Pattern Unification for the Lambda Calculus with Linear and Affine Types

Anders Schack-Nielsen, Carsten Schürmann
IT University of Copenhagen
Copenhagen, Denmark
anderssn|carsten@itu.dk

We define the pattern fragment for higher-order unification problems in linear and affine type theory and give a deterministic unification algorithm that computes most general unifiers.

1 Introduction

Logic programming languages, type inference algorithms, and automated theorem provers are all examples of systems that rely on unification. If the unification problem has to deal with logic variables at higher type (functional type), we speak of higher-order unification [4]. Higher-order unification is in general undecidable, but it can be turned decidable, if appropriately restricted to a fragment. For example, Miller’s pattern fragment characterizes a first-order fragment, for which unification is decidable [5].

As substructural type theories are becoming more prevalent, for example, in systems that need to represent consumable resources, higher-order unification algorithms need to deal with logic variables at linear or affine type. Linear and affine type theories, for example, refine intuitionistic type theory in the following way: Besides intuitionistic assumptions, which can be referred to an arbitrary number of times, linear and affine assumptions are treated as resources that must be referred to exactly once and at most once, respectively.

As substructural type theories are mere refinements, one might erroneously suspect that the standard intuitionistic pattern unification algorithm can be applied to this setting directly. This, unfortunately, is not the case. Consider the following two linear unification problems, where we write, as usual, \( \hat{\cdot} \) for linear application and juxtaposition for intuitionistic application.

\[
F\hat{x} = c\hat{(H_1 x)}\hat{(H_2 x)} \tag{1}
\]

\[
F\hat{x} = c\hat{(H x)} \tag{2}
\]

These examples take place in a context in which \( x \) is an intuitionistic variable. However, the linear application on the left-hand side implies that the variable must occur exactly once in any valid instantiation of \( F \), but in (1) we cannot know whether \( x \) should occur in \( H_1 \) or \( H_2 \). This additional problem over normal intuitionistic higher-order unification is caused exactly by the interaction of linear and intuitionistic variables. We solve this issue by imposing a separation of linear, affine, and intuitionistic variables.

In this paper, we refine the intuitionistic pattern fragment into a pattern fragment for linear and affine type theory. We describe a unification algorithm for this fragment and prove and prove it correct. Furthermore, we show that in this fragment most general unifiers exist. Finally, we extend the algorithm with a procedure we call linearity pruning. This procedure goes beyond the pattern fragment and treats equations such as (1) and (2) where variables may have to change their status, for example from being

\*\*This work was in part supported by NABIIT grant 2106-07-0019 of the Danish Strategic Research Council.
affine to linear. Unification problems in this extended fragment continue to be decidable. For example, for [2], the algorithm finds the most general unifier, which is \( F = \hat{\lambda} x. c^\ast (G^\ast x) \) and \( H = \hat{\lambda} x. G^\ast x \). Our focus in this paper is finding unique most general unifiers, and since (1) has a set of most general unifiers of size two, we are not going to try to solve it. However, one could easily extend linearity pruning to these cases by considering the finite number of context splits.

Previous approaches to higher-order linear unification have been restricted to highly non-deterministic algorithms, such as the preunification by Cervesato and Pfenning [1]. In contrast, our algorithm is completely deterministic, and very well suited for implementation. It is the core algorithm of the Celf proof assistant [7].

2 Language

In [8] we introduced a calculus of explicit substitutions for the \( \hat{\lambda} \)-calculus with linear, affine, and intuitionistic variables and logic variables. Along with the calculus we introduced a type system and a reduction semantics, which was proven to be type-preserving, confluent, and terminating.

| Types:       | \( A, B ::= a \mid A \& B \mid A \rightarrow B \mid A \rightarrow @ B \mid A \rightarrow B \) |
| Terms:       | \( M, N ::= 1^f \mid M[s] \mid \langle M, N \rangle \mid \text{fst} M \mid \text{snd} M \mid X[s] \)  |
|              | \( \hat{\lambda} M \mid \hat{\lambda} M \mid \lambda M \mid M^\ast N \mid M @ N \mid M N \) |
| Substitutions: | \( s, t ::= \text{id} \mid \uparrow \mid M^f . s \mid s \circ t \) |
| Linearity flags: | \( f ::= \text{I} \mid \text{A} \mid \text{L} \) |
| Contexts:    | \( \Gamma ::= \cdot \mid \Gamma, A^f \) |
| Context linearity flags: | \( l ::= f \mid U_L \mid U_A \) |

We tag each variable \( 1^f \) with a flag signifying whether the variable is intuitionistic, affine, or linear. We use \( \uparrow^n \) where \( n \geq 0 \) as a short-hand for \( n \) compositions of shift, i.e. \( \uparrow (\uparrow \circ \ldots \circ (\uparrow \circ \ldots)) \), where \( \uparrow^0 \) means id. Additionally, de Bruijn indices \( n^f \) with \( n > 1 \) are short-hand for \( 1^f [\uparrow^{n-1}] \). The context linearity flags and the corresponding assumptions in contexts are denoted intuitionistic (I), affine (A), used affine (U_A), linear (L), and used linear (U_L).

In this paper we will work exclusively with the corresponding calculus of canonical forms and hereditary substitutions. This can be obtained simply by viewing each term as a short-hand for its unique normal form and assuming that everything is fully \( \eta \)-expanded. The resulting type system is shown in Figures [1][3]. We write \( \Gamma \vdash M : A \) as a shorthand for either \( \Gamma \vdash M \leftrightarrow A \) or \( \Gamma \vdash M \Rightarrow A \).
The intuitionistic part of a context $\Gamma$ is formed by rendering all linear and affine variables unavailable, which corresponds to updating the context linearity flags from $L$ to $U_L$ and $A$ to $U_A$. Similarly, the largest context that can split to a given context is denoted $\Gamma$ and constructed by changing every $U_L$ to $L$ and $U_A$ to $A$. The predicate $\text{noin}(\Gamma)$ specifies that no linear assumptions occur in $\Gamma$, i.e. no flag in $\Gamma$ is equal to $L$. The relaxed typing judgment $\Gamma, M : A$ is similar to $\Gamma \vdash M : A$ except that it makes all variables available everywhere disregarding linearity and affineness. The typing judgments could be augmented with an additional kind of context for looking up logic variables, but we will keep this lookup implicit and simply write $\Gamma_X$ and $A_X$ for the context and type of a logic variable $X$.

Restricting ourselves to canonical forms while retaining the syntax of redices and closures as short-hands for their corresponding normal forms induces equalities corresponding to the rewrite rules of the original system. The induced equalities are shown in Figure 4. Additionally, the two typing rules for $M[s]$ and $s_1 \circ s_2$ from [8], which are left out, are now simply admissible rules proving type preservation of hereditary substitution:

\[
\begin{align*}
\Gamma \vdash s : \Gamma' & \quad \Gamma' \vdash M : A \\
\Gamma \vdash s \Gamma' M : A & \\
\Gamma \vdash M[s] : A & \\
\end{align*}
\]
The term extension hereditary substitutions.

Our pattern fragment for the linear and affine calculus that we are going to introduce next continues to guarantee this important property.

3 Patterns

We use spine notation as a convenient short-hand for series of applications and projections:

\[ S ::= () \mid M \cdot S \mid M; S \mid M \cdot N; S \mid \text{fst}; S \mid \text{snd}; S \]

The term \( M \cdot S \) is short-hand for the term where all the terms and projections in \( S \) are applied to \( M \) as follows:

\[
\begin{align*}
& M \cdot () = M \\
& M \cdot (N; S) = (M \cdot N) \cdot S \\
& M \cdot (M; S) = M \cdot (M \cdot S) \\
& M \cdot (\text{fst}; S) = (\text{fst} M) \cdot S \\
& M \cdot (\text{snd}; S) = (\text{snd} M) \cdot S \\
\end{align*}
\]

We write \( S[s] \) for the argumentwise application of \( s \) in \( S \) and observe that \( (M \cdot S)[s] = M[s] \cdot S[s] \).

We write \([X \leftarrow N]M\) for the instantiation of the logic variable \( X \) with term \( N \) in term \( M \). This instantiation is type preserving, which follows by induction on \( M \) and the subject reduction property of hereditary substitutions.

**Theorem 2.1.** If \( \Gamma X \vdash N : A_X \text{ and } \Gamma \vdash M : A \) then \( \Gamma \vdash [X \leftarrow N]M : A \).

**Theorem 2.1** is also called the contextual modal cut admissibility theorem for linear and affine contextual modal logic.

3 Patterns

The hallmark characteristic of the intuitionistic pattern fragment is the invertibility of substitutions. Our pattern fragment for the linear and affine calculus that we are going to introduce next continues to guarantee this important property.

Consider a substitution \( \Gamma \vdash a^{f_1}_j \ldots a^{f_n}_{j'} : \Gamma' \). Assume that \( a_j \) is a variable \( n^{f_j}_j \). We say the substitution extension \( n^{f_j}_{j'} \) is linear if \( f_j f_{j'} = \text{LL} \), it is affine if \( f_j f_{j'} = \text{AA} \), it is intuitionistic if \( f_j f_{j'} = \text{II} \), and it is linear-changing if \( f_j f_{j'} = \text{IL} \), \( f_j f_{j'} = \text{IA} \), or \( f_j f_{j'} = \text{AL} \). Notice that the possibilities \( \text{IL}, \text{AI}, \text{ID} \) cannot occur in well-typed substitutions since this would imply referencing a linear or affine assumption in an intuitionistic context or a linear assumption in an affine context.
Definition 3.1. A substitution \( \Gamma \vdash a_{i_1}^{f_1} \ldots a_{i_p}^{f_p} : \Gamma' \) is said to be a pattern substitution if all the terms \( a_j \) for \( j \in \{1, \ldots, p\} \) are distinct de Bruijn indices and none of them are linear-changing extensions in the substitution. A pattern substitution is called a weakening substitution if the indices \( a_j \) form an increasing sequence.

Note that in a pattern substitution all de Bruijn indices are less than or equal to \( n \) since \( n \) is equal to the length of \( \Gamma \). To understand pattern substitutions in the presence of logic variables during lowering (discussed in Section 4.1), we define the extension of pattern substitution \( s \) by spine \( S \), written as \( S.s \):

\[
\begin{align*}
(().s &= s \\
(N;S).s &= S.(N^{N^A}.s) \\
(fst;S).s &= S.s \\
(N;S).s &= S.(N^{N^L}.s) \\
(N;S).s &= S.(N^{N^L}.s)
\end{align*}
\]

Definition 3.2. A term \( M \) is said to be a pattern or within the pattern fragment if all occurrences of logic variables \( X[s] \cdot S \) satisfy the property that the substitution \( S.s \) is a pattern substitution.

Recall example \( 1 \) from the introduction. In our system, the equation is written as \( F[\uparrow^1] \cdot (1^1 \vdash (\cdot)) \equiv c \cdot (H \cdot \uparrow^1) \cdot (1^1 \vdash 1) \cdot (1^1 \vdash (\cdot)) \). We observe that it is not a pattern since there is a linear-changing substitution extension on the left-hand side in \( (1^1 \vdash (\cdot)), \uparrow^1 = 1^{I^L} \).

It can be proven that the pattern fragment is stable under hereditary substitution, logic variable instantiation, and inversion of substitutions. In particular, the following two theorems hold:

Theorem 3.3. The pattern fragment is stable under logic variable instantiation. I.e. for any patterns \( M \) and \( N \), \( X \leftarrow N \cdot M \) is a pattern.

Theorem 3.4. If \( s \) is a pattern substitution and \( M[s] \) is a pattern then \( M \) is a pattern.

The proofs are relatively straight-forward extensions of the proofs given in [3] for the intuitionistic pattern fragment.

Next, we define the inverse of a pattern substitution. The name is justified by Theorem [5.7] below.

Definition 3.5. Let \( s = a_{i_1}^{f_1} \ldots a_{i_p}^{f_p} : \uparrow^p \) be a pattern substitution. We define its inverse to be \( s^{-1} = e_{i_1}^{\epsilon_{i_1}} \ldots e_{i_p}^{\epsilon_{i_p}} : \uparrow^p \) where \( e_{i_j}^{\epsilon_{i_j}} = i^j \cdot \alpha_{i_j} \) when \( a_i = j^j \) and \( e_j \) is undefined otherwise. The undefined extensions \( e_{i_j}^{\epsilon_{i_j}} \) are flagged intuitionistic, affine, or linear depending on the \( j \)th assumption in the codomain of \( s \).

Intuitively, this definition is well defined: the \( a_i \)s are distinct and less than or equal to \( n \). For the undefined \( e_j \) one can think of an arbitrary term of the right type, e.g. a freshly created logic variable.

In the following we will refer to affine weakening on contexts \( \Gamma \vdash \Gamma' \equiv \exists \Gamma'' \cdot \Gamma = \Gamma'' \times \Gamma' \wedge \text{nolin}(\Gamma'') \)

Notice that affine weakening is reflexive and transitive, as it merely amounts to changing some number of \( \text{As} \) into \( \text{Us} \).

Lemma 3.6. For a pattern substitution \( \Gamma_2 \vdash \Gamma' \) there exists a \( \Gamma_1 \) with \( \Gamma_2 \vdash \text{aff} \Gamma_1 \) such that \( \Gamma_1 \vdash \Gamma' \) and the inverse is well-typed with \( \Gamma' \vdash s^{-1} : \Gamma_1 \).

Proof. Let \( s = a_{i_1}^{f_1} \ldots a_{i_p}^{f_p} : \uparrow^p \). Then \( \Gamma_2 = \cdot, B_{i_1}^{f_1}, \ldots, B_{i_p}^{f_p} \) and \( \Gamma' = \cdot, A_{i_1}^{f_1}, \ldots, A_{i_p}^{f_1} \). Intuitively we are going to take \( \Gamma_1 \) to be the smallest possible such that \( s \) is still well-typed, i.e. we are going to make all the affine assumptions that are not used in \( s \) unavailable. More formally we are going to set \( \Gamma_1 = \cdot, B_{i_1}^{f_1}, \ldots, B_{i_p}^{f_p} \) where \( l_1^j = l_2^j \) when \( l_1^j \in \{I, L, U_L, U_A \} \). When \( l_3^j = A \) the \( l_3^j \) will be defined below.

Consider each variable \( a_{i_j}^{f_j} = j^j \cdot \alpha_{i_j} \) in \( s \). Note that we have \( A_i = B_j \). If \( l_1^j = f \) where \( f \) is either \( I \) or \( L \) then we have \( f_j = f \) and \( l_2^j = l_3^j = f \). In the case where \( l_1^j = U_L \) then \( f_j = L \) and \( l_2^j = l_3^j \) are either equal
to $U_L$ or $L$, but since all the variables in $s$ are distinct it has to be $U_L$. If $l'_j = A$ then $f_i = A$ and $l^j = A$, and in this case we set $l^j = A$. Finally, if $l'_j = U_A$ then $f_i = A$ and $l^j$ is either $U_A$ or $A$. If $l^j = U_A$ then $l^j$ is also equal to $U_A$, and if $l^j = A$ then we can set $l^j = U_A$ since $j$ does not occur anywhere else in $s$. This means that for all defined extensions $e_j = i^j$ in $s^{-1}$ we have $l'_j = l^j$.

The remaining $B_l^j$ s for which there are no $a_i = j^j$ are all shifted away by the $\triangleright^n$ part of $s$. Therefore none of them can be linear, and if any of them are affine, i.e. have $l^j = A$, we set $l^j = U_A$. This means that all the undefined extensions in $s^{-1}$ correspond to intuitionistic, used linear, or used affine assumptions in $\Gamma_1$, and we see that $s^{-1}$ indeed is well-typed with $\Gamma \vdash s^{-1} : \Gamma_1$.

**Theorem 3.7.** Given a pattern substitution $\Gamma \vdash s : \Gamma'$, we have $\Gamma \vdash s \circ s^{-1} : \Gamma'$ and $s \circ s^{-1} = id$.

**Proof.** Let $s = a_1^{j_1} \ldots a_n^{j_n} \triangleright^n$. Since $a_i = j^j$ then the $j$th extension in $s^{-1}$ is equal to $i^j$, and thus $a_i[s^{-1}] = i^j$ for all $i$.

We have the usual definition of occurrence, rigid occurrence, and flexible occurrence written as $\in$, $\in_{rig}$, and $\in_{flex}$ respectively. These relations are only defined for canonical forms in which all logic variables are of base type (lowering will achieve this). Occurrence is defined as $\in = \in_{rig} \cup \in_{flex}$. Rigid and flexible occurrence are defined as follows, where we write $\in_*$ for either rigid or flexible occurrence.

| $n \in_{rig} s$ | $a_i = n^j$ | $n \in_\ast M$ |
|-----------------|--------------|-----------------|
| $n \in_{flex} X[s]$ | $n \in_{rig} a_1^{f_1} \ldots a_q^{f_q} \triangleright^n$ | $n \in \ast (M_1 ; M_2)$ |
| $n \in_\ast \lambda M$ | $n \in_\ast \lambda M$ | $n \in_\ast \lambda M$ |
| $n \in_\ast \lambda M$ | $n \in_\ast \lambda M$ | $n \in_\ast \lambda M$ |

If $n \in_{flex} M$ then the definition implies that there is some logic variable $X[a_1^{f_1} \ldots a_q^{f_q} \triangleright^n]$ in $M$ beneath $k$ lambdas such that $(n + k)^{f_i} = a_i$. In this case we say that $n$ occurs in the $k$th argument of $X$.

**Lemma 3.8.** Linearity implies occurrence.

1. Let $\Gamma \vdash s : \Gamma'$ be a pattern substitution and the $n$th assumption in $\Gamma$ be linear. Then $n$ occurs in $s$.
2. Let $\Gamma \vdash M : A$ be a pattern and let the $n$th assumption in $\Gamma$ be linear. Then $n$ occurs in $M$.

**Proof.** If $s = a_1^{f_1} \ldots a_q^{f_q} \triangleright^n$ then we must have $n = a_i$ for some $a_i$ since a linear assumption cannot be shifted away. The second case is by induction on $M$.

**Definition 3.9.** Given the typing of a substitution $\Gamma \vdash s : \Gamma'$ we will call it strong if there exists no $\Gamma'' \not= \Gamma'$ such that $\Gamma'' \models_{aff} \Gamma'$ and $\Gamma \vdash s : \Gamma''$.

For a pattern substitution $\cdot, B_h^a, \ldots, B_1^{a_1}, \ldots, a_1^{f_1} \ldots a_q^{f_q} \triangleright^n \cdot, A_1^{f_1}, \ldots, A_1^{f_q}$ we see that it is strong if and only if for each affine variable $a_i = j^j$ we have $l'_j = U_A$ implies $l_j = U_A$.

Consider the split of a strong pattern substitution $\Gamma \vdash s : \Gamma'$ over a context split $\Gamma' = \Gamma'_1 \bowtie \Gamma'_2$ into $\Gamma_1 \vdash s : \Gamma'_1$ and $\Gamma_2 \vdash s : \Gamma'_2$ with $\Gamma = \Gamma_1 \bowtie \Gamma_2$. For any used affine assumption in $\Gamma'_1$ the assumption is either affine or used affine in $\Gamma'$ and $\Gamma'_2$. If it is used affine then the corresponding assumption is also used affine in $\Gamma$ and thereby $\Gamma_1$. If it is affine then the corresponding assumption has to be affine in $\Gamma_2$ and is thereby used affine in $\Gamma_1$. This means that $\Gamma_1 \vdash s : \Gamma'_1$ is strong and by symmetry so is $\Gamma_2 \vdash s : \Gamma'_2$.

**Theorem 3.10.** Let $\Gamma \vdash s : \Gamma'$ be a pattern substitution and $\Gamma \vdash M : A$ be a term in which all logic variables are of base type.
1. If there exists a term $\Gamma' \vdash M' : A$ such that $M = M'[s]$ then every variable occurring in $M$ also occurs in $s$.

2. If the typing $\Gamma \vdash s : \Gamma'$ is strong and every variable occurring in $M$ also occurs in $s$ then there exists a term $\Gamma' \vdash M' : A$ such that $M = M'[s]$.

Proof. 1. follows by induction on $M'$ and 2. by induction on $M$ using the fact that context splits preserve a strong typing of $s$. It is easy to see that a strong typing of $s$ implies a strong typing of $1f.(s \circ \uparrow)$ when going beneath a lambda-binder.

For the base case $M = n^f$ we get that $n \in s$ implies that the $n$th assumption in $\Gamma$ corresponds to an assumption, say the $m$th, in $\Gamma'$. Now, we can take $M' = m^f$, and since $s$ is strong, availability of the $n$th assumption in $\Gamma$ implies availability of the $m$th assumption in $\Gamma'$ and thus that $M'$ is well-typed. The base case $M = X[t]$ is similar, when noting that the shift at the end of $s$ is equal to the shift at the end of $t$, since they are both equal to the length of $\Gamma$.

Theorem 3.10 states that occurrence is a conservative approximation of the set of variables occurring in any instantiation of a term, i.e. if $n \in [X \leftarrow N]M$ then $n \in M$. The opposite is not necessarily true.

4 Pattern unification

A unification problem $P$ is a conjunction of unification equations, and a solution to a unification problem is an instantiation of the logic variables such that all equations are satisfied. Such a collection of logic variable instantiations will be written as $\theta$ and we say that $\theta$ solves $P$. In this section we describe an algorithm that returns “no” if no such solution exists or a most general unifier otherwise, i.e. a solution that all other solutions are refinements of.

More formally, we write $\Gamma \vdash M_1 \doteq M_2 : A$ for a unification equation or simply $M_1 \doteq M_2$ with the implicit understanding that both terms have the same type in the same context. Unification equations are symmetric and we will implicitly switch from $M_1 \doteq M_2$ to $M_2 \doteq M_1$ when needed. Unification problems are given by the following grammar, where $\top$ is the solved unification problem and $\bot$ is the unification problem with no solutions.

$$P ::= \top | \bot | P \land (\Gamma \vdash M_1 \doteq M_2 : A)$$

For convenience we generalize unification equations to spines and write $S_1 \doteq S_2$ as a short-hand for the argumentwise conjunction of unification equations (see below).

4.1 Unification algorithm

The unification algorithm consists of a set of transformation rules of the form $P \mapsto P'$. We will see that the repeated application of these rule to any unification problem will eventually terminate resulting in either $\bot$, which indicates that the original problem has no solution, or $\top$, which indicates that all equations have been solved and that a most general unifier has been found. In this case the most general unifier is a mapping from logic variables to their instantiations as computed during the execution of the algorithm. The unification algorithm is given in Figure 5 and each rule is explained in detail below. For convenience we write the decomposition of a term $M$ into one of its subterms $N$ and the surrounding term with a hole $M'\{\cdot\}$ as $M = M'\{N\}$.

Decomposition. Consider a unification equation $\Gamma \vdash M_1 \doteq M_2 : A$ and assume that $A$ is not a base type.
Figure 5: Pattern unification rules
If $A = B \leadsto C$ then we must have $M_1 = \bar{\lambda} M'_1$ and $M_2 = \bar{\lambda} M'_2$. In this case $M_1$ is equal to $M_2$ under some $\theta$ if and only if $M'_1$ is equal to $M'_2$ under $\theta$ and we therefore apply dec-lam-i. The other non-base type cases for $A$ are similar and give rise to dec-lam-a, dec-lam-i, and dec-pair.

If $A$ is a base type then $M_1 = H_1 \cdot S_1$ and $M_2 = H_2 \cdot S_2$ where $H_1$ and $H_2$ are either variables or logic variables. The case of logic variables is handled below. We therefore have $n^f \cdot S_1 = m^f \cdot S_2$. If $n \neq m$ then no $\theta$ can make the two equal and we can therefore apply dec-atomic-neq. If $n = m$ then the spines must unify and we apply dec-atomic-eq where $P \land S_1 \vdash S_2$ is defined as:

\[
\begin{align*}
P \land () \equiv () & \quad = P \\
(P \land (M_1; S_1) \equiv (M_2; S_2)) & \quad = P \land M_1 \equiv M_2 \land S_1 \equiv S_2 \\
(P \land (M_1; S_1) \equiv (M_2; S_2)) & \quad = P \land M_1 \equiv M_2 \land S_1 \equiv S_2 \\
(P \land (M_1; S_1) \equiv (M_2; S_2)) & \quad = P \land M_1 \equiv M_2 \land S_1 \equiv S_2 \\
(P \land (M_1; S_1) \equiv (M_2; S_2)) & \quad = P \land M_1 \equiv M_2 \land S_1 \equiv S_2
\end{align*}
\]

No other cases can occur because $n = m$ trivially imply that they have the same type.

**Lowering.** When a logic variable occurs in a unification problem in the form $X[s] \cdot S$ with a non-empty spine, we know that $A_X$ cannot be a base type. And since canonical forms of non-base type have unique head constructors, we can safely instantiate $X$ to that particular constructor. This is accomplished by the rules lower-*\footnote{Notice that this argument relies on the fact that $Y$ is under a pattern substitution and thus has no linear-changing variables.}. Therefore we can assume that all logic variables are of base type.

**Occurs check.** Consider a unification equation of the form $X[s] \equiv M$. If $X$ also occurs in the right-hand side then either $M = n^f \cdot S[X[t]]$ or $M = X[t]$. The latter case is handled below in **Intersection.** In the former case we have the equation $X[s] \equiv n^f \cdot S[X[t]]$. Since a pattern substitution $t$ applied to any term can never alter the shape of the term but only rename variables this equation has no solutions, and we can apply occurs-check.

**Pruning.** When we have $X[s] \equiv M$ then Theorem 3.10 tells us that under some $\theta$ solving the equation, variables that do not occur in $s$ cannot occur in $M$. Assume that $n \notin s$ and $n \in M$. If $n \in \text{rig} M$ then no instantiation of logic variables can get rid of the occurrence and we apply pruning-fail. If on the other hand $n \in \text{flex} M$ then the occurrence is in the $i$th argument of some logic variable $Y$. This means, however, that no instantiation of $Y$ in a solution can contain $i$. By Lemma 3.8 we know that $n$ cannot refer to a linear assumption in the context in which $X[s]$ and $M$ are typed and therefore the $i$th assumption in $\Gamma_Y$ cannot be linear\footnote{Notice that this argument relies on the fact that $Y$ is under a pattern substitution and thus has no linear-changing variables.}.

Let $w$ be the weakening substitution $\text{weaken}(\Gamma_Y; i)$ where weaken is defined as:

\[
\begin{align*}
\text{weaken}(\Gamma, A^i; 1) & \quad = \uparrow & \text{if } i \neq L \\
\text{weaken}(\Gamma, A^i; i + 1) & \quad = \text{I}^L \cdot \text{weaken}(\Gamma; i) \circ \uparrow & \text{if } i \in \{A, U_A\} \\
\text{weaken}(\Gamma, A^i; i + 1) & \quad = \text{I}^L \cdot \text{weaken}(\Gamma; i) \circ \uparrow & \text{if } i \in \{L, U_L\}
\end{align*}
\]

Define $\Gamma \downarrow i$ to be the context $\Gamma$ with the $i$th assumption removed. We see that $\Gamma \vdash \text{weaken}(\Gamma; i) : \Gamma \downarrow i$. Furthermore, this is a strong typing. Since the $i$th assumption in $\Gamma_Y$ is not linear then $w = \text{weaken}(\Gamma_Y; i)$ does indeed exist. Theorem 3.10 tells us that $Y$ has to be instantiated to something on the form $M[w]$ and we can therefore apply pruning.

**Context pruning.** If a logic variable $X$ is declared in context $\Gamma_X = \cdot, A^p, \ldots, A^n$ with $l_a \in \{U_A, U_L\}$, we know that $n$ cannot occur in a well-typed instantiation of $X$. Therefore, by Theorem 3.10 $X$ has to be instantiated to something on the form $M[\text{weaken}(\Gamma_X; n)]$ and we can therefore apply ctx-pruning.
Note that pruning the context of $X$ in this way in the case of $X[s] = M$ may allow further pruning in $M$. Additionally, repeated applications of this step will ensure that no used affine assumptions occur in the context of logic variables. Therefore all typings of the associated substitutions are strong.

**Instantiation.** Consider the unification equation $X[s] = M$ where all used affine assumptions have been pruned from $\Gamma_X$ and the typing of $s$ therefore is strong. If all $n \in M$ also occur in $s$ then Theorem 3.10 tells us that $M$ is equal to $M'[s]$ for some $M'$. By Theorem 3.7 we know that $M'$ is equal to $M[s^{-1}]$ and we can therefore instantiate $X$ by the rule instantiation provided that $X$ does not occur in $M$.

**Intersection.** The final case is when we have $X[s] = X[t]$. If $s = t$ then the equation will be trivially satisfied no matter what term $X$ might be instantiated to, so we can simply remove the equation by the rule intersection-eq.

Consider an instantiation of $X$ to some $M$. If for all $n \in M$ we have $n[s] = n[t]$ then the equation is clearly satisfied. If on the other hand there is some $n \in M$ such that $n[s] \neq n[t]$ then the two sides of the equation will not be equal. Therefore any variable $n$ for which $n[s] \neq n[t]$ cannot occur in an instantiation of $X$. If such an $n$ is linear then Lemma 3.8 tells us that $n$ has to occur in all instantiations and we can conclude that there is no solution and apply intersection-fail. Otherwise, any instantiation of $X$ has to be on the form $M'[s \cap t]$ for some $M'$ where $s \cap t$ is defined as the following weakening substitution:

$$
M^f . s \cap M^f . t = 1^f . (s \cap t) \circ \uparrow,
$$

$$
n^f . s \cap n^f . t = (s \cap t) \circ \uparrow \quad \text{if } n \neq m \text{ and } f \in \{I, A\},
$$

$$
\uparrow^n \cap \uparrow^m = \text{id}
$$

Note that $s \cap t$ exists exactly when $n[s] = n[t]$ for all linear $n$. The domain of $s \cap t$ is seen to be $\Gamma_X$ with those assumptions removed for which $n[s] \neq n[t]$. This step is summarized by the rule intersection.

### 4.2 Correctness

Correctness of the unification algorithm has three parts: preservation, progress, and termination.

**Theorem 4.1.** The unification algorithm solves all pattern unification problems correctly.

1. If $P \rightarrow P'$ then the set of solutions to $P$ is equal to the set of solutions to $P'$.
2. If $P$ has unsolved equations (i.e. $P$ is not equal to $\top$ or $\bot$) then there exists a $P'$ such that $P \rightarrow P'$.
3. The unification algorithm terminates.

**Proof.** The discussion above in section 4.1 proves preservation of solutions (1) and progress (2). For termination (3) we will consider the lexicographic ordering of

1. The total size of all types of all logic variables occurring in the unification problem.
2. The total size of all contexts of the logic variables occurring in the unification problem.
3. The total size of all terms in the unification problem.

We see that the decomposition rules $\text{dec-}$ decrease (3) while keeping (1) and (2) constant. The lowering rules $\text{lower-}$ and instantiation decrease (1). The $\text{intersection-eq}$ rule decreases (3) while keeping (1) and (2) constant. The pruning, ctx-pruning, and intersection rules decrease (2) while keeping (1) constant. \qed


Figure 6: Modified pruning rules

5 Linearity pruning

Within the pattern fragment we know that most general unifiers exist and we have a decidable algorithm for finding them. For practical applications, however, it is often necessary to relax the pattern restriction and accept that the algorithm sometimes returns left-over unification problems. Reed [6], for example, describes the dynamic intuitionistic pattern fragment that postpones any unification equation as constraints that cannot be solved immediately.

In this section we will relax the restriction of pattern substitutions from Definition 3.1 to linear-changing pattern substitutions permitting linear-changing extensions, greatly expanding the applicability of our unification algorithm. If a unification equation involving linear-changing pattern substitutions cannot be resolved, it is simply postponed as a constraint. Instead of just returning $\mathbb{F}$ or $\mathbb{F}$, the unification algorithm using linearity pruning may fail with leftover constraints.

In order to handle linear-changing extensions in substitutions we first need to revisit the notion of variable occurrence that was defined in section 3. So far, occurrences have been divided into two categories; rigid and flexible. We will need to make further distinctions into a total of 12 categories.

We say that an occurrence is in an intuitionistic position in a term if the term can be written as $M\{n^I \cdot S \cdot (N; S')\}$ such that the occurrence is within $N$. If an occurrence is not in an intuitionistic position and the term can be written as $M\{n^I \cdot S \cdot (N; S')\}$ such that the occurrence is within $N$ we say that it is in an affine position. If an occurrence is neither in an intuitionistic position nor in an affine position we say that it is in a linear position. This means that intuitionistic positions are precisely those in which top-level affine and linear assumptions are not available. Similarly, affine positions are those in which top-level affine assumptions are available but the linear are not. Finally, linear positions are those where all top-level assumptions are available.

If $n$ occurs flexibly in a term $M$, i.e. it occurs in the $i$th argument of some logic variable $X$, there are five possibilities for the $i$th assumption in $\Gamma_X$; it can be intuitionistic, affine, used affine, linear, or used linear. We say that $n$ occurs in an intuitionistic argument if the $i$th assumption in $\Gamma_X$ is intuitionistic, we say that it occurs in an affine argument if the $i$th assumption in $\Gamma_X$ is affine, and we say that it occurs in a linear argument if the $i$th assumption in $\Gamma_X$ is linear. We will write this as $n \in_{\text{flex, I}} M$, $n \in_{\text{flex, A}} M$, and $n \in_{\text{flex, L}} M$, respectively. Occurrences where the $i$th assumption in $\Gamma_X$ is either used affine or used linear are not relevant, since context pruning will have removed them (see rule $\text{ctx-pruning}$ in Figure 5).

This gives a total of 12 categories of occurrence, since any occurrence is either in an intuitionistic, affine, or linear position and it is either a rigid occurrence or a flexible occurrence in an intuitionistic, affine, or linear argument.

If we are at any time forced to prune a variable occurring in a linear argument we can simply fail, since the reason for pruning implies that the variable cannot occur in the given place but the linear typing tells us that it will. Consider the case $X[s] = M$ with $n \notin s$ and $n \in M$. Since we have widened
the fragment we are considering to include linear-changing pattern substitutions it is now possible that \( n \in \text{flex}, L \). This was previously impossible since if every substitution is a pattern then \( n \in \text{flex}, L \) implies that \( n \) is linear which in turn implies \( n \in s \). The pruning and pruning-fail rules therefore has to be modified slightly in this case as shown in Figure 6.

5.1 Linear-changing pattern substitutions

Definition 5.1. A linear-changing pattern substitution \( s \) is called a linear-changing identity substitution if it is on the form:

\[
1^n_1, 2^n_2, \ldots, n^n_n, \uparrow^n
\]

or equivalently that it is \( \eta \)-equivalent to \( \text{id} \) except for some number of linear-changing extensions.

Theorem 5.2. Linear-changing identity substitutions are injective. Given \( M, M' \), and a linear-changing identity substitution \( s \), then \( M[s] = M'[s] \) implies \( M = M' \).

Proof. The substitution \( s \) simply changes the linearity flags in \( M \) and \( M' \) from \( L \) to \( A \) or \( I \) or from \( A \) to \( I \) on those variables that are linear-changing in \( s \) and it is therefore trivially injective.

Theorem 5.3. A linear-changing pattern substitution can be decomposed into a pattern substitution and a linear-changing identity substitution. If \( s \) is a linear-changing pattern substitution then there exists a pattern substitution \( s' \) and a linear-changing identity substitution \( t \) such that \( s = s' \circ t \).

Proof. Take \( s' \) to be \( s \) with all linear-changing extensions \( \text{AL} \) and \( \text{IL} \) changed to linear extensions and all linear-changing extensions \( \text{IA} \) changed to affine extensions and \( t \) to be a linear-changing identity substitution with the corresponding linear-changing extensions.

Theorem 5.4. Let \( s \) be a linear-changing identity substitution with exactly one linear-changing extension \( n^n_1 \) and \( M \) be some term.

1. If the linear-changing extension is \( ff' = \text{IL} \) then there exists an \( M' \) such that \( M = M'[s] \) if and only if the following five properties hold:
   (a) \( n \) occurs in \( M \).
   (b) There are no occurrences of \( n \) in intuitionistic or affine positions in \( M \).
   (c) For all subterms \( \langle M_1, M_2 \rangle \) of \( M \) under \( k \) lambdas \( n + k \) occurs in \( M_1 \) if and only if it occurs in \( M_2 \).
   (d) For all subterms \( M_1 \downarrow M_2 \) of \( M \) under \( k \) lambdas \( n + k \) occurs in at most one of \( M_1 \) and \( M_2 \).
   (e) All flexible occurrences of \( n \) in \( M \) are in linear arguments.

2. If the linear-changing extension is \( ff' = \text{IA} \) then there exists an \( M' \) such that \( M = M'[s] \) if and only if the following three properties hold:
   (a) There are no occurrences of \( n \) in intuitionistic positions in \( M \).
   (b) For all subterms \( M_1 \downarrow M_2 \) and \( M_1 \uparrow M_2 \) of \( M \) under \( k \) lambdas \( n + k \) occurs in at most one of \( M_1 \) and \( M_2 \).
   (c) All flexible occurrences of \( n \) in \( M \) are in linear or affine arguments.

3. If the linear-changing extension is \( ff' = \text{AL} \) then there exists an \( M' \) such that \( M = M'[s] \) if and only if the following four properties hold:
   (a) \( n \) occurs in \( M \).
(b) There are no occurrences of \( n \) in affine positions in \( M \).

(c) For all subterms \( \langle M_1, M_2 \rangle \) of \( M \) under \( k \) lambdas \( n + k \) occurs in \( M_1 \) if and only if it occurs in \( M_2 \).

(d) All flexible occurrences of \( n \) in \( M \) are in linear arguments.

Proof. By induction on \( M \) noting that each of the three sets of properties are precisely the occurrence requirements for, respectively, linear variables, affine variables, and linear variables known to adhere to the affine occurrence requirements.

Theorem 5.4 tells us when there exists an \( M' \) such that \( M = M'[s] \) for a linear-changing identity substitution \( s \) with a single linear-changing extension. As a corollary we get the conditions when \( s \) is a general linear-changing identity substitution. The existence of \( M' \) is equivalent to the conjunction of the requirements for each linear-changing extension, since we can decompose any linear-changing identity substitution \( s \) with \( k \) linear-changing extensions into \( s = s_1 \circ s_2 \circ \cdots \circ s_k \) where each \( s_i \) is a linear-changing identity substitution with exactly one linear-changing extension.

5.2 Linearity pruning

Consider the following unification equation where \( s \) is a linear-changing pattern substitution:

\[
\Gamma \vdash X[s] = M : B
\]

We cannot invert \( s \) directly but we can decompose it by Theorem 5.3 into a pattern substitution \( s' \) and a linear-changing identity substitution \( t \) changing the problem to:

\[
\Gamma \vdash X[s'][t] = M : B
\]

In this case we perform a number of pruning steps on the right-hand side since in any solution the \( M \) must adhere to the requirements in Theorem 5.4. We will consider each linear-changing extension \( n^f f' \) in \( t \) individually. The entire algorithm is given in Figure 7 and each rule is explained below.

Since many of the rules rely on pruning, we extend our language of unification problems with the constraint \( \text{prune}(n; M) \) to simplify the presentation. This constraint states that \( n \) cannot occur in \( M \) in a solution. If this is already the case then the rule \( \text{prune-finish} \) removes it. If \( n \) occurs either rigidly or flexibly in a linear argument in \( M \) then no instantiation of logic variables can remove the occurrence, and therefore there are no solutions. The rule \( \text{prune-fail} \) covers this case. If there are flexible occurrences in either intuitionistic or affine arguments then we can safely prune them away with the rule \( \text{prune} \).

Position-based pruning. The variable \( n \) cannot occur in any intuitionistic position. Furthermore, if \( f' = L \) then \( n \) also cannot occur in affine positions. These occurrences can therefore be pruned away with the rules \( \text{int-pos} \) and \( \text{aff-pos} \).

Pruning at multiplicative context splits. We will now consider all linear applications \( M_1 \sim M_2 \) and all affine applications \( M_1 @ M_2 \) in the term \( M \) and compare occurrences in \( M_1 \) and \( M_2 \), as these positions are where the context is split multiplicatively.

For any multiplicative context split the variable should only occur in one of the branches by Theorem 5.4. A multiplicative split with rigid or linear argument occurrences in one of the branches therefore allows us to prune any occurrences in the other branch with the rule \( \text{multiplicative} \), and if this is impossible due to rigid or linear argument occurrences in both branches, we conclude that there is no solution.
\[
P \land X[s] \models M \quad \Rightarrow \quad P \land X[s] \models M \land \text{prune}(n + k; N)
\]
\text{if } n^{1/\ell} \text{ is a linear-changing extension in } s \text{ and } n \text{ occurs in an intuitionistic position in } M
\text{ in the subterm } N \text{ under } k \text{ lambdas}

\text{aff-pos}
\[
P \land X[s] \models M \quad \Rightarrow \quad P \land X[s] \models M \land \text{prune}(n + k; N)
\]
\text{if } n^{L} \text{ is a linear-changing extension in } s \text{ and } n \text{ occurs in an affine position in } M \text{ in the}
\text{subterm } N \text{ under } k \text{ lambdas}

\text{prune-fail}
\[
P \land \text{prune}(n; M) \quad \Rightarrow \quad \text{F}
\]
\text{if } n \in \text{rig } M \text{ or } n \in \text{flex } L \text{ M}

\text{prune}
\[
P \land \text{prune}(n; M) \quad \Rightarrow \quad [Y \leftarrow Z[w]](P \land \text{prune}(n; M))
\]
\text{if } n \text{ occurs in the } i^{th} \text{ argument of the logic variable } Y \text{ in } M, \text{ the argument is either}
\text{intuitionistic or affine, } w = \text{weaken}(\Gamma_Y; i), \text{ and } Z \text{ is a fresh logic variable with } A_Z = A_Y
\text{and } \Gamma_Z = \Gamma_Y \div i

\text{prune-finish}
\[
P \land \text{prune}(n; M) \quad \Rightarrow \quad P
\]
\text{if } n \notin M

\text{multiplicative}
\[
P \land X[s] \models M \quad \Rightarrow \quad P \land X[s] \models M \land \text{prune}(n + k; M_2)
\]
\text{if } n^{1/\ell} \text{ is a linear-changing extension in } s, n + k \text{ occurs either rigidly or flexibly in a linear}
\text{argument in } M_1, \text{ and } n + k \text{ occurs in } M_2, \text{ where either } M_1 \bowtie M_2, M_2 \bowtie M_1, M_1 \bowtie M_2,
\text{or } M_2 \bowtie M_1 \text{ is a subterm of } M \text{ beneath } k \text{ lambdas}

\text{additive}
\[
P \land X[s] \models M \quad \Rightarrow \quad P \land X[s] \models M \land \text{prune}(n + k; M_2)
\]
\text{if } n^{L} \text{ is a linear-changing extension in } s, n + k \notin M_1, \text{ and } n + k \in M_2, \text{ where } (M_1, M_2)
\text{or } (M_1, M_2) \text{ is a subterm of } M \text{ beneath } k \text{ lambdas}

\text{int-strengthen}
\[
P \land X[s] \models M \quad \Rightarrow \quad [Y \leftarrow Z[t]](P \land X[s] \models M)
\]
\text{if } n^{1/\ell} \text{ is a linear-changing extension in } s, n \text{ occurs flexibly in } M \text{ in the } i^{th} \text{ argument}
\text{of the logic variable } Y, \text{ the argument is intuitionistic, } t = \text{linweaken}(i; IA), \text{ and } Z \text{ is a fresh logic}
\text{variable with } A_Z = A_Y \text{ and } \Gamma_Z = \text{strengthen}(\Gamma_Y; i; IA)

\text{aff-strengthen}
\[
P \land X[s] \models M \quad \Rightarrow \quad [Y \leftarrow Z[t]](P \land X[s] \models M)
\]
\text{if } n^{L} \text{ is a linear-changing extension in } s, n \text{ occurs flexibly in } M \text{ in the } i^{th} \text{ argument}
\text{of the logic variable } Y, \text{ the argument is affine, } t = \text{linweaken}(i; AL), \text{ and } Z \text{ is a fresh logic}
\text{variable with } A_Z = A_Y \text{ and } \Gamma_Z = \text{strengthen}(\Gamma_Y; i; AL)

\text{no-occur}
\[
P \land X[s] \models M \quad \Rightarrow \quad \text{F}
\]
\text{if } n^{L} \text{ is a linear-changing extension in } s \text{ and } n \notin M

\text{int-aff-invert}
\[
P \land X[s] \models M \quad \Rightarrow \quad P \land X[s'] \models M'
\]
\text{if } n^{1/\ell} \text{ is a linear-changing extension in } s, \text{ there are no occurrences of } n \text{ in intuitionistic}
\text{positions in } M, \text{ for all subterms } M_1 \bowtie M_2 \text{ and } M_1 \bowtie M_2 \text{ of } M \text{ under } k \text{ lambdas}
\text{n + k occurs in at most one of } M_1 \text{ and } M_2, \text{ and all flexible occurrences of } n \text{ in } M \text{ are in linear or affine arguments;}
\text{s'} \text{ and } M' \text{ are given by } s = s' \circ t \text{ and } M = M'[t] \text{ where } t = \text{linweaken}(n; IA)

\text{aff-lin-invert}
\[
P \land X[s] \models M \quad \Rightarrow \quad P \land X[s'] \models M'
\]
\text{if } n^{L} \text{ is a linear-changing extension in } s, \text{ n occurs in } M, \text{ there are no occurrences of } n \text{ in affine}
\text{positions in } M, \text{ for all subterms } (M_1, M_2) \text{ of } M \text{ under } k \text{ lambdas } n + k \text{ occurs in }
\text{M_1 if and only if it occurs in } M_2, \text{ and all flexible occurrences of } n \text{ in } M \text{ are in linear arguments;}
\text{s'} \text{ and } M' \text{ are given by } s = s' \circ t \text{ and } M = M'[t] \text{ where } t = \text{linweaken}(n; AL)
by following up with prune-fail. We can restrict the multiplicative rule to the case where \( f = I \), since \( \text{ff}' = AL \) implies that \( n \) already occurs in at most one of the branches at each multiplicative split.

Pruning at additive context splits. Similarly, we consider all pairs \((M_1, M_2)\) in the term \( M \), i.e. the places where the context is split additively. If \( \text{ff}' = L \) then the variable \( n \) must occur in either both branches of the additive split or in none of them. An additive split without occurrences in one of the branches therefore allows us to prune any occurrences in the other branch using the additive rule.

Strengthening intuitionistic variables. Consider the case when \( f = I \), i.e. \( n \) is intuitionistic, and consider some flexible occurrence of \( n \) in an intuitionistic argument, say the \( i \)th, of some logic variable \( Y \) in \( M \). If \( \text{ff}' = L \) then we do not necessarily know whether this particular occurrence should be pruned away or strengthened to a linear occurrence, but in either case, and also if \( \text{ff}' = A \), we can safely strengthen the \( i \)th assumption of \( Y \) from intuitionistic to affine. Let \( t = \text{linweaken}(\Gamma_Y; i; IA) \) and \( Z \) be a fresh logic variable with \( A_Z = A_Y \) and \( \Gamma_Z = \text{strengthen}(\Gamma_Y; i; IA) \), where linweaken and strengthen are defined as follows:

\[
\begin{align*}
\text{linweaken}(\Gamma, A^f; i + 1; \text{ff}') & = 1f' \\
\text{linweaken}(\Gamma, A^I; i + 1; \text{ff}') & = 1L.\text{linweaken}(\Gamma; i; \text{ff}') \circ \uparrow \\
\text{linweaken}(\Gamma, A^A; i + 1; \text{ff}') & = 1A.\text{linweaken}(\Gamma; i; \text{ff}') \circ \uparrow \text{ if } i \in \{A, U_A\} \\
\text{linweaken}(\Gamma, A^L; i + 1; \text{ff}') & = 1L.\text{linweaken}(\Gamma; i; \text{ff}') \circ \uparrow \text{ if } i \in \{L, U_L\} \\
\text{strengthen}(\Gamma, A^f; i; \text{ff}') & = \Gamma, A^f \\
\text{strengthen}(\Gamma, A^A; i + 1; \text{ff}') & = \text{strengthen}(\Gamma; i; \text{ff}')A^I
\end{align*}
\]

Note that \( \Gamma \vdash \text{linweaken}(\Gamma; i; \text{ff}') : \text{strengthen}(\Gamma; i; \text{ff}') \) when the \( i \)th assumption in \( \Gamma \) is \( A^f \) and \( \text{ff}' \) is either \( IA, IL, \) or \( AL \). When referring to linweaken we will sometimes leave out the context and simply write \( \text{linweaken}(i; \text{ff}') \) as \( \Gamma \) can be inferred from the codomain of the substitution.

We can now instantiate \( Y \) to \( Z[t] \) as shown in the int-strengthen rule. When we cannot apply this rule anymore, and we furthermore cannot apply any of the pruning steps described above, then either \( M \) satisfies the three conditions of part 2 of Theorem 5.4 or else there is some subterm \( M_1 \bowtie M_2 \) or \( M_1 \odot M_2 \) with flexible occurrences in both \( M_1 \) and \( M_2 \). In the latter case there is really nothing else to do.\(^2\) In the former case, we can write the equation \( X[s] \equiv M \) as \( X[s][t] \equiv M'[t] \) where \( t = \text{linweaken}(n; IA) \). Since \( t \) is injective this equation simplifies to \( X[s][t] \equiv M' \), which corresponds to changing every occurrence of \( n^I \) to \( n^A \). This is summarized by the rule int-aff-invert.

Strengthening affine variables. Consider now the case when \( \text{ff}' = AL \), i.e. \( n \) is affine. Since we know that \( n \) occurs affordly but should occur linearly, no more pruning will be necessary. This means that any flexible occurrence of \( n \) in an affine argument, say the \( i \)th, of some logic variable \( Y \) in \( M \) can be strengthened to a linear occurrence. Thus, as is summarized in the aff-strengthen rule we instantiate \( Y \) to \( Z[t] \), where \( Z \) is a fresh logic variable with \( A_Z = A_Y \), \( \Gamma_Z = \text{strengthen}(\Gamma_Y; i; AL) \), and \( t = \text{linweaken}(\Gamma_Y; i; AL) \). Since we know that \( n \) is supposed to be linear then it should also occur. If it does not, we can fail with the rule no-occur.

If none of the rules no-occur, aff-pos, additive, or aff-strengthen apply then \( n^AL \) satisfies the four properties of part 3 of Theorem 5.4 and can be strengthened from affine to linear using aff-lin-invert.

As an example we sketch how the algorithm solves equation (2) supposing that it has already been lowered. \( F_1[1L. \uparrow] \equiv c \circ H_1[1L. \uparrow] \Rightarrow F_1[1L. \uparrow] \equiv c \circ H_2[1A. \uparrow] \Rightarrow F_1[1A. \uparrow] \equiv c \circ H_2[1AA. \uparrow] \Rightarrow F_1[1AL. \uparrow] \equiv c \circ H_3[1AL. \uparrow] \Rightarrow F_1[1L. \uparrow] \equiv c \circ H_3[1L. \uparrow] \). The last equation is a pattern, which can be solved directly.

\(^2\)If we instead of a most general unifier were looking for the set of most general unifiers then we could easily enumerate the different possible solutions by introducing a disjunction and then either prune the variable from \( M_1 \) or \( M_2 \).
5.3 Correctness

The discussion above relies heavily on Theorem 5.3 and proves that the algorithm preserves solutions. It is therefore easily possible to generalize part 1 of the Correctness Theorem 4.1 to the version of the unification algorithm including linearity pruning. Termination (part 3) also holds for the extended algorithm with a slight elaboration of the termination ordering. When calculating the size of a term we will order the linearity flags $I > A > L$ because with this ordering, the strengthening rules int-strengthen, aff-strengthen, int-aff-invert, and aff-lin-invert decrease unification problems in size. Furthermore, we require that every introduction of the prune($\cdot;\cdot$) constraint is followed by a sequence of prune steps followed by a prune-fail or prune-finish step. When the introduction and elimination of the prune($\cdot;\cdot$) constraint are seen together as one step then the combined result always reduces the termination measure. However, since the extended algorithm can get stuck on certain equations with a “don’t know”, we have to accept that progress, as stated in part 2 of the theorem, no longer holds. In these cases we can simply report a set of leftover constraints, each of which require strengthening of some intuitionistic variable that occurs flexibly in multiple parts of the right-hand side.

6 Conclusion

We have defined the pattern fragment for higher-order unification problems in linear and affine type theory. We have proved that all higher-order unification equations within this fragment have no solutions or a most general unifier, and given an algorithm to construct it. Furthermore, we have extended the unification algorithm beyond the pattern fragment to those non-pattern equations that arise due to the additional constraints from the linear and affine type system.

References

[1] Iliano Cervesato & Frank Pfenning (1997): Linear higher-order pre-unification. In: Twelfth Annual Symposium on Logic in Computer Science — LICS’97, IEEE Computer Society Press, pp. 422–433.
[2] Iliano Cervesato & Frank Pfenning (2003): A Linear Spine Calculus. Journal of Logic Computation 13(5), pp. 639–688.
[3] Gilles Dowek, Thérèse Hardin, Claude Kirchner & Frank Pfenning (1998): Unification via Explicit Substitutions: The Case of Higher-Order Patterns. Rapport de Recherche 3591, INRIA. Preliminary version appeared at JICSLP’96.
[4] Gérard Huet (1975): A Unification Algorithm for Typed $\lambda$-Calculus. Theoretical Computer Science 1, pp. 27–57.
[5] Dale Miller (1991): A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification. Journal of Logic and Computation 1(4), pp. 497–536.
[6] Jason Reed (2009): Higher-order constraint simplification in dependent type theory. In: LFMTTP ’09: Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages, ACM, New York, NY, USA, pp. 49–56.
[7] Anders Schack-Nielsen & Carsten Schürmann (2008): System Description: Celf - A Logical Framework for Deductive and Concurrent Systems. In: Alessandro Armando, Peter Baumgartner & Gilles Dowek, editors: International Joint Conference on Automated Reasoning (IJCAR), Sydney, Australia, pp. 320–331.
[8] Anders Schack-Nielsen & Carsten Schürmann (2010): Curry-Style Explicit Substitutions for the Linear and Affine Lambda Calculus. In: International Joint Conference on Automated Reasoning (IJCAR), Springer Verlag, Edinburgh, UK. To appear.