\sigma_j \in \mathbb{S}$, and $\vec{F}_i \in \text{EVar}^*$, with $m, n \geq 0$ and $\text{Var}(\bar{\rho}_i) \cap \text{Var}(\bar{\sigma}_i) = \emptyset$ for every $1 \leq i \leq n$. Note that every constraint in $\Gamma(M)$ is good of form (a) in definition 4.3; good constraints of form (b) and form (c) are generated only once the process of β -unification is started.

- (1) $\text{Triple}(M)$ is well-named, i.e., given the form of $\text{Triple}(M)$ described in property (0), the type:

$$\bar{\rho} \wedge \sigma_1 \wedge \dots \wedge \sigma_m \wedge \vec{F}_1(\bar{\rho}_1 \wedge \bar{\sigma}_1) \wedge \dots \wedge \vec{F}_n(\bar{\rho}_n \wedge \bar{\sigma}_n)$$

is well-named.

- (2) If expansion variable $F \in \text{EVar}$ occurs in $\text{Triple}(M)$, it occurs exactly once as $+F$.
- (3) If type variable $\alpha \in \text{TVar}$ occurs in $\text{Triple}(M)$, it occurs exactly once as $-\alpha$ and at most once as $+\alpha$.
- (4) Identical to condition (D) in definition 4.8 with $\Delta = \Gamma(M)$.

(5) For every λ -term N , if neither M nor N is a subterm occurrence of the other, then

$$\text{Var}(\text{Triple}(M)) \cap \text{Var}(\text{Triple}(N)) = \emptyset .$$

Although we do not use it in the induction, it is worth pointing out in property (0), $\{x_1, \dots, x_m\} = \text{FV}(M)$ and n is the number of subterm occurrences in M that are applications.

Observe that (1), (2) and (3) here imply (A), (B) and (C) in definition 4.8, while (4) here is identical to (D) in definition 4.8. The hard part is to set up the induction hypothesis above; the rest of the induction is routine, if somewhat tedious.

Let $\text{IH}(M)$ denote the 6 parts of the induction hypothesis relativized to λ -term M . We omit the straightforward verification that $\text{IH}(x)$ is true, for every λ -variable x . Assuming that $\text{IH}(N)$ is true, it is also readily checked that $\text{IH}(\lambda x.N)$ is true, and we therefore omit the details.

Finally, assuming that $\text{IH}(N)$ and $\text{IH}(P)$ are true, we want to show that $\text{IH}(NP)$ is true. Part (0) in $\text{IH}(NP)$ follows from (0) in both $\text{IH}(N)$ and $\text{IH}(P)$, and the definitions of $\pm\text{Typ}(NP)$, $\pm\text{Env}(NP)$ and $\pm\Gamma(NP)$. Each of parts (1), (2), (3) and (5) in $\text{IH}(NP)$ follows from the corresponding parts in $\text{IH}(N)$ and $\text{IH}(P)$. For part (4) in $\text{IH}(NP)$, we need to show that $\text{graph}(\Gamma(NP))$ is acyclic. Let Δ_0 be the set:

$$\Delta_0 = \{\text{Typ}(N) \doteq F \text{Typ}(P) \rightarrow \beta\}.$$

This is the new constraint, of form (a) as specified in definition 4.3, which is added to $\Gamma(N)$ and $F\Gamma(P)$ to obtain $\Gamma(NP)$. Then

$$\begin{aligned} \text{graph}(\Gamma(NP)) &= \text{graph}(\Gamma(N)) \cup \text{graph}(\Gamma(P)) \cup \text{graph}(\Delta_0) \quad \text{where, by definition 4.6,} \\ \text{graph}(\Delta_0) &= \begin{cases} \{\alpha \rightsquigarrow F, \alpha \rightsquigarrow \beta\} & \text{if } \text{Typ}(N) = \alpha \in \text{TVar}, \\ \{v \rightsquigarrow \beta \mid v \in \text{Var}(\bar{\rho}) \text{ and } \text{E-path}(v, \Delta_0) = \varepsilon\} & \text{if } \text{Typ}(N) = \sigma \rightarrow \bar{\rho} \\ & \text{for some } \sigma \in \mathbb{S} \text{ and } \bar{\rho} \in \mathbb{R}^\rightarrow . \end{cases} \end{aligned}$$

The acyclicity of $\text{graph}(\Gamma(NP))$ follows from the acyclicity of $\text{graph}(\Gamma(N))$ by part 4 in $\text{IH}(N)$, the acyclicity of $\text{graph}(\Gamma(P))$ by part 4 in $\text{IH}(P)$, and the fact $\text{Var}(\text{Triple}(N)) \cap \text{Var}(\text{Triple}(P)) = \emptyset$ by part (5) in $\text{IH}(N)$ (or also part (5) in $\text{IH}(P)$). This last fact guarantees $v \not\rightsquigarrow w$ for all $v \in \text{Var}(F\Gamma(P))$ and $w \in \text{Var}(\Gamma(N))$. \square

REMARK 6.3. Not every λ -compatible constraint set is induced by a λ -term; for example, a λ -compatible constraint set Δ as simple as $\{\alpha \doteq F\beta \rightarrow \gamma\}$ is not induced by any λ -term. Moreover, the same constraint set Δ may be induced by different λ -terms; for example, $\Gamma(\lambda x.\lambda y.M) = \Gamma(\lambda y.\lambda x.M)$ for all λ -terms M . \square

Lemma 6.4 (All Derivations Instances of Skel(M)). *If \mathcal{D} is a derivation of System \mathbb{I} with final judgement $A \vdash M : \tau$, then there is a substitution \mathbf{S} such that $\mathcal{D} = \mathbf{S}(\text{Skel}(M))$.* \square

Proof. By induction on the structure of \mathcal{D} . (See the proof of theorem 22 in the appendix of [Kfo99].) \square

Theorem 6.5 (Constraint Set and Skeleton Equivalent). *Let M be a λ -term. Then a substitution \mathbf{S} is a solution for $\Gamma(M)$ if and only if $\mathbf{S}(\text{Skel}(M))$ is a derivation of System \mathbb{I} . Thus, $\Gamma(M)$ is solvable if and only if M is typable in System \mathbb{I} .* \square

Proof. By induction on M , using the definitions of $\Gamma(M)$, $\text{Skel}(M)$, and substitution together with lemma 2.19. \square

Corollary 6.6 (Undecidability of Beta-Unification). *It is undecidable whether an arbitrarily chosen λ -compatible constraint set Δ has a solution.* \square

From the principality property for λ -compatible β -unification, we can derive the following.

Theorem 6.7 (Principal Typings and Completeness of Type Inference). *Given λ -term M , let PT be the algorithm such that*

$$\text{PT}(M) = \begin{cases} \mathbf{S}(\text{Skel}(M)) & \text{if some evaluation of } \text{Unify}(\Gamma(M)) \text{ converges and returns } \mathbf{S}, \\ \text{undefined} & \text{if every evaluation of } \text{Unify}(\Gamma(M)) \text{ diverges,} \end{cases}$$

It holds that if M is typable in System \mathbb{I} , then $\text{PT}(M)$ returns a principal typing for M , else $\text{PT}(M)$ diverges. Thus, System \mathbb{I} has the principality property and PT is a complete type inference algorithm for System \mathbb{I} . \square

Proof. By lemma 6.2 and theorem 5.16, if $\mathbf{S} = \text{Unify}(\Gamma(M))$ is defined, it is a principal solution for $\Gamma(M)$. By theorem 6.5, $\mathbf{S}(\text{Skel}(M))$ is a typing for M . By lemma 6.4, if \mathcal{D} is a typing for M , then $\mathcal{D} = \mathbf{S}'(\text{Skel}(M))$ for some substitution \mathbf{S}' , which must be a solution for $\Gamma(M)$ by theorem 6.5. By definition 4.13, it must hold that $\mathbf{S}'(\Gamma(M)) = \mathbf{S}''(\mathbf{S}(\Gamma(M)))$ for some substitution \mathbf{S}'' . Thus, $\mathcal{D} = \mathbf{S}''(\text{PT}(M))$ ⁷, proving that $\text{PT}(M)$ is a principal typing for M . \square

7 Skeletons and Derivations at Finite Ranks

This section makes a finer analysis of the relationship between $\text{Skel}(M)$ and $\Gamma(M)$ for an arbitrary λ -term M . We then define the “rank” of skeletons and derivations, and the “rank” of solutions for constraint sets. Finally, we prove a correspondence between the rank of typings for M and the rank of solutions for $\Gamma(M)$.

Definition 7.1 (Left Types). For every skeleton \mathcal{S} we define the set of *left types of \mathcal{S}* , denoted $\text{left-types}(\mathcal{S})$, and the *final type of \mathcal{S}* , denoted $\text{final-type}(\mathcal{S})$. If \mathcal{S} is the skeleton $\langle R, J, Q_1 \cdots Q_n \rangle$ where J is the judgement $A \vdash_{\tau} M : \tau$, then

$$\begin{aligned} \text{final-type}(\mathcal{S}) &= \tau, \\ \text{left-types}(\mathcal{S}) &= \begin{cases} \emptyset & \text{if } n = 0 \text{ and } R = \text{VAR}, \\ \text{left-types}(Q_1) \cup \text{left-types}(Q_2) \cup \{\text{final-type}(Q_1)\} & \text{if } n = 2 \text{ and } R = \text{APP}, \\ \{F\tau \mid \tau \in \text{left-types}(Q_1)\} & \text{if } n = 1 \text{ and } R = F \in \text{EVar}, \\ \text{left-types}(Q_1) \cup \cdots \cup \text{left-types}(Q_n) & \text{otherwise.} \end{cases} \end{aligned}$$

A simple induction (omitted) shows that a type in $\text{left-types}(\mathcal{S})$ is always of the form $\vec{F}\bar{\tau}$ for some $\vec{F} \in \text{EVar}^*$ and $\bar{\tau} \in \mathbb{T}^{\rightarrow}$. In words, $\vec{F}\bar{\tau} \in \text{left-types}(\mathcal{S})$ iff $\bar{\tau}$ is the final type of the left premise Q_1 in a subskeleton $\langle \text{APP}, J, Q_1 Q_2 \rangle$ of \mathcal{S} . The number of types in $\text{left-types}(\mathcal{S})$ is therefore the number of times rule APP is used in \mathcal{S} .

The definition of $\text{left-types}(\mathcal{S})$ is somewhat complicated, but is adjusted in order to be exactly the set of left types of a corresponding constraint set. Let Δ be the constraint set $\{\tau_1 \doteq \tau'_1, \dots, \tau_n \doteq \tau'_n\}$. We define the set $\text{left-types}(\Delta)$ by:

$$\text{left-types}(\Delta) = \{\tau_1, \tau_2, \dots, \tau_n\}.$$

Lemma 7.3 relates $\text{left-types}(\mathcal{S})$ and $\text{left-types}(\Delta)$. \square

Definition 7.2 (The ‘Split’ Operation). The operation $\text{split}(\)$ is first defined on a single constraint $\tau \doteq \tau'$, where $\tau, \tau' \in \mathbb{T}$ are arbitrary:

$$\text{split}(\tau \doteq \tau') = \begin{cases} F \text{split}(\tau_0 \doteq \tau'_0) & \text{if } \tau = F\tau_0 \text{ and } \tau' = F\tau'_0, \\ \text{split}(\tau_1 \doteq \tau'_1) \cup \text{split}(\tau_2 \doteq \tau'_2) & \text{if } \tau = \tau_1 \wedge \tau_2 \text{ and } \tau' = \tau'_1 \wedge \tau'_2, \\ \{\tau \doteq \tau'\} & \text{otherwise.} \end{cases}$$

What $\text{split}(\)$ does is this:⁸ If τ and τ' are types such that $\tau = e[\tau_1, \dots, \tau_n]$ and $\tau' = e[\tau'_1, \dots, \tau'_n]$ where $\tau_i \neq \tau'_i$ for every $1 \leq i \leq n$, then

$$\text{split}(\tau \doteq \tau') = \{\vec{F}_1 \tau_1 \doteq \vec{F}_1 \tau'_1, \dots, \vec{F}_n \tau_n \doteq \vec{F}_n \tau'_n\},$$

where $\vec{F}_i = \text{E-path}(\square^{(i)}, e)$ for every $1 \leq i \leq n$. We extend the operation to constraint sets Δ . If $\Delta = \emptyset$, then $\text{split}(\Delta) = \emptyset$. If $\Delta = \{\tau \doteq \tau'\} \cup \Delta'$, then $\text{split}(\Delta) = \text{split}(\tau \doteq \tau') \cup \text{split}(\Delta')$. \square

Lemma 7.3 (Relating Left Types of Skeletons and Left Types of Constraint Sets). *Let M be a λ -term, $\mathcal{S} = \text{Skel}(M)$ and $\Delta = \Gamma(M)$. We then have, for every substitution \mathbf{S} :*

$$\text{left-types}(\mathbf{S}(\mathcal{S})) = \text{left-types}(\text{split}(\mathbf{S}\Delta))$$

In particular, if \mathbf{S} is the identity substitution, then $\text{left-types}(\mathcal{S}) = \text{left-types}(\Delta)$. \square

⁷This requires some additional facts about substitutions, e.g., the principal solution constructed by `Unify` is “safe” for composition in a certain sense. It also depends on the fact that $\text{E-path}(v, \text{Skel}(M)) = \text{E-path}(v, \Gamma(M))$ for all $v \in \text{Var}$.

⁸The operation $\text{split}(\)$ here does only a part of the operation `simplify`() in figure 5. The two are not identical.

Proof. By induction on the structure of M . To push the induction through, we prove a slightly stronger induction hypothesis, namely,

IH: For every substitution \mathbf{S} , every sequence \vec{F} of fresh and distinct basic E-variables, and every offset $t \in \{0, 1\}^*$, we have $\text{left-types}(\mathbf{S}\langle\vec{F}\mathcal{S}\rangle^t) = \text{left-types}(\text{split}(\mathbf{S}\langle\vec{F}\Delta\rangle^t))$.

We relativize IH to \mathcal{S} and Δ by writing $\text{IH}(\mathcal{S}, \Delta)$.

For the base case $M = x$, we have $\text{Skel}(M) = \langle \text{VAR}, x : \alpha \vdash x : \alpha, \rangle$ and $\Gamma(M) = \emptyset$. The desired conclusion is immediate in this case.

Proceeding inductively, let $M = N_1 N_2$. For $i = 1, 2$, let $\mathcal{S}_i = \text{Skel}(N_i)$ and $\Delta_i = \Gamma(N_i)$. Using the simultaneous definitions of $\text{Skel}(M)$ and $\Gamma(M)$ in figure 6, posing also $\bar{\rho}_i = \text{Typ}(N_i)$ and $A_i = \text{Env}(N_i)$ for $i = 1, 2$, we have:

$$\mathcal{S} = \text{Skel}(M) = \langle \text{APP}, A_1 \wedge H A_2 \vdash M : \beta, \mathcal{S}_1(H \mathcal{S}_2) \rangle, \quad (\text{a})$$

$$\Delta = \Gamma(M) = \Delta_1 \cup H \Delta_2 \cup \{\bar{\rho}_1 \doteq H \bar{\rho}_2 \rightarrow \beta\}, \quad (\text{b})$$

where $H \in \text{EVar}_b$ and $\beta \in \text{TVar}_b$ are fresh. Let \mathbf{S} be a substitution, \vec{F} a sequence of fresh and distinct basic E-variables, and $t \in \{0, 1\}^*$. Suppose $\mathbf{S}\langle\vec{F}\square\rangle^t = e$, $\text{paths}(e) = (u_1, \dots, u_n)$ where $n = \#\square(e) \geq 1$, and $\vec{G}_i = \text{E-path}(\square^{(i)}, e)$, for every $1 \leq i \leq n$. By definition 2.17, it is easy to see that:

$$\mathbf{S}\langle\vec{F}\mathcal{S}\rangle^t = e[\mathbf{S}\langle\mathcal{S}\rangle^{t \cdot u_1}, \dots, \mathbf{S}\langle\mathcal{S}\rangle^{t \cdot u_n}].$$

Hence, by definition 7.1 together with (a) above and the fact that $\text{final-type}(\mathcal{S}_1) = \bar{\rho}_1$, we have:

$$\begin{aligned} \text{left-types}(\mathbf{S}\langle\vec{F}\mathcal{S}\rangle^t) &= \bigcup_{1 \leq i \leq n} \vec{G}_i \text{left-types}(\mathbf{S}\langle\mathcal{S}\rangle^{t \cdot u_i}) \\ &= \bigcup_{1 \leq i \leq n} \vec{G}_i (\text{left-types}(\mathbf{S}\langle\mathcal{S}_1\rangle^{t \cdot u_i}) \cup \text{left-types}(\mathbf{S}\langle H \mathcal{S}_2 \rangle^{t \cdot u_i}) \cup \{\mathbf{S}\langle\bar{\rho}_1\rangle^{t \cdot u_i}\}) \end{aligned} \quad (\text{c})$$

For arbitrary $\tau_1, \dots, \tau_n, \tau'_1, \dots, \tau'_n \in \mathbb{T}$, it is readily checked that:

$$\text{split}(e[\tau_1, \dots, \tau_n] \doteq e[\tau'_1, \dots, \tau'_n]) = \vec{G}_1(\text{split}(\tau_1 \doteq \tau'_1)) \cup \dots \cup \vec{G}_n(\text{split}(\tau_n \doteq \tau'_n)).$$

Moreover, if τ_i or τ'_i is in \mathbb{T}^\rightarrow for every $1 \leq i \leq n$, then:

$$\text{split}(e[\tau_1, \dots, \tau_n] \doteq e[\tau'_1, \dots, \tau'_n]) = \vec{G}_1\{\tau_1 \doteq \tau'_1\} \cup \dots \cup \vec{G}_n\{\tau_n \doteq \tau'_n\},$$

which in turn implies that:

$$\text{left-types}(\text{split}(e[\tau_1, \dots, \tau_n] \doteq e[\tau'_1, \dots, \tau'_n])) = \{\vec{G}_1\tau_1, \dots, \vec{G}_n\tau_n\}.$$

Hence, by definition 7.1 together with (b), we also have:

$$\begin{aligned} \text{left-types}(\text{split}(\mathbf{S}\langle\vec{F}\Delta\rangle^t)) &= \text{left-types}(\text{split}(\mathbf{S}\langle\vec{F}\Delta_1\rangle^t)) \cup \text{left-types}(\text{split}(\mathbf{S}\langle\vec{F}H\Delta_2\rangle^t)) \cup \\ &\quad \text{left-types}(\text{split}(\mathbf{S}\langle\vec{F}\bar{\rho}_1\rangle^t \doteq \mathbf{S}\langle\vec{F}(H\bar{\rho}_2 \rightarrow \beta)\rangle^t)) \\ &= \bigcup_{1 \leq i \leq n} \vec{G}_i \left(\text{left-types}(\text{split}(\mathbf{S}\langle\Delta_1\rangle^{t \cdot u_i})) \cup \right. \\ &\quad \left. \text{left-types}(\text{split}(\mathbf{S}\langle H \Delta_2 \rangle^{t \cdot u_i})) \cup \{\mathbf{S}\langle\bar{\rho}_1\rangle^{t \cdot u_i}\} \right) \end{aligned} \quad (\text{d})$$

The equality of (c) and (d) follows from $\text{IH}(\mathcal{S}_1, \Delta_2)$ and $\text{IH}(\mathcal{S}_2, \Delta_2)$, which implies $\text{IH}(\mathcal{S}, \Delta)$ is also true.

The last case of the induction is $M = (\lambda x.N)$. There are two subcases, depending on whether $x \in \text{FV}(N)$ or not. Consider the first subcase only; the second is analyzed in the same way. Let $\mathcal{S}' = \text{Skel}(N)$ and $\Delta' = \Gamma(N)$. Using the simultaneous definitions of $\text{Skel}(M)$ and $\Gamma(M)$ in figure 6, posing also $\text{Typ}(N) = \bar{\rho}$ and $\text{Env}(N) = A[x \mapsto \sigma]$, we have:

$$\mathcal{S} = \text{Skel}(M) = \langle \text{ABS-I}, A \vdash M : \sigma \rightarrow \bar{\rho}, \mathcal{S}' \rangle, \quad (\text{e})$$

$$\Delta = \Gamma(M) = \Gamma(N) = \Delta'. \quad (\text{f})$$

That $\text{IH}(\mathcal{S}, \Delta)$ is true in this case follows from definition 7.1, together with (e), (f) and $\text{IH}(\mathcal{S}', \Delta')$. Remaining details omitted. \square

Definition 7.4 (Rank of Types). For every $s \in \{\mathsf{L}, \mathsf{R}, 0, 1\}^*$, let $\#_{\mathsf{L}}(s)$ denote the number of occurrences of L in s . Formally,

$$\#_{\mathsf{L}}(s) = \begin{cases} 0 & \text{if } s = \varepsilon, \\ 1 + \#_{\mathsf{L}}(t) & \text{if } s = \mathsf{L} \cdot t, \\ \#_{\mathsf{L}}(t) & \text{if } s = \mathsf{R} \cdot t \text{ or } 0 \cdot t \text{ or } 1 \cdot t. \end{cases}$$

Let $\tau \in \mathbb{T}$. There is a smallest (and, therefore, unique) $\varphi \in \mathbb{T}_{\square}$ with $n \geq 1$ holes such that

1. $\tau = \varphi[\tau_1, \dots, \tau_n]$ for some $\tau_1, \dots, \tau_n \in \mathbb{T}$.
2. None of the types in $\{\tau_1, \dots, \tau_n\}$ contains an occurrence of “ \wedge ”.

The *rank of hole* $\square^{(i)}$ in φ is given by $\text{hole-rank}(\square^{(i)}, \varphi) = \#_{\mathsf{L}}(\text{path}(\square^{(i)}, \varphi))$. If $\varphi = \square$, i.e., τ does not mention any “ \wedge ”, we define $\text{rank}(\tau) = 0$. If $\varphi \neq \square$, we define $\text{rank}(\tau)$ by:

$$\text{rank}(\tau) = 1 + \max\{\text{hole-rank}(\square^{(i)}, \varphi) \mid 1 \leq i \leq \#\square(\varphi)\}.$$

Let $T \subset \mathbb{T}$ be a non-empty finite subset of types. We define $\text{rank}(T)$ by:

$$\text{rank}(T) = \max\{\text{rank}(\tau) \mid \tau \in T\}.$$

If $T = \emptyset$, we define $\text{rank}(T) = 0$. This definition of rank is equivalent to others found in the literature. \square

Definition 7.5 (Rank of Skeletons, Derivations and Typings). Let J be the following judgement of System II:

$$x_1 : \tau_1, \dots, x_k : \tau_k \vdash M : \tau$$

for some appropriate types $\tau_1, \dots, \tau_k, \tau$ and λ -term M . We say that τ_1, \dots, τ_k are the *environment types* in J and τ is the *derived type* in J . If \mathcal{S} is a skeleton of System II, an *environment type* (resp., a *derived type*) in \mathcal{S} is an environment type (resp., a derived type) in one of the judgements of \mathcal{S} .

If \mathcal{S} is a skeleton of System II where every environment type has rank 0 and every derived type has rank 0, we write $\text{rank}(\mathcal{S}) = 0$ and say that \mathcal{S} is a *rank-0 skeleton*.

Let $k \geq 1$. If \mathcal{S} is a skeleton of System II where every environment type has rank $\leq k-1$ and every derived type has rank $\leq k$, we write $\text{rank}(\mathcal{S}) \leq k$ and say that \mathcal{S} is a *skeleton of rank at most k* . If $\text{rank}(\mathcal{S}) \leq k$ but $\text{rank}(\mathcal{S}) \not\leq k-1$, we write $\text{rank}(\mathcal{S}) = k$ and say that \mathcal{S} is a *rank- k skeleton*. A particular subset of the rank- k skeletons are the rank- k derivations, and a particular subset of the rank- k derivations are the rank- k typings. \square

Definition 7.6 (Lambda-Terms in Special Form). Let M be a λ -term. We say that M is *in special form* if $M = (\lambda y_1 \cdots \lambda y_n. N)z_1 \cdots z_n$ where N is not a λ -abstraction, $\text{FV}(N) \subseteq \{y_1, \dots, y_n\}$ and z_1, \dots, z_n are n distinct λ -variables, with $n \geq 0$.

Let $M = \lambda y_1 \cdots \lambda y_n. N$ be a λ -term (not necessarily in special form) where N is not a λ -abstraction, with $n \geq 0$, and let $\text{FV}(M) = \{x_1, \dots, x_m\}$ with $m \geq 0$. We define the *special form of M* by:

$$\text{special}(M) = (\lambda x_1 \cdots \lambda x_m \lambda y_1 \cdots \lambda y_n. N)z_1 \cdots z_{m+n},$$

where z_1, \dots, z_{m+n} are $m+n$ distinct fresh λ -variables. Clearly, $\text{special}(M)$ is a λ -term in special form. (It is not the case in general that if M is in special form, then $\text{special}(M) = M$.) \square

Lemma 7.7 (Relating Rank of Typings and Rank of Left Types). *Let M be a λ -term and let \mathcal{D} be a typing for M . If M is in special form, then $\text{rank}(\mathcal{D}) = \text{rank}(\text{left-types}(\mathcal{D}))$.* \square

The conclusion of the lemma is generally false if M is not in special form. Consider, for example, the λ -term $M = xx$: There is a typing \mathcal{D} for xx of rank 2, while $\text{rank}(\text{left-types}(\mathcal{D})) = 0$.

Proof. If \mathcal{D} is a derivation with final judgement $x_1 : \tau_1, \dots, x_m : \tau_m \vdash? P : \tau$, we define the set of types $\text{final-type}^*(\mathcal{D})$ by:

$$\text{final-type}^*(\mathcal{D}) = \{\tau_1 \rightarrow \alpha, \dots, \tau_m \rightarrow \alpha, \tau\}$$

where α is a fresh T-variable (not occurring anywhere in \mathcal{D}). Although the usual definition of “rank” allows us to write the equation:

$$\text{rank}(\text{final-type}^*(\mathcal{D})) = \text{rank}(\tau_1 \rightarrow \dots \rightarrow \tau_m \rightarrow \tau),$$

we have to avoid the type on the right-hand side of this equation because, if $\tau \notin \mathbb{T}^\rightarrow$, it is not legal according to our syntax of types (definition 2.3).

We also define $\text{left-types}^*(\mathcal{D}) = \text{left-types}(\mathcal{D}) \cup \text{final-type}^*(\mathcal{D})$. Without restrictions on the λ -term P , we now prove by induction on the structure of \mathcal{D} that:

$$\text{IH: } \text{rank}(\mathcal{D}) = \text{rank}(\text{left-types}^*(\mathcal{D})).$$

This is the induction hypothesis, which we write $\text{IH}(\mathcal{D})$ when relativized to \mathcal{D} .

For the base case, we have $\mathcal{D} = \langle \text{VAR}, x : \bar{\tau} \vdash x : \bar{\tau}, \rangle$ and $\text{left-types}^*(\mathcal{D}) = \{\bar{\tau} \rightarrow \alpha, \bar{\tau}\}$. This implies $\text{IH}(\mathcal{D})$ is true for the base case.

Suppose the final judgement in \mathcal{D} is obtained by using ABS-I, i.e., $\mathcal{D} = \langle \text{ABS-I}, A \vdash \lambda x.M : \tau \rightarrow \bar{\tau}, \mathcal{D}' \rangle$ where $\mathcal{D}' = \langle R, A[x \mapsto \tau] \vdash M : \bar{\tau}, \vec{Q} \rangle$ for some rule R and appropriate sequence \vec{Q} of subderivations. We now have the following sequence of equalities:

$$\begin{aligned} \text{rank}(\mathcal{D}) &= \text{rank}(\mathcal{D}') && \text{by definition of } \text{rank}(\mathcal{D}), \\ &= \text{rank}(\text{left-types}^*(\mathcal{D}')) && \text{by IH}(\mathcal{D}'), \\ &= \text{rank}((\text{left-types}^*(\mathcal{D}') - \{\tau \rightarrow \alpha, \bar{\tau}\}) \cup \{\tau \rightarrow \bar{\tau}\}) \\ &= \text{rank}(\text{left-types}^*(\mathcal{D})) && \text{by definition of } \text{left-types}^*(\mathcal{D}'), \end{aligned}$$

which implies $\text{IH}(\mathcal{D})$ is true for the case when ABS-I is the last rule used in \mathcal{D} .

We omit the case when the final judgement in \mathcal{D} is obtained by using ABS-K. This case is an easy variation of the preceding case, and we omit all the straightforward details.

Suppose the final judgement in \mathcal{D} is obtained by using APP, i.e., $\mathcal{D} = \langle \text{APP}, A_1 \wedge A_2 \vdash MN : \bar{\tau}, \mathcal{D}_1 \mathcal{D}_2 \rangle$ where $\mathcal{D}_1 = \langle R_1, A_1 \vdash M : \tau \rightarrow \bar{\tau}, \vec{Q}_1 \rangle$ and $\mathcal{D}_2 = \langle R_2, A_2 \vdash? N : \tau, \vec{Q}_2 \rangle$, for some rules R_1 and R_2 and appropriate sequences \vec{Q}_1 and \vec{Q}_2 of subderivations. Let

$$\begin{aligned} A_1 &= x_1 : \tau_1^1, \dots, x_m : \tau_m^1, y_1 : \tau_{m+1}^1, \dots, y_n : \tau_{m+n}^1 \\ A_2 &= x_1 : \tau_1^2, \dots, x_m : \tau_m^2, z_1 : \tau_{m+1}^2, \dots, z_p : \tau_{m+p}^2 \end{aligned}$$

for some $m, n, p \geq 0$, where the sets $\{x_1, \dots, x_m\}$, $\{y_1, \dots, y_n\}$ and $\{z_1, \dots, z_p\}$ are pairwise disjoint. This implies that

$$A_1 \wedge A_2 = x_1 : \tau_1^1 \wedge \tau_1^2, \dots, x_m : \tau_m^1 \wedge \tau_m^2, y_1 : \tau_{m+1}^1, \dots, y_n : \tau_{m+n}^1, z_1 : \tau_{m+1}^2, \dots, z_p : \tau_{m+p}^2$$

By the definition of $\text{final-type}^*(\)$, we thus have:

$$\begin{aligned} \text{final-type}^*(\mathcal{D}_1) &= \{\tau_1^1 \rightarrow \alpha, \dots, \tau_m^1 \rightarrow \alpha, \tau_{m+1}^1 \rightarrow \alpha, \dots, \tau_{m+n}^1 \rightarrow \alpha, \tau \rightarrow \bar{\tau}\}, \\ \text{final-type}^*(\mathcal{D}_2) &= \{\tau_1^2 \rightarrow \alpha, \dots, \tau_m^2 \rightarrow \alpha, \tau_{m+1}^2 \rightarrow \alpha, \dots, \tau_{m+p}^2 \rightarrow \alpha, \tau\}, \\ \text{final-type}^*(\mathcal{D}) &= \{\tau_1^1 \wedge \tau_1^2 \rightarrow \alpha, \dots, \tau_m^1 \wedge \tau_m^2 \rightarrow \alpha, \\ &\quad \tau_{m+1}^1 \rightarrow \alpha, \dots, \tau_{m+n}^1 \rightarrow \alpha, \tau_{m+1}^2 \rightarrow \alpha, \dots, \tau_{m+p}^2 \rightarrow \alpha, \bar{\tau}\}. \end{aligned}$$

A straightforward calculation now shows that:

$$(\dagger) \quad \text{rank}(\text{final-type}^*(\mathcal{D}) \cup \{\tau \rightarrow \bar{\tau}\}) \geq \text{rank}(\text{final-type}^*(\mathcal{D}_1) \cup \text{final-type}^*(\mathcal{D}_2)).$$

Finally, we have the following sequence of equalities:

$$\begin{aligned}
\text{rank}(\mathcal{D}) &= \max\{\text{rank}(\mathcal{D}_1), \text{rank}(\mathcal{D}_2), \text{rank}(\text{final-type}^*(\mathcal{D}))\} \\
&= \text{rank}(\text{left-types}^*(\mathcal{D}_1) \cup \text{left-types}^*(\mathcal{D}_2) \cup \text{final-type}^*(\mathcal{D})) \\
&= \text{rank}(\text{left-types}(\mathcal{D}) \cup \text{final-type}^*(\mathcal{D}_1) \cup \text{final-type}^*(\mathcal{D}_2) \cup \text{final-type}^*(\mathcal{D})) \\
&= \text{rank}(\text{left-types}(\mathcal{D}) \cup \text{final-type}^*(\mathcal{D})) \\
&= \text{rank}(\text{left-types}^*(\mathcal{D})) ,
\end{aligned}$$

where the second equality follows from IH(\mathcal{D}_1) and IH(\mathcal{D}_2), and the fourth equality from (\dagger) and the fact that $\{\tau \rightarrow \bar{\tau}\} \subseteq \text{left-types}(\mathcal{D})$. This establishes IH(\mathcal{D}) when APP is the last rule used in \mathcal{D} .

Suppose the final judgement in \mathcal{D} is obtained by using rule \wedge , i.e., $\mathcal{D} = \langle \wedge, A_1 \wedge A_2 \vdash_e M : \tau_1 \wedge \tau_2, \mathcal{D}_1 \mathcal{D}_2 \rangle$ where $\mathcal{D}_1 = \langle R_1, A_1 \vdash_{?} M : \tau_1, \vec{Q}_1 \rangle$ and $\mathcal{D}_2 = \langle R_2, A_2 \vdash_{?} M : \tau_2, \vec{Q}_2 \rangle$, for some rules R_1 and R_2 and appropriate sequences \vec{Q}_1 and \vec{Q}_2 of subderivations. Suppose $\text{FV}(M) = \{x_1, \dots, x_m\}$ for some $m \geq 0$, so that

$$\begin{aligned}
A_1 &= x_1 : \tau_1^1, \dots, x_m : \tau_m^1 \\
A_2 &= x_1 : \tau_1^2, \dots, x_m : \tau_m^2 \\
A_1 \wedge A_2 &= x_1 : \tau_1^1 \wedge \tau_1^2, \dots, x_m : \tau_m^1 \wedge \tau_m^2
\end{aligned}$$

Hence, by the definition of $\text{final-type}^*(\)$:

$$\begin{aligned}
\text{final-type}^*(\mathcal{D}_1) &= \{\tau_1^1 \rightarrow \alpha, \dots, \tau_m^1 \rightarrow \alpha, \tau_1\} , \\
\text{final-type}^*(\mathcal{D}_2) &= \{\tau_1^2 \rightarrow \alpha, \dots, \tau_m^2 \rightarrow \alpha, \tau_2\} , \\
\text{final-type}^*(\mathcal{D}) &= \{\tau_1^1 \wedge \tau_1^2 \rightarrow \alpha, \dots, \tau_m^1 \wedge \tau_m^2 \rightarrow \alpha, \tau_1 \wedge \tau_2\} .
\end{aligned}$$

It is therefore clear that:

$$(\ddagger) \quad \text{rank}(\text{final-type}^*(\mathcal{D})) \geq \text{rank}(\text{final-type}^*(\mathcal{D}_1) \cup \text{final-type}^*(\mathcal{D}_2)) .$$

We now have the following sequence of equalities:

$$\begin{aligned}
\text{rank}(\mathcal{D}) &= \max\{\text{rank}(\mathcal{D}_1), \text{rank}(\mathcal{D}_2), \text{rank}(\text{final-type}^*(\mathcal{D}))\} \\
&= \text{rank}(\text{left-types}^*(\mathcal{D}_1) \cup \text{left-types}^*(\mathcal{D}_2) \cup \text{final-type}^*(\mathcal{D})) \\
&= \text{rank}(\text{left-types}(\mathcal{D}) \cup \text{final-type}^*(\mathcal{D}_1) \cup \text{final-type}^*(\mathcal{D}_2) \cup \text{final-type}^*(\mathcal{D})) \\
&= \text{rank}(\text{left-types}(\mathcal{D}) \cup \text{final-type}^*(\mathcal{D})) \\
&= \text{rank}(\text{left-types}^*(\mathcal{D})) ,
\end{aligned}$$

where the second equality follows from IH(\mathcal{D}_1) and IH(\mathcal{D}_2), and the fourth equality from (\ddagger) . This establishes IH(\mathcal{D}) when rule \wedge is the last used in \mathcal{D} .

The last case of the induction is when the final judgement in \mathcal{D} is obtained by using rule F , for some $F \in \text{EVar}$. This case is straightforward and simpler than all the preceding cases. We omit all the details of this last case.

To complete the proof of lemma 7.7, consider an arbitrary λ -term M in special form and a typing \mathcal{D} for M . Then \mathcal{D} must have the following form (see definition 7.6):

$$\frac{\frac{\frac{\vdots}{\vdash \lambda y_1 \cdots \lambda y_n . N : \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \bar{\tau}} \text{ABS-x}}{z_1 : \tau_1 \vdash_{?} z_1 : \tau_1} R_1}{z_1 : \tau_1 \vdash (\lambda y_1 \cdots \lambda y_n . N) z_1 : \tau_2 \rightarrow \cdots \rightarrow \tau_n \rightarrow \bar{\tau}} \text{APP}}{\frac{\frac{\vdots}{z_2 : \tau_2 \vdash_{?} z_2 : \tau_2} R_2}{z_1 : \tau_1, \dots, z_n : \tau_n \vdash (\lambda y_1 \cdots \lambda y_n . N) z_1 \cdots z_{n-1} : \tau_n \rightarrow \bar{\tau}} \text{APP}}{\frac{\vdots}{z_1 : \tau_1, \dots, z_n : \tau_n \vdash (\lambda y_1 \cdots \lambda y_n . N) z_1 \cdots z_n : \bar{\tau}} \text{APP}} \text{APP}}$$

for some $\tau_1, \dots, \tau_n \in \mathbb{T}$ and $\bar{\tau} \in \mathbb{T}^\rightarrow$, where $\text{ABS-x} \in \{\text{ABS-I}, \text{ABS-K}\}$ and $R_i \in \{\text{VAR}, \wedge\} \cup \text{EVar}$ for every $1 \leq i \leq n$. By the definition of $\text{left-types}(\cdot)$ and $\text{left-types}^*(\cdot)$, we have:

$$\text{left-types}^*(\mathcal{D}) = \text{left-types}(\mathcal{D}) \cup \text{final-type}^*(\mathcal{D}) = \text{left-types}(\mathcal{D}) \cup \{\tau_1 \rightarrow \alpha, \dots, \tau_n \rightarrow \alpha, \bar{\tau}\}.$$

Because $\tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \bar{\tau} \in \text{left-types}(\mathcal{D})$ and $\text{rank}(\tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \bar{\tau}) = \text{rank}(\{\tau_1 \rightarrow \alpha, \dots, \tau_n \rightarrow \alpha, \bar{\tau}\})$, it now follows that $\text{rank}(\text{left-types}^*(\mathcal{D})) = \text{rank}(\text{left-types}(\mathcal{D}))$ which, together with $\text{IH}(\mathcal{D})$, implies the conclusion of the lemma. \square

Definition 7.8 (Rank of Solutions of Constraint Sets). Let Δ be the non-empty λ -compatible constraint set $\{\tau_1 \doteq \tau'_1, \dots, \tau_n \doteq \tau'_n\}$, and let $k \geq 0$. We say that a substitution $\mathbf{S} : \text{Var} \rightarrow (\mathbb{E} \cup \mathbb{T}^\rightarrow)$ is a *solution* for Δ of rank at most k provided:

1. \mathbf{S} is a solution for Δ .
2. $\max\{\text{rank}(\mathbf{S}\tau_1), \dots, \text{rank}(\mathbf{S}\tau_n)\} \leq k$.

We say that \mathbf{S} is a rank- k solution for Δ if \mathbf{S} is of rank $\leq k$ but not $\leq k-1$. If $\Delta = \emptyset$, every substitution is a solution for Δ , and we define the rank of a solution for $\Delta = \emptyset$ to be 0 (this is an arbitrary choice which does not affect the analysis). \square

Theorem 7.9 (Rank- k Typings vs. Rank- k Solutions). Let M be a λ -term and $M' = \text{special}(M)$. The following are equivalent conditions, for all $k \geq 0$:

1. There is a rank- k typing for M .
2. There is a rank- k typing for M' .
3. There is a rank- k solution for $\Gamma(M')$. \square

In the theorem statement we can restrict k to be $\neq 1$. It is easy to see there is no rank-1 typing for any λ -term P (as opposed to a derivation). Moreover, it can be shown that there is no rank-1 solution for $\Gamma(P)$ for any λ -term P , using the fact that all positive inner occurrences of expansion variables in $\Gamma(P)$ are at rank 2 (details omitted). This last assertion does not hold for λ -compatible constraint sets in general.

Proof. Let $k \geq 0$ be fixed throughout the proof. We first prove that, given an arbitrary $\tau \in \mathbb{T}$ and arbitrary $z \in \lambda\text{-Var}$, there is a derivation, call it $\mathcal{D}(z : \tau)$, whose final judgement is $z : \tau \vdash_{\text{?}} z : \tau$ where $\text{?} = \varepsilon$ or $\text{?} = e$ depending on whether $\tau \in \mathbb{T}^\rightarrow$ or not. For this, we write τ in the form $\tau = e[\bar{\tau}_1, \dots, \bar{\tau}_j]$ where e is an expansion with $\#_{\square}(e) = j \geq 1$ and $\bar{\tau}_1, \dots, \bar{\tau}_j \in \mathbb{T}^\rightarrow$. The construction of $\mathcal{D}(z : \tau)$ is a straightforward induction on the size of e , obtained by using repeatedly rule \wedge and rule F in figure 1; rule \wedge (resp. rule F) is used as many times as there are occurrences of “ \wedge ” (resp. occurrences of F) in e . We omit the details of this induction.

If $\text{rank}(\tau) = 0$, then $\text{rank}(\mathcal{D}(z : \tau)) = 0$. If $\text{rank}(\tau) = 1$, then $\text{rank}(\bar{\tau}_1) = \dots = \text{rank}(\bar{\tau}_j) = 0$ and $j \geq 2$, which in turn implies $\text{rank}(\mathcal{D}(z : \tau)) = 2$. If $\text{rank}(\tau) = k \geq 2$, then $\max\{\text{rank}(\bar{\tau}_1), \dots, \text{rank}(\bar{\tau}_j)\} = k$ which in turn implies $\text{rank}(\mathcal{D}(z : \tau)) = k + 1$. Note that for all $\tau \in \mathbb{T}$, it holds that $\text{rank}(\mathcal{D}(z : \tau)) \neq 1$.

We now prove the equivalence of parts 1 and 2 in the theorem statement: There is a rank- k typing for M iff there is a rank- k typing for M' . The right-to-left implication is easy to see, because M is a subterm of M' . For the left-to-right implication, suppose \mathcal{D} is a rank- k typing for M . Hence, \mathcal{D} is of the form $\mathcal{D} = \langle R, J, \bar{Q} \rangle$ where J is the judgement $x_1 : \tau_1, \dots, x_m : \tau_m \vdash M : \tau$. We can assume that $R \in \{\text{VAR}, \text{ABS-I}, \text{ABS-K}, \text{APP}\}$ in the judgement J , which must therefore be of the form (see definition 7.6):

$$x_1 : \tau_1, \dots, x_m : \tau_m \vdash \lambda y_1 \dots \lambda y_n. N : \tau_{m+1} \rightarrow \dots \rightarrow \tau_{m+n} \rightarrow \bar{\tau}$$

for some $\tau_1, \dots, \tau_{m+n} \in \mathbb{T}$ and $\bar{\tau} \in \mathbb{T}^\rightarrow$, where $n \geq 0$ and N is not a λ -abstraction.

If $\text{rank}(\mathcal{D}) = 0$, then $\text{rank}(\tau_i) = 0$ for every $i \in \{1, \dots, m+n\}$. There are no rank-1 typings, i.e., typings of rank at most 1 but not 0. If $\text{rank}(\mathcal{D}) = k \geq 2$, then $\text{rank}(\tau_i) \leq k-1$ for every $i \in \{1, \dots, m+n\}$. The

desired typing \mathcal{D}' for M' has the following form:

$$\begin{array}{c}
\frac{\vec{Q}}{\frac{x_1 : \tau_1, \dots, x_m : \tau_m \vdash M : \tau_{m+1} \rightarrow \dots \rightarrow \tau_{m+n} \rightarrow \bar{\tau}}{\text{ABS-I}} R} \\
\vdots \\
\frac{\vdash \lambda x_1 \dots \lambda x_m. M : \tau_1 \rightarrow \dots \rightarrow \tau_{m+n} \rightarrow \bar{\tau}}{\text{ABS-I}} \quad \mathcal{D}(z_1 : \tau_1) \\
\frac{\frac{z_1 : \tau_1 \vdash (\lambda x_1 \dots \lambda x_m. M) z_1 : \tau_2 \rightarrow \dots \rightarrow \tau_{m+n} \rightarrow \bar{\tau}}{\text{APP}} \quad \mathcal{D}(z_2 : \tau_2)}{\text{APP}} \\
\vdots \\
\frac{\mathcal{D}(z_{m+n} : \tau_{m+n})}{z_1 : \tau_1, \dots, z_{m+n} : \tau_{m+n} \vdash (\lambda x_1 \dots \lambda x_m. M) z_1 \dots z_{m+n} : \bar{\tau}} \text{APP}
\end{array}$$

It is easy to check that if $\text{rank}(\mathcal{D}) = 0$ then $\text{rank}(\mathcal{D}') = 0$, and if $\text{rank}(\mathcal{D}) \geq 2$ then $\text{rank}(\mathcal{D}') = \text{rank}(\mathcal{D})$.

We next prove the equivalence of parts 2 and 3 in the theorem statement: There is a rank- k typing for M' iff there is a rank- k solution for $\Gamma(M')$. Let $\mathcal{S}' = \text{Skel}(M')$ and $\Delta' = \Gamma(M')$. By lemma 6.4 and theorem 6.5, there is a typing \mathcal{D}' for M' iff there is a solution \mathbf{S} for Δ' such that $\mathcal{D}' = \mathbf{S}(\mathcal{S}')$. We also have:

$$\text{left-types}(\mathcal{D}') = \text{left-types}(\mathbf{S}(\mathcal{S}')) = \text{left-types}(\text{split}(\mathbf{S}\Delta'))$$

where the second equality follows from lemma 7.3. Hence, we also have:

$$\begin{aligned}
\text{rank}(\mathcal{D}') = k & \quad \text{iff} \quad \text{rank}(\text{left-types}(\mathcal{D}')) = k \\
& \quad \text{iff} \quad \text{rank}(\text{left-types}(\mathbf{S}(\mathcal{S}'))) = k \\
& \quad \text{iff} \quad \text{rank}(\text{left-types}(\text{split}(\mathbf{S}\Delta'))) = k \\
& \quad \text{iff} \quad \mathbf{S} \text{ is a rank-}k \text{ solution for } \Delta',
\end{aligned}$$

where the first “iff” is true by lemma 7.7 and the last “iff” follows from the fact that $k \neq 1$. \square

8 Termination and Decidability at Finite Ranks

This section defines `UnifyFR`, an adaptation of algorithm `Unify` which produces a solution \mathbf{S} with bounded *rank* k for a λ -compatible constraint set Δ . The definition of `UnifyFR` differs from `Unify` only in the “mode of operation” as presented in figure 7. The invocation of `UnifyFR` on Δ at rank k produces a solution \mathbf{S} if `Unify`(Δ) produces \mathbf{S} and $\text{rank}(\mathbf{S}\Delta) \leq k$. Otherwise, `UnifyFR` halts indicating failure, unlike `Unify` which diverges if it can not find a solution.

Note that the principality of solutions produced by `UnifyFR` follows from the principality of solutions produced by `Unify`.

The presentation of `UnifyFR` in figure 7 includes two new operations, “the coding of a constraint set Δ as a pair (τ, τ') ” and “the decomposition of a pair (τ, τ') as a constraint set $\Delta(\tau, \tau')$ ”. These are precisely stated in definitions 8.1 and 8.2.

Definition 8.1 (From Constraint Sets to Constraint Pairs). The coding $\lceil \Delta \rceil$ of a constraint set $\Delta = \{\tau_1 \doteq \tau'_1, \dots, \tau_n \doteq \tau'_n\}$ is the pair (τ, τ') given by:

$$\lceil \Delta \rceil = (\tau, \tau') = ((\tau_1 \wedge \dots \wedge (\tau_{n-1} \wedge \tau_n)), (\tau'_1 \wedge \dots \wedge (\tau'_{n-1} \wedge \tau'_n))). \quad \square$$

To show that for a fixed $k \geq 0$, an evaluation of `UnifyFR`(Δ, k) terminates, we need to reason about the *rank of a constraint* in a constraint set. The following definitions support this.

Definition 8.2 (λ -Compatible Pairs). Let (τ, τ') be a pair of types. We define its *constraint decomposition sequence* $d(\tau, \tau') = (\varphi, \hat{\tau}_1, \dots, \hat{\tau}_n, \hat{\tau}'_1, \dots, \hat{\tau}'_n)$ with $1 + 2n$ entries, where $\varphi \in \mathbb{T}_\square$ with $n \geq 0$ holes is the largest (and therefore unique) type context such that

1. $\tau = \varphi[\tau_1, \dots, \tau_n]$ and $\tau' = \varphi[\tau'_1, \dots, \tau'_n]$, where $\{\tau_i, \tau'_i\} = \{\hat{\tau}_i, \hat{\tau}'_i\}$ for every $1 \leq i \leq n$.

Metavariable conventions:

- $\bar{\rho} \in \mathbb{R}^\rightarrow$, $\rho \in \mathbb{R}$, $\bar{\sigma}, \bar{\sigma}_i \in \mathbb{S}^\rightarrow$, $\sigma \in \mathbb{S}$, $\tau, \tau' \in \mathbb{T}$, $e \in \mathbb{E}$, $\alpha \in \text{TVar}$, $F \in \text{EVar}$.

Mode of operation:

- Initial call: $\text{UnifyFR}(\Delta, k) \Rightarrow \text{UnifyFR}(\ulcorner \Delta \urcorner, \{\!\! \{ \} \!\!\}, \text{E-env}(\Delta), k)$ where $k \geq 0$.
- Final call: $\text{UnifyFR}((\tau, \tau), \mathbf{S}, \mathcal{E}, k) \Rightarrow \mathbf{S}$.
- $\text{UnifyFR}((\tau_0, \tau'_0), \mathbf{S}_0, \mathcal{E}, k) \Rightarrow \text{UnifyFR}((\tau_1, \tau'_1), \mathbf{S}_1, \mathcal{E}, k)$, provided:
 - $\Delta(\tau_0, \tau'_0) = \Delta \cup \vec{F}\{\rho \doteq \sigma\}$ and $\rho \doteq \sigma \Rightarrow \mathbf{S}$ is an instance of one of the rewrite rules.
 - $(\tau_1, \tau'_1) = (\mathbf{S}\tau_0, \mathbf{S}\tau'_0)$ and $\mathbf{S}_1 = \mathbf{S} \otimes_{\mathcal{E}} \mathbf{S}_0$.
 - $\text{rank}(\tau_1) \leq k$ and $\text{rank}(\tau'_1) \leq k$.

Rewrite rules:

$$\begin{aligned} \alpha \doteq \bar{\sigma} &\Rightarrow \{\!\! \{ \alpha := \bar{\sigma} \} \!\!\} && \text{(rule 1)} \\ \bar{\rho} \doteq \alpha &\Rightarrow \{\!\! \{ \alpha := \bar{\rho} \} \!\!\} && \text{(rule 2)} \\ F\bar{\rho} \doteq e[\bar{\sigma}_1, \dots, \bar{\sigma}_n] &\Rightarrow \{\!\! \{ F := e \} \!\!\} && \text{(rule 3)} \end{aligned}$$

Applying substitutions to constraint sets:

- $\mathbf{S}\emptyset = \emptyset$.
- $\mathbf{S}(\{\tau \doteq \tau'\} \cup \Delta) = \{\mathbf{S}\tau \doteq \mathbf{S}\tau'\} \cup \mathbf{S}\Delta$.

Figure 7: Algorithm UnifyFR (every part other than “Mode of operation” is copied from figure 5).

2. For every $1 \leq i \leq n$, if $\text{hole-rank}(\square^{(i)}, \varphi)$ is even, then $(\tau_i, \tau'_i) = (\hat{\tau}_i, \hat{\tau}'_i)$.
3. For every $1 \leq i \leq n$, if $\text{hole-rank}(\square^{(i)}, \varphi)$ is odd, then $(\tau_i, \tau'_i) = (\hat{\tau}'_i, \hat{\tau}_i)$.

If $\vec{G}_i = \text{E-path}(\square^{(i)}, \varphi)$ for $1 \leq i \leq n$, then the *constraint set decomposition* of (τ, τ') is:

$$\Delta(\tau, \tau') = \{\vec{G}_1(\hat{\tau}_1 \doteq \hat{\tau}'_1), \dots, \vec{G}_n(\hat{\tau}_n \doteq \hat{\tau}'_n)\}.$$

If $\Delta(\tau, \tau')$ is a λ -compatible constraint set, then we say that (τ, τ') is a λ -compatible pair. In this case, $\hat{\tau}_i \in \mathbb{R}$ and $\hat{\tau}'_i \in \mathbb{S}$ for $1 \leq i \leq n$, so we can let $\hat{\tau}_i = \rho_i$ and $\hat{\tau}'_i = \sigma_i$ for $1 \leq i \leq n$ and write $\Delta(\tau, \tau')$ in the form:

$$\Delta(\tau, \tau') = \{\vec{G}_1(\rho_1 \doteq \sigma_1), \dots, \vec{G}_n(\rho_n \doteq \sigma_n)\}.$$

Note that $\text{simplify}(\Delta(\tau, \tau')) = \Delta(\tau, \tau')$, because $d(\tau, \tau')$ chooses the largest φ with the stated property. For the same reason, note also that $\Delta(\tau, \tau) = \emptyset$. We define the *rank of constraint $\vec{G}_i(\rho_i \doteq \sigma_i)$ in (τ, τ')* :

$$\text{rank}(\vec{G}_i(\rho_i \doteq \sigma_i), (\tau, \tau')) = \text{hole-rank}(\square^{(i)}, \varphi).$$

We also define $h(\tau, \tau')$:

$$h(\tau, \tau') = \min\{\text{hole-rank}(\square^{(i)}, \varphi) \mid 1 \leq i \leq n\},$$

i.e., $h(\tau, \tau')$ is a lower bound on the L-distance of all the holes in φ from its root (viewed as a binary tree). If $\tau = \tau' = \varphi$, i.e., φ has 0 holes, we leave $h(\tau, \tau')$ undefined. \square

Definition 8.3 (Successful and Unsuccessful Evaluations of UnifyFR). Let Δ be a λ -compatible constraint set and $k \geq 0$. Consider a fixed, but otherwise arbitrary, evaluation of $\text{UnifyFR}(\Delta, k)$:

$$\text{UnifyFR}((\tau_0, \tau'_0), \{\!\! \{ \} \!\!\}, \mathcal{E}, k) \Rightarrow \text{UnifyFR}((\tau_1, \tau'_1), \mathbf{S}_1, \mathcal{E}, k) \Rightarrow \dots \Rightarrow \text{UnifyFR}((\tau_i, \tau'_i), \mathbf{S}_i, \mathcal{E}, k) \Rightarrow \dots,$$

where $(\tau_0, \tau'_0) = \lceil \Delta \rceil$ and $\mathcal{E} = \mathbf{E}\text{-env}(\Delta)$. We say this evaluation *succeeds* if it halts at the i th call with $\tau_i = \tau'_i$, for some $i \geq 0$, in which case it also returns the substitution \mathbf{S}_i .

We say this evaluation of $\text{UnifyFR}(\Delta, k)$ *fails* if either it diverges or it halts at the i th call with $\tau_i \neq \tau'_i$, for some $i \geq 0$. In the latter case, we also say that the evaluation *halts unsuccessfully*, which means that $\tau_i \neq \tau'_i$ and for all λ -compatible pairs (τ, τ') and all substitutions \mathbf{S} ,

$$\text{UnifyFR}((\tau_i, \tau'_i), \mathbf{S}_i, \mathcal{E}, k) \not\approx \text{UnifyFR}((\tau, \tau'), \mathbf{S}, \mathcal{E}, k),$$

i.e., the evaluation is unable to continue beyond the i th call. \square

Definition 8.4 (Evaluating λ -Compatible Pairs). Let (τ_0, τ'_0) and (τ_1, τ'_1) be λ -compatible pairs. Let **rule a** be one of the 3 rules listed in figure 5. We write

$$(\tau_0, \tau'_0) \xRightarrow{\mathbf{a}} (\tau_1, \tau'_1)$$

iff $\Delta(\tau_0, \tau'_0) = \{\vec{G}_1(\rho_1 \doteq \sigma_1), \dots, \vec{G}_n(\rho_n \doteq \sigma_n)\}$ and there is $i \in \{1, \dots, n\}$ such that:

1. $\rho_i \doteq \sigma_i \Rightarrow \mathbf{S}$ is an instance of **rule a**.
2. $(\tau_1, \tau'_1) = (\mathbf{S}\tau_0, \mathbf{S}\tau'_0)$.

In such a case, we say that (τ_0, τ'_0) *evaluates to*, or *reduces to*, (τ_1, τ'_1) by **rule a** using constraint $\vec{G}_i(\rho_i \doteq \sigma_i)$. Moreover, if $\text{rank}(\vec{G}_i(\rho_i \doteq \sigma_i), (\tau_0, \tau'_0)) = k$, we say that the constraint $\rho_i \doteq \sigma_i$ is *at rank k* and that the evaluation from (τ_0, τ'_0) to (τ_1, τ'_1) is also *at rank k*, indicated by writing

$$(\tau_0, \tau'_0) \xrightarrow{(\mathbf{a}, k)} (\tau_1, \tau'_1).$$

We write $(\tau_0, \tau'_0) \Rightarrow (\tau_1, \tau'_1)$ in case $(\tau_0, \tau'_0) \xRightarrow{\mathbf{a}} (\tau_1, \tau'_1)$ for some **rule a**, and $\xRightarrow{*}$ for the reflexive transitive closure of \Rightarrow . Let $\mathcal{R} \subseteq \{\mathbf{1}, \mathbf{2}, \mathbf{3}\}$. Let $(\tau_0, \tau'_0) \Rightarrow \dots \Rightarrow (\tau_n, \tau'_n)$ be an evaluation sequence with $n \geq 1$ steps. We write $(\tau_0, \tau'_0) \xrightarrow{\mathcal{R}}^n (\tau_n, \tau'_n)$ to indicate that the evaluation has n steps, and that each step is carried out using **rule a** for some $\mathbf{a} \in \mathcal{R}$.

Finally, if there is no pair (τ_1, τ'_1) such that $(\tau_0, \tau'_0) \xrightarrow{\mathcal{R}} (\tau_1, \tau'_1)$, then we say that the pair (τ_0, τ'_0) is in *\mathcal{R} -normal form*. \square

Lemma 8.5 (Evaluating without Rule 3). Let $\mathcal{R} = \{\mathbf{1}, \mathbf{2}\}$ and (τ_0, τ'_0) a λ -compatible pair. Then there is a bound $M(\tau_0, \tau'_0)$ solely depending on (τ_0, τ'_0) such that for every evaluation with \mathcal{R} , say $(\tau_0, \tau'_0) \xrightarrow{\mathcal{R}}^n (\tau_1, \tau'_1)$, we have $n \leq M(\tau_0, \tau'_0)$. In words, a diverging evaluation of (τ_0, τ'_0) must use **rule 3** infinitely many times. \square

Proof. Consider a single evaluation step $(\tau_0, \tau'_0) \xrightarrow{\mathcal{R}} (\tau_1, \tau'_1)$. Let $\Delta_0 = \Delta(\tau_0, \tau'_0)$. and $\Delta_1 = \Delta(\tau_1, \tau'_1)$. Then $\text{TVar}(\Delta_1)$ is a strict subset of $\text{TVar}(\Delta_0)$. The desired bound $M(\tau_0, \tau'_0)$ is therefore the size of the set $\text{TVar}(\Delta_0)$. \square

Definition 8.6 (E-Redexes in λ -Compatible Pairs). Let (τ, τ') be a λ -compatible pair and $k \geq 0$. We define the set of *E-redexes of (τ, τ') at rank k* as the following subset of the constraints in $\Delta(\tau, \tau')$:

$$\text{E-redexes}((\tau, \tau'), k) = \{ \vec{G}(F\bar{\rho} \doteq \sigma) \in \Delta(\tau, \tau') \mid \text{rank}(\vec{G}(F\bar{\rho} \doteq \sigma), (\tau, \tau')) = k \}. \quad \square$$

Lemma 8.7 (Ordering E-Redexes in λ -Compatible Pairs). Let (τ, τ') be a λ -compatible pair and fix the integer $k \geq 0$. Then we can list the constraints in $\text{E-redexes}((\tau, \tau'), k)$ in a particular order:

$$\vec{G}_1(F_1\bar{\rho}_1 \doteq \sigma_1), \vec{G}_2(F_2\bar{\rho}_2 \doteq \sigma_2), \dots, \vec{G}_n(F_n\bar{\rho}_n \doteq \sigma_n)$$

such that for every $1 \leq j \leq n$:

$$F_j \notin \text{EVar}(\vec{G}_1) \cup \dots \cup \text{EVar}(\vec{G}_{j-1}).$$

In particular, for $j = n$, we have $F_n \notin \text{EVar}(\vec{G}_1) \cup \dots \cup \text{EVar}(\vec{G}_{n-1})$. \square

Proof. Let $A = \{F_1, \dots, F_n\}$ and $B = \text{EVar} - A$. If $\vec{G} \in \text{EVar}^*$, we write $\vec{G}^{\mathcal{B}}$ for the sequence in A^* obtained from \vec{G} after erasing all E-variables in B . We define disjoint subsets A_1, A_2, A_3, \dots of A as follows:

$$A_1 = \{F \in \text{EVar} \mid \vec{G}(F\bar{\rho} \doteq \sigma) \in \text{E-redexes}((\tau, \tau'), k) \text{ and } \vec{G}^{\mathcal{B}} = \varepsilon\}.$$

If $\text{E-redexes}((\tau, \tau'), k) \neq \emptyset$, then clearly $A_1 \neq \emptyset$. For every $i \geq 1$, define:

$$A_{i+1} = \{F \in \text{EVar} \mid \vec{G}(F\bar{\rho} \doteq \vec{G}\sigma) \in \text{E-redexes}((\tau, \tau'), k) \text{ and } \vec{G}^{\mathcal{B}} \in A_1 \cdots A_i\}.$$

This definition of A_1, A_2, A_3, \dots is not circular, because $\Delta(\tau, \tau')$ is a λ -compatible constraint set, implying that $\text{E-path}(F, \Delta(\tau, \tau'))$ is uniquely defined without repeated entries, for every $F \in \text{EVar}$. Because A is finite, there is a smallest p such that $A = A_1 \cup \dots \cup A_p$. Moreover, by definition, if $A_{i+1} \neq \emptyset$ then $A_i \neq \emptyset$. Hence, assuming $A \neq \emptyset$, we have a partition of A into $p \geq 1$ disjoint (non-empty) subsets. We partition $\text{E-redexes}((\tau, \tau'), k)$ similarly, by defining a subset Δ_i of constraints, one for every $i \geq 1$:

$$\Delta_i = \{\vec{G}(F\bar{\rho} \doteq \sigma) \in \text{E-redexes}((\tau, \tau'), k) \mid F \in A_i\}$$

Clearly, $\text{E-redexes}((\tau, \tau'), k) = \Delta_1 \cup \dots \cup \Delta_p$. For every $1 \leq i \leq p$, let $A_i = \{F_{i,1}, \dots, F_{i,n_i}\}$. Thus, in particular, $n = n_1 + n_2 + \dots + n_p$. To conclude the proof, we assign a fixed but otherwise arbitrary order to each Δ_i separately:

$$\vec{G}_{i,1}(F_{i,1}\bar{\rho}_{i,1} \doteq \sigma_{i,1}), \dots, \vec{G}_{i,n_i}(F_{i,n_i}\bar{\rho}_{i,n_i} \doteq \sigma_{i,n_i}).$$

The sequence $1, 2, \dots, n$ in the lemma statement is just the sequence (appropriately renamed):

$$(1, 1), \dots, (1, n_1), (2, 1), \dots, (2, n_2), \dots, (p, 1), \dots, (p, n_p).$$

If $\vec{G}(F\bar{\rho} \doteq \sigma)$ is in Δ_i , for some $1 \leq i \leq p$, then $\text{E-path}(F, \Delta(\tau, \tau')) = \vec{G}$ with $|\vec{G}^{\mathcal{B}}| = i - 1$ by construction. This, together with the fact that the E-path of a variable is uniquely defined and does not contain repeated entries, implies that for all $1 \leq j \leq p$ and all $1 \leq r \leq n_j$ it holds that

$$F_{j,r} \notin \text{EVar}(\vec{G}_{i,q})$$

where either $i = j$ and $1 \leq q < r$, or $1 \leq i < j$ and $1 \leq q \leq n_i$. The conclusion of the lemma follows. \square

Definition 8.8 (Conservative E-Redexes in λ -Compatible Pairs). Let (τ, τ') be a λ -compatible pair, $k \geq 0$, and $\vec{G}(F\bar{\rho} \doteq \sigma)$ be an E-redex of (τ, τ') at rank k . We say $\vec{G}(F\bar{\rho} \doteq \sigma)$ is *conservative* if its reduction decreases the number of E-redexes at rank k . Specifically, if there is a λ -compatible pair (τ_1, τ'_1) such that $(\tau, \tau') \xrightarrow{\exists} (\tau_1, \tau'_1)$ by reducing $\vec{G}F\bar{\rho} \doteq \vec{G}\sigma$, then $|\text{E-redexes}((\tau, \tau'), k)| > |\text{E-redexes}((\tau_1, \tau'_1), k)|$. \square

Lemma 8.9 (Conservative E-Redexes Exist). Let (τ, τ') be a λ -compatible pair and fix integer $k \geq 0$. If $\text{E-redexes}((\tau, \tau'), k) \neq \emptyset$, then there is a constraint in $\text{E-redexes}((\tau, \tau'), k)$ which is conservative. \square

Proof. If there is exactly one constraint $\vec{G}(F\bar{\rho} \doteq \sigma)$ in $\text{E-redexes}((\tau, \tau'), k)$, then the lemma conclusion is readily verified: The reduction of $\vec{G}(F\bar{\rho} \doteq \sigma)$ does not create E-redexes at rank k , although it may create E-redexes at ranks $\neq k$. If $\text{E-redexes}((\tau, \tau'), k)$ contains $n \geq 2$ constraints, say

$$\vec{G}_1(F_1\bar{\rho}_1 \doteq \sigma_1), \vec{G}_2(F_2\bar{\rho}_2 \doteq \sigma_2), \dots, \vec{G}_n(F_n\bar{\rho}_n \doteq \sigma_n),$$

then, by lemma 8.7, they can be ordered so that $F_n \notin \text{EVar}(\vec{G}_1) \cup \dots \cup \text{EVar}(\vec{G}_{n-1})$. The last constraint $\vec{G}_n(F_n\bar{\rho}_n \doteq \sigma_n)$ in this list is a conservative E-redex at rank k . \square

Lemma 8.10 (Evaluating with Rule 3 at a Fixed Rank). Let $\mathcal{R} = \{1, 2\}$ as in lemma 8.5.

Hypothesis: Let (τ_0, τ'_0) be a λ -compatible pair in \mathcal{R} -normal form, with $k = h(\tau_0, \tau'_0)$, and consider an arbitrary evaluation with the rules in \mathcal{R} (with no rank restriction) and **rule 3** restricted to conservative E-redexes at rank k , i.e.,

$$(\#) \quad (\tau_0, \tau'_0) \xrightarrow{\mathbf{a}_1} (\tau_1, \tau'_1) \xrightarrow{\mathbf{a}_2} (\tau_2, \tau'_2) \xrightarrow{\mathbf{a}_3} \cdots \xrightarrow{\mathbf{a}_i} (\tau_i, \tau'_i) \xrightarrow{\mathbf{a}_{i+1}} \cdots$$

where for every $i \geq 1$, either $\mathbf{a}_i \in \mathcal{R}$ or $\mathbf{a}_i = \mathbf{3}$ and the step $(\tau_{i-1}, \tau'_{i-1}) \xrightarrow{\mathbf{3}} (\tau_i, \tau'_i)$ is the result of reducing a conservative E-redex at rank k .

Conclusion: The evaluation shown in (#) cannot be infinite. Moreover, if (τ_i, τ'_i) cannot be evaluated further, i.e., if (τ_i, τ'_i) is in \mathcal{R} -normal form and in $(\mathbf{3}, k)$ -normal form (see definition 8.4), then either $\tau_i = \tau'_i$ or $h(\tau_i, \tau'_i) > k$. \square

Proof. We assume E-redexes $((\tau, \tau'), k) \neq \emptyset$, otherwise the conclusion follows immediately from lemma 8.5. For arbitrary λ -compatible pair (τ, τ') , define the measure:

$$P(\tau, \tau') = |\text{E-redexes}((\tau, \tau'), h(\tau, \tau'))| = \text{the number of E-redexes of } (\tau, \tau') \text{ at rank } h(\tau, \tau').$$

We also define the measure $N(\tau, \tau')$ as follows:

$$N(\tau, \tau') = \left(P(\tau, \tau'), \text{size}(\Delta(\tau, \tau')) \right).$$

The $\text{size}(\)$ function is given in definition 5.13. We use the lexicographic ordering on pairs of natural numbers and, relative to this ordering, we show that $N(\tau, \tau')$ is strictly decreasing – provided also that the evaluation starts from a λ -compatible pair (τ_0, τ'_0) in \mathcal{R} -normal form.

If **rule 1** or **rule 2** is used, then $P(\tau, \tau')$ does not change, but $\text{size}(\Delta(\tau, \tau'))$ decreases by at least 2. In general, **rule 1** and **rule 2** can introduce new E-redexes, but these are necessarily at ranks strictly greater than $h(\tau, \tau')$, because the evaluation under consideration starts at (τ_0, τ'_0) in \mathcal{R} -normal form.

If **rule 3** is used, then $P(\tau, \tau')$ decreases by lemma 8.9, while in general $\text{size}(\Delta(\tau, \tau'))$ increases. Because we restrict **rule 3** to conservative E-redexes at rank $h(\tau, \tau')$, all new E-redexes are at ranks $> h(\tau, \tau')$.

It follows that the measure $N(\tau, \tau')$ is well-ordered, of order type ω^2 . This implies the conclusion of the lemma. \square

Definition 8.11 (Rank-Increasing Evaluations). Let $\mathcal{R} = \{\mathbf{1}, \mathbf{2}\}$. Let (τ_0, τ'_0) be a λ -compatible pair. A *rank-increasing* evaluation of (τ_0, τ'_0) is of the form:

$$(\tau_0, \tau'_0) \xrightarrow{\mathcal{R}}^* (\tau_1, \tau'_1) \xrightarrow{\mathcal{R}_1}^* (\tau_2, \tau'_2) \xrightarrow{\mathcal{R}_2}^* \cdots$$

where (τ_1, τ'_1) is in \mathcal{R} -normal form and, for every $i \geq 1$, if $\tau_i \neq \tau'_i$ then $\mathcal{R}_i = \mathcal{R} \cup \{(\mathbf{3}, k_i)\}$ where $k_i = h(\tau_i, \tau'_i)$ and $(\tau_{i+1}, \tau'_{i+1})$ is in \mathcal{R}_i -normal form. \square

Lemma 8.12 (Rank-Increasing Evaluations Complete). Let Δ be a λ -compatible constraint set, and let $(\tau, \tau') = \ulcorner \Delta \urcorner$. If a rank-increasing evaluation of (τ, τ') diverges, then Δ has no solution. \square

Proof. A rank-increasing evaluation of (τ, τ') induces an evaluation of $\text{Unify}(\Delta)$ such that, if the evaluation of (τ, τ') diverges, then the induced evaluation of $\text{Unify}(\Delta)$ also diverges. By lemma 5.11, if Δ has a solution, then every evaluation of $\text{Unify}(\Delta)$ terminates. This implies the desired result. \square

Theorem 8.13 (Decidability of Finite-Rank Beta-Unification). Let Δ be a λ -compatible constraint set and let k be a fixed but otherwise arbitrary integer ≥ 1 .

1. Δ has a solution of rank $\leq k$ if and only if there is a successful evaluation of $\text{UnifyFR}(\Delta, k)$.
2. There is an evaluation of $\text{UnifyFR}(\Delta, k)$ which terminates.
3. It is decidable whether Δ has a solution of rank $\leq k$. \square

We purposely impose the restriction $k \neq 0$ because, if Δ has more than one constraint and $\ulcorner \Delta \urcorner = (\tau, \tau')$, then $\text{rank}\{\tau, \tau'\} \geq 1$. In such a case, an evaluation of $\text{UnifyFR}(\Delta, 0)$ always fails, even if Δ has a rank-0 solution.

We conjecture part 2 of the theorem can be strengthened to: “Every evaluation of $\text{UnifyFR}(\Delta, k)$ terminates”. The conjecture will be settled if one proves that every evaluation of a λ -compatible pair (τ_0, τ'_0) that diverges is rank-increasing; lemma 8.10 proves it for only a particular evaluation strategy.

Proof. For the left-to-right implication in part 1, suppose Δ has a solution. By theorem 5.16, $\text{Unify}(\Delta) \xRightarrow{*} \mathbf{S}$ for some substitution \mathbf{S} , which is also a principal solution for Δ . The evaluation $\text{Unify}(\Delta) \xRightarrow{*} \mathbf{S}$ induces an evaluation $\text{UnifyFR}(\Delta, \ell) \xRightarrow{*} \mathbf{S}$ for some $\ell \geq 0$. Let ℓ be the least integer such that $\text{UnifyFR}(\Delta, \ell) \xRightarrow{*} \mathbf{S}$. Suppose \mathbf{S} is a rank- m solution for Δ , with $m \geq 0$. Because \mathbf{S} is a principal solution for Δ , every other solution is of rank $\geq m$. If $m \geq 1$, it is easy to see that $\ell = m$, which also implies that $\text{UnifyFR}(\Delta, k) \xRightarrow{*} \mathbf{S}$ for every $k \geq m$. (If $m = 0$ and Δ has more than one constraint, then $\ell = 1$.)

For the right-to-left implication in part 1, suppose there is a successful evaluation $\text{UnifyFR}(\Delta, k) \xRightarrow{*} \mathbf{S}$ for some $k \geq 0$, and suppose k is the least integer with this property. (We do not need to impose the restriction $k \neq 0$ for this implication.) The evaluation $\text{UnifyFR}(\Delta, k) \xRightarrow{*} \mathbf{S}$ induces an evaluation $\text{Unify}(\Delta) \xRightarrow{*} \mathbf{S}$. By theorem 5.16, \mathbf{S} is a principal solution for Δ . Moreover, if $k = 0$ or $k \geq 2$, then \mathbf{S} is a rank- k solution; and if $k = 1$, then \mathbf{S} is of rank $\leq k$.

We prove parts 2 and 3 of the theorem simultaneously. Let $(\tau_0, \tau'_0) = \lceil \Delta \rceil$ and consider a rank-increasing evaluation of (τ_0, τ'_0) as specified in definition 8.11:

$$(\tau_0, \tau'_0) \xrightarrow[\mathcal{R}]{} (\tau_1, \tau'_1) \xrightarrow[\mathcal{R}_1]{} (\tau_2, \tau'_2) \xrightarrow[\mathcal{R}_2]{} \dots$$

Such an evaluation of (τ_0, τ'_0) always exists by lemma 8.10 and, depending on whether it diverges or not, there are two cases. In the first case, if the evaluation diverges, Δ has no solution by lemma 8.12 and, moreover, there is $n \geq 1$ such that $k_n > k$ where k_n is given in definition 8.11. In this case, the induced evaluation of $\text{UnifyFR}(\Delta, k)$ terminates unsuccessfully.

In the second case, the evaluation of (τ_0, τ'_0) exhibited above terminates at (τ_n, τ'_n) with $\tau_n = \tau'_n$. There are two subcases, depending on whether $k_n > k$ or not. In the first subcase, $k_n > k$, it is clear that the induced evaluation of $\text{UnifyFR}(\Delta, k)$ terminates unsuccessfully, corresponding to the fact that every solution for Δ has rank $\geq k_n > k$. In the second subcase, $k_n \leq k$, the induced evaluation of $\text{UnifyFR}(\Delta, k)$ terminates successfully, corresponding to the fact that there is a rank- k_n principal solution for Δ . \square

Corollary 8.14 (Decidability of Finite-Rank Typability). *Let M be a λ -term and let $k \geq 0$. It is decidable whether there is a rank- k typing for M in system \mathbb{I} .* \square

Proof. For $k \leq 1$, there is a typing for M of rank ≤ 1 iff M is simply-typable. For $k \geq 2$, the result follows from theorems 7.9 and 8.13. \square

References

- [AC98] R. Amadio and P.-L. Curien. *Domains and Lambda Calculi*, vol. 46 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 1998.
- [Ban97] A. Banerjee. A modular, polyvariant, and type-based closure analysis. In *Proc. 1997 Int'l Conf. Functional Programming*. ACM Press, 1997.
- [CDC80] M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ -calculus. *Notre Dame J. Formal Logic*, 21(4):685–693, 1980.
- [CDCV80] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Principal type schemes and λ -calculus semantics. In Hindley and Seldin [HS80], pp. 535–560.
- [CDCV81] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Functional characters of solvable terms. *Z. Math. Logik Grundlag. Math.*, 27(1):45–58, 1981.
- [CG92] M. Coppo and P. Giannini. A complete type inference algorithm for simple intersection types. In *17th Colloq. Trees in Algebra and Programming*, vol. 581 of *LNCS*, pp. 102–123. Springer-Verlag, 1992.
- [DM82] L. Damas and R. Milner. Principal type schemes for functional programs. In *Conf. Rec. 9th Ann. ACM Symp. Princ. of Prog. Langs.*, pp. 207–212, 1982.

- [GJS96] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison Wesley, 1996.
- [HS80] J. R. Hindley and J. P. Seldin, eds. *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Academic Press, 1980.
- [Jen98] T. Jensen. Inference of polymorphic and conditional strictness properties. In *Conf. Rec. POPL '98: 25th ACM Symp. Princ. of Prog. Langs.*, 1998.
- [Jim96] T. Jim. What are principal typings and what are they good for? In *Conf. Rec. POPL '96: 23rd ACM Symp. Princ. of Prog. Langs.*, 1996.
- [JMZ92] B. Jacobs, I. Margaria, and M. Zacchi. Filter models with polymorphic types. *Theoret. Comput. Sci.*, 95:143–158, 1992.
- [Kfo96] A. J. Kfoury. Beta-reduction as unification. A refereed extensively edited version is [Kfo99]. This preliminary version was presented at the Helena Rasiowa Memorial Conference, July 1996.
- [Kfo99] A. J. Kfoury. Beta-reduction as unification. In D. Niwinski, ed., *Logic, Algebra, and Computer Science (H. Rasiowa Memorial Conference, December 1996)*, Banach Center Publication, Volume 46, pp. 137–158. Springer-Verlag, 1999. Supersedes [Kfo96] but omits a few proofs included in the latter.
- [KW94] A. J. Kfoury and J. B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order λ -calculus. In *Proc. 1994 ACM Conf. LISP Funct. Program.*, pp. 196–207, 1994.
- [KW99] A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In *Conf. Rec. POPL '99: 26th ACM Symp. Princ. of Prog. Langs.*, pp. 161–174, 1999. Superseded by [KW02].
- [KW02] A. J. Kfoury and J. B. Wells. Principality and type inference for intersection types using expansion variables. Supersedes [KW99] and contains proofs omitted in [KW0X], Aug. 2002.
- [KW0X] A. J. Kfoury and J. B. Wells. Principality and type inference for intersection types using expansion variables. *Theoret. Comput. Sci.*, 200X. Supersedes [KW99], omitted proofs included in the full report [KW02]. Accepted subject to revisions.
- [Lei83] D. Leivant. Polymorphic type inference. In *Conf. Rec. 10th Ann. ACM Symp. Princ. of Prog. Langs.*, pp. 88–98, 1983.
- [MTHM97] R. Milner, M. Tofte, R. Harper, and D. B. MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.
- [Pie94] B. Pierce. Bounded quantification is undecidable. *Inform. & Comput.*, 112:131–165, 1994.
- [PJHH⁺93] S. L. Peyton Jones, C. Hall, K. Hammond, W. Partain, and P. Wadler. The Glasgow Haskell compiler: A technical overview. In *Proc. UK Joint Framework for Information Technology (JFIT) Technical Conf.*, 1993.
- [Pot80] G. Pottinger. A type assignment for the strongly normalizable λ -terms. In Hindley and Seldin [HS80], pp. 561–577.
- [RDR88] S. Ronchi Della Rocca. Principal type schemes and unification for intersection type discipline. *Theoret. Comput. Sci.*, 59(1–2):181–209, Mar. 1988.
- [RDRV84] S. Ronchi Della Rocca and B. Venneri. Principal type schemes for an extended type theory. *Theoret. Comput. Sci.*, 28(1–2):151–169, Jan. 1984.
- [Sal78] P. Sallé. Une extension de la théorie des types en λ -calcul. In G. Ausiello and C. Böhm, eds., *Fifth International Conference on Automata, Languages and Programming*, vol. 62 of *LNCS*, pp. 398–410. Springer-Verlag, July 1978.

- [SM96] É. Sayag and M. Mauny. A new presentation of the intersection type discipline through principal typings of normal forms. Technical Report RR-2998, INRIA, Oct. 16, 1996.
- [SM97] É. Sayag and M. Mauny. Structural properties of intersection types. In *Proceedings of the 8th International Conference on Logic and Computer Science – Theoretical Foundations of Computing (LIRA)*, pp. 167–175, Novi Sad, Yugoslavia, Sept. 1997.
- [Urz97] P. Urzyczyn. Type reconstruction in \mathbf{F}_ω . *Math. Structures Comput. Sci.*, 7(4):329–358, 1997.
- [vB93] S. J. van Bakel. *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems*. Ph.D. thesis, Catholic University of Nijmegen, 1993.
- [Wel94] J. B. Wells. Typability and type checking in the second-order λ -calculus are equivalent and undecidable. In *Proc. 9th Ann. IEEE Symp. Logic in Comput. Sci.*, pp. 176–185, 1994. Superseded by [Wel99].
- [Wel96] J. B. Wells. Typability is undecidable for $\mathbf{F}+\text{eta}$. Tech. Rep. 96-022, Comp. Sci. Dept., Boston Univ., Mar. 1996.
- [Wel99] J. B. Wells. Typability and type checking in System F are equivalent and undecidable. *Ann. Pure Appl. Logic*, 98(1–3):111–156, 1999. Supersedes [Wel94].
- [Wel02] J. B. Wells. The essence of principal typings. In *Proc. 29th Int’l Coll. Automata, Languages, and Programming*, vol. 2380 of *LNCS*, pp. 913–925. Springer-Verlag, 2002.

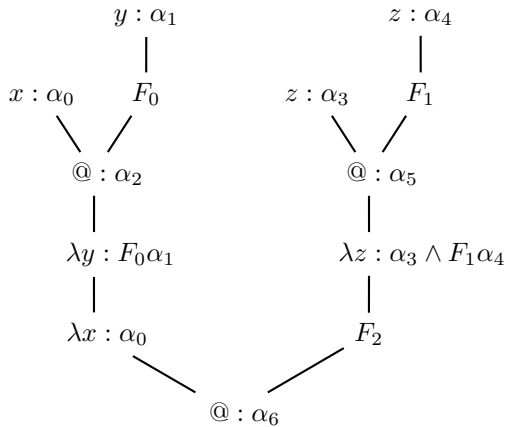
A Complete Run of the Type-Inference Procedure

We revisit example 2.22, illustrating a run of the type-inference procedure in section 6. The λ -term in this example is $(\lambda x.\lambda y.xy)(\lambda z.zz)$. The steps presented below are obtained by running an actual implementation of the procedure, which is found at <http://www.church-project.org/modular/compositional-analysis/>. (This is only one of several implementations available at the website, which are all developed in the context of the compositional-analysis effort of the Church Project.) There are non-essential differences between the initial and final skeletons below and those in example 2.22 for the same λ -expression; we follow the organization in the “type-inference report” produced by the implementation at the forementioned website.

1. Initial typing judgement:

$$\vdash (\lambda x.\lambda y.xy)(\lambda z.zz) : \alpha_6$$

2. Initial skeleton:



3. Initial constraint set:

$$\{ \begin{array}{l} \alpha_0 \doteq F_0 \alpha_1 \rightarrow \alpha_2, \\ F_2 \alpha_3 \doteq F_2 (F_1 \alpha_4 \rightarrow \alpha_5), \\ \alpha_0 \rightarrow F_0 \alpha_1 \rightarrow \alpha_2 \doteq F_2 (\alpha_3 \wedge F_1 \alpha_4 \rightarrow \alpha_5) \rightarrow \alpha_6 \end{array} \}$$

4. Final substitution, obtained by running algorithm `Unify` in section 5 on the initial constraint set:

$$\{ \{ \begin{array}{l} \alpha_0 := (F_1 \alpha_4 \rightarrow \alpha_2) \wedge F_1 \alpha_4 \rightarrow \alpha_2, \\ \alpha_1^0 := F_1 \alpha_4 \rightarrow \alpha_2, \\ \alpha_1^1 := \alpha_4, \\ \alpha_3 := F_1 \alpha_4 \rightarrow \alpha_2, \\ \alpha_5 := \alpha_2, \\ \alpha_6 := (F_1 \alpha_4 \rightarrow \alpha_2) \wedge F_1 \alpha_4 \rightarrow \alpha_2, \\ F_0 := \square \wedge F_1 \square, \\ F_2 := \square \end{array} \} \}$$

5. Final skeleton, also a derivation, obtained by applying the final substitution to the initial skeleton:

