

# Type Inference in Intuitionistic Linear Logic

Patrick Baillot

LIP - UMR 5668 CNRS ENS-Lyon UCBL INRIA -  
Université de Lyon  
patrick.baillot@ens-lyon.fr

Martin Hofmann

Ludwig-Maximilians-Universität München  
mhofmann@informatik.uni-muenchen.de

## Abstract

We study the type checking and type inference problems for intuitionistic linear logic: given a System F typed  $\lambda$ -term, (i) for an alleged linear logic type, determine whether there exists a corresponding typing derivation in linear logic (type checking) (ii) provide a concise description of all possible corresponding linear logic typings (type inference).

We solve these problems using a novel algorithmic type system for linear logic whose typing rules carry arithmetic side conditions describing essentially the nesting depth of (proof-net) boxes. By understanding these side conditions as unknowns we then reduce type inference to solving a system of arithmetic constraints. We show that these constraint systems fall into a tractable class hence leading to an efficient (polynomial-time) solution.

There are two important restrictions: first, our source language is typed System F rather than untyped lambda calculus; this is necessary because type inference for System F is known to be undecidable. Second, we assume that sharing is made explicit in the input, thus we do not try to automatically infer opportunities for sharing identical subterms. Relieving the latter restriction is left as a challenge for future work.

**Categories and Subject Descriptors** F.4.1 [Mathematical Logic and formal languages]: Mathematical logic—lambda calculus and related systems; F.3.3 [Logics and meanings of programs]: Studies of program constructs—type structure

**General Terms** algorithms, theory

**Keywords** Lambda calculus, Linear logic, Type inference, Type systems

## 1. Introduction

Linear Logic has been invented by Girard [11] motivated by the desire to model consumption of resources as modus ponens and also by proof-theoretic considerations. It has subsequently been advocated and successfully used [9, 15, 20, 21] as a type system for programming languages. The idea behind such linear type systems is that they are capable of tracking whether a variable or a reference will be used at most once. In this case, it is safe to deallocate it after that usage has happened [13, 20]; it is also sound to alter the type of that variable after an update has happened (“strong update”) [10].

These type systems based on linear logic restrict the occurrence of the  $!$ -operator to the left-hand side of function spaces, possibly using a different syntax, because then type checking and linearity inference essentially reduce to tracking the number of uses of variables which can be done by augmentation of standard type checking algorithms such as Hindley-Milner [3, 18]. The results of this paper constitute a step towards the removal of such restrictions which would allow the general syntax of linear logic with exponentials to be used in practical type systems.

In this paper we study type checking and in particular linearity inference for (intuitionistic) linear logic in its standard form, where the  $!$ -operator is allowed anywhere. This poses interesting new challenges as can be seen from the following example: Suppose that  $x:!(B \multimap C), y:!(A \multimap B), z:!A$  and we are to check that  $t \equiv x(yz)$  can be typed  $!C$ . A priori, this might not be true since  $x$  returns results of type  $C$  only. Since, however, all the free variables of  $t$  have  $!$ -type the promotion rule allows one to improve  $t$ 's type to  $!C$ . Note that this reasoning is also required for typechecking the application of  $x$  to  $yz$  since  $x$  expects arguments of type  $!B$ . Furthermore, if the  $!$ -annotations of the variables are not or only partially given then we have many more possibilities and there does in particular not exist a “best” one in the sense of the subtyping induced by  $!A <: A$ . Note also that if the  $!$ -annotations of the variables are not given, a particular (but uninformative) solution is obtained by using the standard translation of intuitionistic types into linear logic:  $(A \multimap B)^* = !A^* \multimap B^*$ .

So, rather than computing a minimal type, we compute a system of arithmetic constraints of polynomial size whose solutions correspond to the possible typings of a term. Moreover, the constraint systems fall into a class for which satisfiability is feasible in polynomial time.

We assume, however, that we are given a valid system F typing and only consider decorations with  $!$  of that given typing. This is because type inference for system F is undecidable and the study of restricted fragments for which it is decidable, e.g. the standard ML type system, is orthogonal to the interests of this paper.

**Outline.** We define in Sect. 3 the linear system F we will be considering and the type checking (resp. type inference) problems. We then provide in Sec. 4 an algorithmic typing system. We turn this into a type inference algorithm in Sect. 5 and prove that it runs in polynomial time.

## 2. Preliminaries

The types of system F are given by the grammar

$$A ::= \alpha \mid A_1 \multimap A_2 \mid \forall \alpha. A$$

where  $\alpha$  ranges over a set of type variables. The variable  $\alpha$  is bound in  $\forall \alpha. A$ .

We write  $A[B/\alpha]$  for the (capture avoiding) substitution of  $B$  for  $\alpha$  in  $A$ .

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PPDP'10, July 26–28, 2010, Hagenberg, Austria.  
Copyright © 2010 ACM 978-1-4503-0132-9/10/07...\$10.00

Church-style lambda terms are given by the grammar

$$t ::= x \mid t_1 t_2 \mid \lambda x:A.t \mid t[A] \mid \Lambda \alpha.t$$

The set  $FV(t)$  of free variables of term  $t$  is defined as usual with the understanding that  $\lambda x.t$  binds  $x$  in  $t$ . Similarly, we define the set  $FTV(T)$  (resp.  $FTV(t)$ ) of type variables occurring free in the type  $T$  (resp. in the term  $t$ ).

A typing context  $\Gamma$  is a mapping of (term) variables to types. We denote  $dom(\Gamma)$  the set of variables bound, i.e. assigned a type, by  $\Gamma$ . If  $\Gamma$  is a context and  $x \notin dom(\Gamma)$  then  $\Gamma, x:A$  denotes the extension of  $\Gamma$  with the binding of  $x$  to  $A$ . Whenever this notation is used it is presumed that  $x$  is not already bound in  $\Gamma$ . Likewise,  $\Gamma, \Delta$  denotes the union of two disjoint contexts (in the sense that  $dom(\Gamma) \cap dom(\Delta) = \emptyset$ ). We define  $FTV(\Gamma) = \bigcup_{x \in dom(\Gamma)} FTV(\Gamma(x))$ .

The typing judgement takes the form  $\Gamma \vdash t : A$ ; its definition is given in Figure 1.

The following rules of weakening, contraction and substitution are easily seen to be admissible.

$$\frac{\Gamma \vdash t : A}{\Gamma, x:B \vdash t : A} \text{(WEAK)}$$

$$\frac{\Gamma, x_1:A, x_2:A \vdash t : B}{\Gamma, x:A \vdash t[x/x_1, x/x_2] : B} \text{(CONTR)}$$

$$\frac{\Gamma, x:A \vdash t : B \quad \Gamma \vdash t' : A}{\Gamma \vdash t[t'/x] : B} \text{(CUT)}$$

### 3. Linear System F

Now we consider the types of linear (affine) system F (LLF), which are given by the following grammar:

$$T ::= \alpha \mid T_1 \multimap T_2 \mid \forall \alpha.T \mid !T$$

If  $\Gamma$  is an LLF context, the context  $! \Gamma$  is defined by:  $(! \Gamma)(x) = !T$  if  $\Gamma(x) = T$ .

The erasure  $|T|$  of LLF type  $T$  is the system F type defined by the following clauses:

$$\begin{aligned} |\alpha| &= \alpha \\ |T_1 \multimap T_2| &= |T_1| \multimap |T_2| \\ |\forall \alpha.T| &= \forall \alpha. |T| \\ |!T| &= |T| \end{aligned}$$

If  $\Gamma$  is an LLF context, the system F context  $|\Gamma|$  is defined in the expected way.

The typing judgement  $\Gamma \vdash t : T$  where this time  $\Gamma$  binds variables to LLF types is defined by the rules in Figure 2. Note the splitting of context in the binary rules LLF-APP and LLF-CUT, and the fact that contraction and weakening are given by explicit rules.

It can be shown that whenever  $\Gamma, x:S \vdash t : T$  and  $x$  occurs more than once in  $t$  then  $S$  is of the form  $!T'$ . If we did not have the side condition on number of occurrences in the rule LLF-CUT then this would not hold. For example, we would then be able to derive the judgement

$$x:\beta \multimap !\alpha, y:\beta, f:!\alpha \multimap !\alpha \multimap \gamma \vdash f(xy)(xy) : \gamma$$

In our presentation this judgement is not derivable, however, the related judgement

$$x:\beta \multimap !\alpha, y:\beta, f:!\alpha \multimap !\alpha \multimap \gamma \vdash \text{let } z:!\alpha = xy \text{ in } fzz : \gamma$$

where

$$\text{let } x:A = t_1 \text{ in } t_2 := (\lambda x:A.t_2)t_1$$

is derivable. We are aware of the fact that subject reduction with respect to full  $\beta$  reduction does therefore not hold for our presentation. An alternative would be to introduce an explicit sharing operation in our term syntax which again would block reduction. The point is that we would like to separate the problem of type reconstruction from the problem of discovering sharing opportunities which might be more difficult.

We also note that the rule LLF-CUT is no longer redundant; this is one of the reasons why the type checking problem which we will consider is nontrivial.

Here are a few examples of derivable LLF typing judgements for the term  $x(yz)$ :

$$\begin{aligned} x:!(B \multimap C), y:!(A \multimap B), z:!\alpha \vdash x(yz) : !C \\ x:!(B \multimap C), y:!(A \multimap B), z:!\alpha \vdash x(yz) : !C \\ x:!(B \multimap C), y:!(A \multimap B), z:\alpha \vdash x(yz) : C \end{aligned}$$

Fig. 3 gives an example of derivation for the first of these judgements.

To automatize the search for such derivations, we consider the following *type checking problem*:

**Type checking for LLF:** Given  $t, T, \Gamma$  is  $\Gamma \vdash t : T$  a valid LLF-judgement?

Observe that this problem is not trivial because the rules are not syntax-directed. In particular it is not obvious when to apply rules LLF-CUT, LLF-PROM, LLF-DER and LLF-DIG. Actually there is also no obvious way to guess beforehand how many LLF-PROM occurrences will be needed in the derivation.

In order to attack this problem we will now introduce an algorithmic version of LLF whose rules are (almost) syntax-directed and thus admit a constraint-based type checking algorithm. In this way we also obtain a solution to the *type inference problem*:

**Type inference for LLF:**

Given a system F type  $A$ , context  $\Delta$  and term  $t$ , find a concise description of the set of LLF types  $T$  and contexts  $\Gamma$  with  $\Gamma \vdash t : T$  and  $|T| = A, |\Gamma| = \Delta$ .

### 4. Algorithmic LLF

The algorithmic system will remove some of the non syntax-directed rules. First, LLF-CONTR and LLF-WEAK will be integrated into other rules. Second one wants to avoid the need for the LLF-CUT rule. A key point for that will be to "decompose" the LLF-PROM rule into two more basic rules, that we will call ENTER and LEAVE.

A possible intuition for this approach comes from linear logic proof-nets [11], where the LLF-PROM rule corresponds to a *box*: in some sense we will here keep with rules ENTER and LEAVE the information about the doors of boxes, but leave implicit which doors correspond to the same box. In order to be able to reconstruct the boxes or LLF-PROM rules, we will need to carry in the judgements an extra piece of information, a natural integer associated to each variable of the context.

#### 4.1 Algorithmic typing contexts

An algorithmic typing context  $\Gamma$  is a finite map on both term and type variables. To a term variable in its domain it assigns a pair  $(T, m)$  with  $T$  an LLF type and  $m$  a natural number. To a type variable  $\alpha$  it assigns a pair  $(\star, m)$  where  $\star$  is a dummy value that can be thought of as the type of all types. For convenience, we let  $X, Y, \dots$  range over both term and type variables and we let  $U, V, \dots$  range over types and  $\star$ . If  $\Gamma(X) = (U, m)$  then we write

$\frac{}{\Gamma, x:A \vdash x : A} \text{(F-VAR)}$ $\frac{\Gamma, x:A \vdash t : B}{\Gamma \vdash \lambda x:A.t : A \rightarrow B} \text{(F-LAM)}$ $\frac{\Gamma \vdash t : A \quad \alpha \notin FTV(\Gamma)}{\Gamma \vdash \Lambda \alpha.t : \forall \alpha.A} \text{(F-TLAM)}$	$\frac{\Gamma \vdash t_1 : A \rightarrow B \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1 t_2 : B} \text{(F-APP)}$ $\frac{\Gamma \vdash t : \forall \alpha.A}{\Gamma \vdash t[B/\alpha] : A[B/\alpha]} \text{(F-TAPP)}$
--	---

**Figure 1.** Typing rules for system F

$\frac{}{x:T \vdash x : T} \text{(LLF-VAR)}$ $\frac{\Gamma, x:S \vdash t : T}{\Gamma \vdash \lambda x: S .t : S \multimap T} \text{(LLF-LAM)}$ $\frac{\Gamma \vdash t : T \quad \alpha \notin FTV(\Gamma)}{\Gamma \vdash \Lambda \alpha.t : \forall \alpha.T} \text{(LLF-TLAM)}$ $\frac{\Gamma \vdash t : T}{!\Gamma \vdash t : !T} \text{(LLF-PROM)}$ $\frac{\Gamma \vdash t : !T}{\Gamma \vdash t : T} \text{(LLF-DER)}$ $\frac{\Gamma, x_1:!S, x_2:!S \vdash t : T}{\Gamma, x:!S \vdash t[x/x_1, x/x_2] : T} \text{(LLF-CONTR)}$	$\frac{\Gamma, x:S \vdash t : T \quad \Delta \vdash t' : S \quad \begin{array}{l} x \text{ has at most one free} \\ \text{occurrence in } t \end{array}}{\Gamma, \Delta \vdash t[t'/x] : T} \text{(LLF-CUT)}$ $\frac{\Gamma \vdash t_1 : S \multimap T \quad \Delta \vdash t_2 : S}{\Gamma, \Delta \vdash t_1 t_2 : T} \text{(LLF-APP)}$ $\frac{\Gamma \vdash t : \forall \alpha.T}{\Gamma \vdash t[ S ] : T[S/\alpha]} \text{(LLF-TAPP)}$ $\frac{\Gamma \vdash t : !T}{\Gamma \vdash t : !!T} \text{(LLF-DIG)}$ $\frac{\Gamma \vdash t : T}{\Gamma, x:S \vdash t : T} \text{(LLF-WEAK)}$
---	---

**Figure 2.** Typing rules for system LLF

$\frac{z:!A \vdash z : !A}{z:!A \vdash z : !!A} \text{VAR DIG}$ $\frac{y:!(A \multimap B) \vdash y : !(A \multimap B)}{y:!(A \multimap B) \vdash y : !!(A \multimap B)} \text{DIG}$ $\frac{x:!(B \multimap C) \vdash x : !(B \multimap C)}{x:!(B \multimap C), y:!(A \multimap B), z:!!A \vdash x(yz) : !C} \text{CUT}$	$\frac{y:A \multimap B \vdash y : A \multimap B \quad z:A \vdash z : A}{z:A \vdash z : A} \text{VAR APP}$ $\frac{y:A \multimap B, z:A \vdash yz : B}{y:!(A \multimap B), z:!A \vdash yz : !B} \text{PROM APP}$ $\frac{x:!(B \multimap C), y:!(A \multimap B), z:!!A \vdash x(yz) : C}{x:!(B \multimap C), y:!(A \multimap B), z:!!A \vdash x(yz) : !C} \text{PROM CUT}$
---	---

**Figure 3.** Example of LLF type derivation

$\Gamma[X] = U$  and  $\Gamma\{X\} = m$ . An algorithmic LLF context is well-formed only if whenever  $x \in \text{dom}(\Gamma)$  and  $\alpha \in FV(\Gamma[x])$  then  $\alpha \in \text{dom}(\Gamma)$  too and  $\Gamma\{\alpha\} \geq \Gamma\{x\}$ .

If  $\Gamma$  is an algorithmic typing context and  $c \in \mathbb{Z}$  then  $\Gamma^{+c}$  ok means that  $\Gamma\{X\} + c \geq 0$  for all  $X \in \text{dom}(\Gamma)$ .

In this case,  $\Gamma^{+c}$  denotes the algorithmic typing context defined by  $\text{dom}(\Gamma^{+c}) = \text{dom}(\Gamma)$  and  $(\Gamma^{+c})(X) = (U, m + c)$  when  $\Gamma(X) = (U, m)$ . Note that for  $c \geq 0$  one always has  $\Gamma^{+c}$  ok.

We denote  $!\Delta$  the algorithmic typing context defined by  $(!\Delta)(x) = (!\Delta[x], \Delta\{x\})$  and  $(!\Delta)(\alpha) = (\star, \Delta\{\alpha\})$ .

If  $\Gamma$  is an ordinary typing context then  $\Gamma^0$  denotes the algorithmic typing context given by  $\text{dom}(\Gamma^0) = \text{dom}(\Gamma) \cup FTV(\Gamma)$  and  $\Gamma^0(x) = (\Gamma(x), 0)$  for  $x \in \text{dom}(\Gamma)$  and  $\Gamma^0(\alpha) = (\star, 0)$  for  $\alpha \in FTV(\Gamma)$ .

We will often need the algorithmic typing context to bind some type variables that appear in the type  $T$  of the subject. So, given an ordinary typing context  $\Gamma$  and a set  $\mathcal{E}$  of type variables, we will

denote by  $(\Gamma + \mathcal{E})^0$  the algorithmic typing context  $\Delta$  defined by:

$$\begin{aligned} \Delta(X) &= \Gamma^0(X) \text{ if } X \in \text{dom}(\Gamma^0), \\ \Delta(\alpha) &= (\star, 0) \text{ if } \alpha \in \mathcal{E}. \end{aligned}$$

If the intersection  $\text{dom}(\Gamma) \cap \text{dom}(\Delta)$  contains only type variables and  $\Gamma(\alpha) = \Delta(\alpha)$  for all  $\alpha \in \text{dom}(\Gamma) \cap \text{dom}(\Delta)$  then  $\Gamma$  and  $\Delta$  are *compatible*. In this case we denote  $\Gamma, \Delta$  the union of  $\Gamma$  and  $\Delta$  maps. Whenever this notation is used it is presumed that  $\Gamma$  and  $\Delta$  are compatible.

An algorithmic typing judgement takes the form  $\Gamma \vdash_a t : T$  where  $\Gamma$  is an algorithmic typing context and is given by the rules in Figure 4. The bindings of type variables in particular prevent ENTER immediately after (bottom-up) TLAM and thus the (illegal) derivation of  $\vdash_a t : \forall \alpha. !T \multimap !\forall \alpha.T$  for some  $T$  which is not in general provable in LLF, e.g. when  $T = (\alpha \multimap \beta) \multimap \alpha$ .

We can observe the following property:

$\frac{\Gamma(x) = (T, 0)}{\Gamma \vdash_a x : T} \text{(VAR)}$ $\frac{\Gamma, x : (S, 0) \vdash_a t : T}{\Gamma \vdash_a \lambda x :  S . t : S \multimap T} \text{(LAM)}$ $\frac{\Gamma \vdash_a t_1 : S \multimap T \quad \Gamma \vdash_a t_2 : S \quad x \in FV(t_1) \cap FV(t_2) \Rightarrow \exists T. \Gamma[x] = !T}{\Gamma \vdash_a t_1 t_2 : T} \text{(APP)}$	
$\frac{\Gamma, \alpha : (\star, 0) \vdash_a t : T \quad \alpha \notin FTV(\Gamma)}{\Gamma \vdash_a \Lambda \alpha. t : \forall \alpha. T} \text{(TLAM)}$ $\frac{\Gamma \vdash_a t : !T}{\Gamma^{+1}, \Delta^0 \vdash_a t : T} \text{(ENTER)}$ $\frac{\Gamma \vdash_a t : !T}{\Gamma \vdash_a t : T} \text{(DER)}$	$\frac{\Gamma \vdash_a t : \forall \alpha. T \quad FTV(S) \subseteq \text{dom}(\Gamma)}{\Gamma \vdash_a t[S] : T[ S /\alpha]} \text{(TAPP)}$ $\frac{\Gamma \vdash_a t : T \quad \Gamma^{-1} \text{ ok}}{\Gamma^{-1} \vdash_a t : !T} \text{(LEAVE)}$ $\frac{\Gamma \vdash_a t : !T}{\Gamma \vdash_a t : !!T} \text{(DIG)}$

Figure 4. Algorithmic typing rules

LEMMA 1. *If  $\Gamma \vdash_a t : T$  and if  $\alpha \in FTV(\Gamma) \cup FTV(T) \cup FTV(t)$ , then we have  $\alpha \in \text{dom}(\Gamma)$ .*

**Proof :** It can easily be proved by induction on the derivation, and essentially follows from the fact that the algorithmic typing contexts in VAR rules are well-formed and from the side condition on the rule TAPP.  $\square$

This algorithmic system is similar to Pfenning and Wong’s “implicit” presentation of the modal logic S4 [17] and also to Martini and Masini’s presentation of linear logic [16]. It can also be motivated by Baillot and Terui’s decomposition [2] of boxes into opening and closing doors required to obey some bracketing conditions. In fact, it arose as a judgemental version of that work; the relationship to Pfenning-Wong and Martini-Masini was only discovered afterwards. An important difference to [17] is that applications of rules ENTER and LEAVE are not recorded in the terms thus making type checking and decoration more challenging but also more useful.

## 4.2 Soundness and completeness of algorithmic LLF

We start by proving that certain rules are admissible in algorithmic LLF, which will then be useful to prove completeness of the system w.r.t. LLF.

LEMMA 2 (contraction). *If  $\Gamma, x_1 : (!A, m), x_2 : (!A, m) \vdash_a t : C$  then also  $\Gamma, x : (!A, m) \vdash_a t[x/x_1, x/x_2] : C$ .*

**Proof :** By induction on derivations. The interesting case is when the assumption has been derived by rule APP, i.e.,  $t = t_1 t_2$ , and when  $x_1 \in FV(t_1)$  and  $x_2 \in FV(t_2)$ . In this case,  $x$  becomes a shared variable of  $t_1[x/x_1, x/x_2]$  and  $t_2[x/x_1, x/x_2]$  so that the additional side condition  $(\Gamma, x : (!A))(x) = !(\dots)$  arises. Fortunately, it is satisfied since  $(\Gamma, x : (!A))[x] = !A$ .

The other cases are straightforward.  $\square$

LEMMA 3 (weakening). *1. If  $\Gamma \vdash_a t : T$  then we also have  $\Gamma, x : (A, m) \vdash_a t : T$ .*  
*2. If  $\Gamma \vdash_a t : T$  and  $\alpha \notin FTV(\Gamma)$ , then we also have  $\Gamma, \alpha : (\star, m) \vdash_a t : T$ .*

**Proof :** For 1. we proceed by induction on derivations. Consider the case of ENTER. Then  $\Gamma$  is of the shape  $\Gamma = \Gamma_0^{+1}, \Delta^0$ , and the premise is of the form  $\Gamma_0 \vdash_a t : !T$ .

If  $m = m_0 + 1$ , by IH we can derive  $\Gamma_0, x : (A, m_0) \vdash_a t : !T$  and conclude using ENTER.

If  $m = 0$  then applying ENTER we obtain  $\Gamma_0^{+1}, \Delta^0, x : (A, 0) \vdash_a t : T$ , as required.

The other cases are straightforward. The proof of statement 2. is analogous.  $\square$

LEMMA 4 (strengthening). *If  $\Gamma, x : (A, m) \vdash_a t : T$  and  $x \notin FV(t)$ , then  $\Gamma \vdash_a t : T$ .*

**Proof :** By induction on derivations. Consider the case of ENTER. We have  $\Gamma, x : (A, m) = \Gamma_0^{+1}, \Delta^0$  and the premise is  $\Gamma_0 \vdash_a t : !T$ .

First, if  $x \in FV(\Delta^0)$ , then  $m = 0$ . Take  $\Theta^0$  such that  $\Delta^0 = \Theta^0, x : (A, 0)$ . Then by applying ENTER to  $\Gamma_0 \vdash_a t : !T$  we get  $\Gamma_0^{+1}, \Theta^0 \vdash_a t : T$ , as required.

Second, if  $x \in FV(\Gamma_0^{+1})$  then  $m = m_0 + 1$  and  $\Gamma_0 = \Theta, x : (A, m_0)$ . By applying the IH we get  $\Theta \vdash_a t : !T$ . Then with ENTER we obtain  $\Theta^{+1}, \Delta^0 \vdash_a t : T$ , as expected.

Consider the case of LEAVE. The premise is of the form  $\Gamma^{+1}, x : (A, m + 1) \vdash_a t : T_0$ , and  $T = !T_0$ . Then by IH we get  $\Gamma^{+1} \vdash_a t : T_0$  and we conclude with LEAVE.

The other cases are easy.  $\square$

LEMMA 5 (cut). *If  $\Gamma, x : (S, c) \vdash_a t : T$  and  $\Delta \vdash_a t' : S$  and  $x$  occurs at most once in  $t$ , and  $\Gamma$  and  $\Delta^{+c}$  are compatible then  $\Gamma, \Delta^{+c} \vdash_a t[t'/x] : T$ .*

**Proof :** First, if  $x \notin FV(t)$  then  $t[t'/x] = t$ ; by applying Lemma 4 on  $\Gamma, x : (S, c) \vdash_a t : T$  we get  $\Gamma \vdash_a t : T$ , and then with Lemma 3 we deduce  $\Gamma, \Delta^{+c} \vdash_a t : T$ .

We thus only need to prove the claim in the case where  $x \in FV(t)$ . We proceed by induction on the derivation of  $\Gamma, x : (S, c) \vdash_a t : T$ .

VAR: By assumption we have  $t = x$ , thus  $c = 0$  and using Lemma 3 we get  $\Gamma, \Delta \vdash_a t' : T$ , as required.

APP: since  $x$  occurs once in  $t$  it cannot occur in both  $t_1$  and  $t_2$ . Applying the IH to the term containing  $x$  followed by APP yields the result.

ENTER: Here  $\Gamma = \Gamma_1^{+1}, \Gamma_2^0$ , and as  $x \in FV(t)$  we have  $c = c_0 + 1$ . The premise is  $\Gamma_1, x : (S, c_0) \vdash_a t : !T$ , and by IH we get  $\Gamma_1, \Delta^{c_0} \vdash_a t[t'/x] : T$ . By applying ENTER we then get  $\Gamma_1^{+1}, \Delta^{c_0+1}, \Gamma_2^0 \vdash_a t[t'/x] : T$ , as required.

The case of LEAVE is handled in a similar way, and the other cases are straightforward.  $\square$

LEMMA 6. If  $\Gamma, x:(S, m) \vdash_a t : T$  then  $\Gamma, x:(!S, m+1) \vdash_a t : T$ .

**Proof :** This is not an immediate consequence of the previous lemma because we do not require that  $x$  occurs at most once in  $t$ . Instead, we prove the lemma by an induction on derivations. The only significant case is that of the VAR rule. In that case if  $t = y \neq x$  then the result is trivial, otherwise if  $y = x$  we have  $m = 0$  and we have: by VAR we get  $x : (!S, 0) \vdash_a x : !S$ , then ENTER yields  $x : (!S, 1) \vdash_a x : S$ , and we conclude with the weakening Lemma 3.  $\square$

LEMMA 7. If  $\Gamma, \alpha:(\star, m) \vdash_a t : T$  then  $\Gamma, \alpha:(\star, m+1) \vdash_a t : T$ .

**Proof :** Straightforward induction on derivations.  $\square$

THEOREM 8 (Completeness of algorithmic LLF).

If  $\Gamma \vdash t : T$  and  $\mathcal{E} = FTV(T) \cup FTV(t)$ , then we have  $(\Gamma + \mathcal{E})^0 \vdash_a t : T$ .

Note that as a particular case, if  $FTV(T) \cup FTV(t) \subseteq FTV(\Gamma)$  then  $\Gamma^0 \vdash_a t : T$ .

**Proof :** We prove this statement by induction on derivations. Rule LLF-CUT is dealt with by Lemma (5), LLF-CONTR by Lemma (2) and LLF-WEAK by Lemma (3).

Consider the case of rule LLF-TAPP. By induction hypothesis we have  $(\Gamma + \mathcal{E})^0 \vdash_a t : \forall \alpha.T$ . Take  $\mathcal{E}' = \mathcal{E} \cup FTV(S)$ . By using Lemma 3(2) we get  $(\Gamma + \mathcal{E}')^0 \vdash_a t : \forall \alpha.T$ . The side condition for applying rule TAPP with  $S$  is then fulfilled, and thus we get  $(\Gamma + \mathcal{E}')^0 \vdash_a t[S] : T[|S|/\alpha]$ .

For LLF-PROM suppose that  $\Gamma = !\Delta$  and  $T = !T_0$  and  $\Delta \vdash t : T_0$ . The induction hypothesis yields  $(\Delta + \mathcal{E})^0 \vdash_a t : T_0$ , with  $\mathcal{E} = FTV(T_0) \cup FTV(t)$ . By applying Lemmas 6 and 7 we get  $((!\Delta + \mathcal{E})^0)^{+1} \vdash_a t : T_0$ , and with rule LEAVE we have  $(!\Delta + \mathcal{E})^0 \vdash_a t : !T_0$ .

All other rules are straightforward applications of the induction hypothesis.  $\square$

In order to prove soundness of algorithmic LLF by induction on derivations we must ascribe a meaning in terms of LLF to sequents that are not of the form  $\Gamma^0 \vdash_a t : T$  for some  $\Gamma$ . We do this now.

Suppose that  $\Gamma \vdash_a t : T$  and write  $dom(\Gamma) = \{X_1, \dots, X_n\}$  such that  $i \leq j$  implies  $\Gamma\{X_i\} \leq \Gamma\{X_j\}$  and if in addition  $\Gamma\{X_i\} = \Gamma\{X_j\}$  and  $X_i$  is a type variable then so is  $X_j$ . In other words, as the index decreases the level never goes up and at any level the type variables come first.

Call such an ordering of the variables in  $\Gamma$  *compatible*. We associate with the judgement  $\Gamma \vdash_a t : T$  an LLF judgement  $\ulcorner \Gamma \mid t : T \urcorner$  recursively by the following clauses. In order for the recursion to go through we have to simultaneously translate judgements of the form  $\Delta \vdash_a t' : T'$  where  $dom(\Delta) = \{X_k, \dots, X_n\}$  for some  $k \geq 1$ .

$$\begin{aligned} \ulcorner t : T \urcorner &= \vdash t : T \\ \ulcorner \Delta, x_k:(S, 0) \mid t : T \urcorner &= \ulcorner \Delta \mid \lambda x_k. |T|. t : S \multimap T \urcorner \text{ when} \\ &\quad X_k \in dom(\Delta) \Rightarrow i \geq k \\ \ulcorner \Delta, \alpha_k:(\star, 0) \mid t : T \urcorner &= \ulcorner \Delta \mid \Lambda \alpha_k. t : \forall \alpha_k. T \urcorner \text{ when} \\ &\quad X_k \in dom(\Delta) \Rightarrow i > k \\ \ulcorner \Delta^{+1} \mid t : T \urcorner &= \ulcorner \Delta \mid t : !T \urcorner \end{aligned}$$

Note that in the third clause well-formedness of  $\Delta, \alpha_k : (\star, 0)$  and the conditions on the variable numbering ensure that  $\alpha_k \notin FTV(\Delta)$  so that  $\Delta$  itself is well-formed. See Figure 5 for an example.

This translation is essentially the same as Guerrini, Martini and Masini's [12]  $(-)^{\sharp}$  translation of their 2-sequents. However, since they only consider provability and not typability, we cannot use it directly. Furthermore, they prove soundness only with respect to a

different translation  $(-)^b$  which in our notation would amount to simply removing all the indices. They then circumvent the difficulty that LEAVE is not sound for that translation using syntactic considerations which do not seem to apply here.

LEMMA 9. If  $\ulcorner \Gamma \mid v[(\lambda x. |S|. t)y/z] : T \urcorner$  then we also have  $\ulcorner \Gamma \mid v[t[y/x]/z] : T \urcorner$ .

**Proof :** We note that  $\ulcorner \Gamma \mid u : T \urcorner$  is a judgement of the form  $\vdash \lambda \vec{z}. u : U$  for  $u \in \{v[(\lambda x. |S|. t)y/z], v[t[y/x]/z]\}$  with  $U$  depending only on  $\Gamma$  and  $T$ . The claim then follows from the fact that LLF is closed under  $\beta$ -reduction of redexes of the form  $(\lambda x. t)y$ , where  $y$  is a variable. It is in fact also closed under arbitrary *linear*  $\beta$ -reduction but we do not need this here.  $\square$

LEMMA 10. If  $\ulcorner \Gamma \mid v[\Lambda \alpha. t[\alpha]/z] : T \urcorner$  then  $\ulcorner \Gamma \mid v[t/z] : T \urcorner$ .

**Proof :** As in the previous lemma this follows from the fact that LLF judgements are preserved by reduction of a term of the form  $v[\Lambda \alpha. t[\alpha]/z]$  into  $v[t/z]$ .  $\square$  Now, the following substitution lemma will be a key to prove afterwards the soundness theorem of algorithmic LLF.

LEMMA 11. Suppose that  $\ulcorner \Gamma_i \mid t_i : T_i \urcorner$  for  $i = 1, \dots, n$  and  $\Delta \vdash t : T$  where  $\Delta = x_1:T_1, \dots, x_n:T_n$ . Suppose furthermore that each  $x_i$  occurs at most once in  $t$  and that the  $\Gamma_i$  are compatible. Then  $\ulcorner \Gamma_1, \dots, \Gamma_n \mid t[t_1/x_1, \dots, t_n/x_n] : T \urcorner$ .

Here the translations of  $\Gamma_i \vdash_a t_i : T_i$  are understood w.r.t. the ordering of the variables that arises as the restriction of a fixed ordering of the variables in  $\Gamma_1, \dots, \Gamma_n$  used to compute the translation of  $\Gamma_1, \dots, \Gamma_n \vdash t[t_1/x_1, \dots, t_n/x_n] : T$ .

**Proof :** We proceed by induction on the quantity:

$$\sum_{i=1}^n \sum_{X \in dom(\Gamma_i)} (\Gamma_i\{X\} + 1).$$

First, if this quantity is 0 then  $\forall i \in \{1, \dots, n\}, \Gamma_i = \emptyset$ . In this case  $\ulcorner \Gamma_i \mid t_i : T_i \urcorner$  gives  $\vdash t_i : T_i$ , and we can conclude by using the rule LLF-CUT.

Otherwise, fix a compatible ordering on  $\Gamma_1, \dots, \Gamma_n$  and suppose w.l.o.g. that the first variable  $X$  in this ordering is bound in  $\Gamma_1$ .

**Case  $X = x$  and  $\Gamma_1\{x\} = 0$ .** We then write  $\Gamma_1 = \Delta_1, x:(U, 0)$  and we have

$$\ulcorner \Gamma_1 \mid t_1 : T_1 \urcorner = \ulcorner \Delta_1 \mid \lambda x. |U|. t_1 : U \multimap T_1 \urcorner$$

We also have  $y_1:U \multimap T_1, x_2:T_2, \dots, x_n:T_n \vdash \lambda x. |U|. t[y_1x/x_1] : U \multimap T$  thanks to the rule LLF-CUT.

The induction hypothesis furnishes (we write  $\lambda x. t_1$  instead of  $\lambda x. |U|. t_1$  and  $\Gamma' = \Gamma_2, \dots, \Gamma_n$  for readability):

$$\ulcorner \Delta_1, \Gamma' \mid \lambda x. |U|. t[(\lambda x. t_1)x/x_1, t_2/x_2, \dots, t_n/x_n] : U \multimap T \urcorner.$$

So, by Lemma 9,

$$\ulcorner \Delta_1, \Gamma' \mid \lambda x. |U|. t[t_1/x_1, t_2/x_2, \dots, t_n/x_n] : U \multimap T \urcorner,$$

and, finally by the definition of  $\ulcorner - \mid - \urcorner$

$$\ulcorner \Gamma_1, \Gamma_2, \dots, \Gamma_n \mid t[t_1/x_1, t_2/x_2, \dots, t_n/x_n] : T \urcorner.$$

**Case  $X = \alpha$  and  $\Gamma_1\{\alpha\} = 0$ .** We then write  $\Gamma_1 = \Delta_1, \alpha:(\star, 0)$  and we have

$$\ulcorner \Gamma_1 \mid t_1 : T_1 \urcorner = \ulcorner \Delta_1 \mid \Lambda \alpha. t_1 : \forall \alpha. T_1 \urcorner.$$

Using rules LLF-CUT and LLF-TLAM we get:

$$y_1:\forall \alpha. T_1, x_2:T_2, \dots, x_n:T_n \vdash \Lambda \alpha. t[y_1\alpha/\alpha]/x_1] : \forall \alpha. T.$$

$$\begin{aligned}
& \lceil x_5:(T_5, 3), \alpha_4:(\star, 2), x_3:(T_3, 2), x_2:(T_2, 2), x_1:(T_1, 0) \mid t : T^\neg \\
= & \lceil x_5:(T_5, 3), \alpha_4:(\star, 2), x_3:(T_3, 2), x_2:(T_2, 2) \mid \lambda x_1:|T_1|.t : T_1 \multimap T^\neg \\
= & \lceil x_5:(T_5, 1), \alpha_4:(\star, 0), x_3:(T_3, 0), x_2:(T_2, 0) \mid \lambda x_1:|T_1|.t : \!(T_1 \multimap T)^\neg \\
= & \lceil x_5:(T_5, 1) \mid \Lambda \alpha_4.\lambda x_3:|T_3|.\lambda x_2:|T_2|.\lambda x_1:|T_1|.t : \forall \alpha_4.T_3 \multimap T_2 \multimap \!(T_1 \multimap T)^\neg \\
= & \vdash \lambda x_5:|T_5|.\Lambda \alpha_4.\lambda x_3:|T_3|.\lambda x_2:|T_2|.\lambda x_1:|T_1|.t : T_5 \multimap \!(\forall \alpha_4.T_3 \multimap T_2 \multimap \!(T_1 \multimap T))
\end{aligned}$$

**Figure 5.** Example of judgement translation from algorithmic LLF to LLF

By IH we then get:

$$\lceil \Delta_1, \Gamma_2, \dots, \Gamma_n \mid \Lambda \alpha.t[(\Lambda \alpha.t_1[\alpha]/x_1, t_2/x_2, \dots, t_n/x_n) : \forall \alpha.T^\neg].$$

So, by Lemma 10,

$$\lceil \Delta_1, \Gamma_2, \dots, \Gamma_n \mid \Lambda \alpha.t[t_1/x_1, t_2/x_2, \dots, t_n/x_n] : \forall \alpha.T^\neg.$$

and, finally by the definition of  $\lceil - \mid - : -^\neg$

$$\lceil \Gamma_1, \Gamma_2, \dots, \Gamma_n \mid t[t_1/x_1, t_2/x_2, \dots, t_n/x_n] : T^\neg.$$

If the previous cases do not apply then we must have:

**Case**  $\Gamma_i = \Delta_i^{+1}$  for  $i = 1, \dots, n$ . We then have

$$\lceil \Gamma_i \mid t_i : T_i^\neg = \lceil \Delta_i \mid t_i : !T_i^\neg$$

by definition. On the other hand, rule LLF-PROM furnishes

$$x_1:!t_1, \dots, x_n:!T_n \vdash t : !T.$$

The induction hypothesis then shows

$$\lceil \Delta_1, \dots, \Delta_n \mid t[t_1/x_1, \dots, t_n/x_n] : !T^\neg$$

but the latter judgement is equal to

$$\lceil \Gamma_1, \dots, \Gamma_n \mid t[t_1/x_1, \dots, t_n/x_n] : T^\neg$$

as required.  $\square$

**THEOREM 12** (Soundness of algorithmic LLF).

If  $\Gamma \vdash_a s : T$  then there exists a compatible ordering of the variables for which  $\lceil \Gamma \mid s : T^\neg$  is a valid LLF judgement.

**Proof :** We proceed by induction on derivations.

**Case VAR:** We have  $s = x$  and  $\Gamma(x) = 0$ . Let  $\text{dom}(\Gamma) = \{X_n, \dots, X_1, x\}$ . We have  $\lceil \Gamma \mid x : T^\neg = \vdash \lambda X_n \dots \lambda X_1.\lambda x:|T|.x : T'$ , where:

- $\lambda X_n$  stands for  $\lambda x_n:|T_n|$  if  $X_n = x_n$  and  $\Gamma(x_n) = T_n$ , and for  $\Lambda \alpha_n$  if  $X_n = \alpha_n$ ,
- $T'$  is obtained from  $T$  by suitable construction steps of the form  $T_n \multimap (\cdot), \forall \alpha_n.(\cdot)$  and  $!(\cdot)$ .

The judgement  $\vdash \lambda X_n \dots \lambda X_1.\lambda x:|T|.x : T'$  is derivable in LLF by using rule LLF-VAR, followed by a sequence of LLF-WEAK, LLF-LAM, LLF-TLAM and LLF-PROM. The validity of the steps LLF-TLAM relies on the fact that  $\Gamma$  is a well-formed algorithmic context.

**Case APP:** It is convenient here to proceed in two steps:

1. first show that the restricted case of APP where the typing judgements for  $t_i$  ( $i = 1, 2$ ) have disjoint contexts  $\Gamma_i$  is sound;
2. then prove that the contraction rule stated in Lemma 2 also is sound.

It is then easy to check that the general APP rule is derivable from these two rules.

Let us examine step 1. We have  $s = t_1 t_2$  and  $\Gamma_1 \vdash t_1 : S \multimap T$ ,  $\Gamma_2 \vdash t_2 : S$ .

By IH we have:

$$\lceil \Gamma_1 \mid t_1 : S \multimap T^\neg$$

and

$$\lceil \Gamma_2 \mid t_2 : S^\neg.$$

On the other hand, the following judgement is derivable in LLF:

$$x : S \multimap T, y : S \vdash xy : T.$$

As  $\Gamma_1$  and  $\Gamma_2$  are compatible, by using Lemma 11 we get:

$$\lceil \Gamma_1, \Gamma_2 \mid t_1 t_2 : T^\neg,$$

which is what we wanted.

Consider now step 2. Here  $\Gamma = \Delta, x:(!S, m)$  and we have

$$\Delta, x_1:(!S, m), x_2:(!S, m) \vdash_a s_0 : T,$$

where  $s = s_0[x/x_1, x/x_2]$ . We have

$$\begin{aligned}
& \lceil \Delta, x_1:(!S, m), x_2:(!S, m) \mid s_0 : T^\neg = \\
& \lceil \Delta', x_1:(!S, 0), x_2:(!S, 0) \mid s'_0 : T'^\neg = \\
& \lceil \Delta' \mid \lambda x_1:|S|.\lambda x_2:|S|.s'_0 : !S \multimap !S \multimap T'^\neg
\end{aligned}$$

and also

$$\begin{aligned}
& \lceil \Delta, x:(!S, m) \mid s : T^\neg = \lceil \Delta', x:(!S, 0) \mid s' : T'^\neg = \\
& \lceil \Delta' \mid \lambda x:|S|.s' : !S \multimap T'^\neg
\end{aligned}$$

where  $s' = s'_0[x/x_1, x/x_2]$  and  $T'$  is obtained from  $T$  by prefixing with  $!$ 's,  $\multimap$ 's and  $\forall$ 's.

The induction hypothesis says that the following holds:

$$\lceil \Delta' \mid \lambda x_1:|S|.\lambda x_2:|S|.s'_0 : !S \multimap !S \multimap T'^\neg.$$

Now apply Lemma 11 with the following LLF-derivable judgement:

$$f:!S \multimap !S \multimap T', x:!S \vdash (\lambda y:!S.fyy)x:T'.$$

We get

$$\lceil \Delta', x:(!S, 0) \mid (\lambda y:!S.(\lambda x_1:!S.\lambda x_2:!S.s'_0)yy)x : T'^\neg$$

from which by Lemma 9 and the previous remarks we deduce:

$$\lceil \Delta', x:(!S, 0) \mid s : T'^\neg = \lceil \Gamma \mid s : T^\neg.$$

This finishes step 2 and we have thus completed this case.

**Case DIG:** we use the derivation of  $x:!U \vdash x : !!U$  in LLF and Lemma 11.

**Cases DER and TAPP:** Analogous.

**Case LAM:** We have  $T = T_1 \multimap T_2$  and  $\Gamma, x:(T_1, 0) \vdash_a t_0 : T_2$  with  $t = \lambda x:|T_1|.t_0$ . Place  $x$  before all the variables in  $\Gamma$  to obtain a compatible ordering of the variables in  $\Gamma, x:(T_1, 0)$ . The induction hypothesis gives  $\lceil \Gamma, x:(T_1, 0) \mid t_0 : T_2^\neg$  but this latter judgement equals  $\lceil \Gamma \mid t : T^\neg$  by definition.

**Case TLAM:** Here  $s = \Lambda \alpha.s_0, T = \forall \alpha.T_0$  and we have  $\Gamma, \alpha : (\star, 0) \vdash_a s_0:T_0$ . Let  $x_1, \dots, x_n$  be the term variables in  $\text{dom}(\Gamma)$  such that  $\Gamma\{x_i\} = 0$  and  $\Gamma = \Gamma_1, x_n : (T_n, 0), \dots, x_1 : (T_1, 0)$ . We know by assumption that  $\alpha \notin FTV(T_i)$  for  $i \in \{1, \dots, n\}$ .

We can take for  $\Gamma, \alpha : (\star, 0)$  an ordering of the form:  $\Gamma_1, \alpha : (\star, 0), x_n : (T_n, 0), \dots, x_1 : (T_1, 0)$ .

By IH we have:

$$\Gamma_1, \alpha : (\star, 0), x_n : (T_n, 0), \dots, x_1 : (T_1, 0) \mid s_0 : T_0^\neg,$$

so

$$\Gamma_1 \mid \Lambda \alpha. \lambda x_n \dots \lambda x_1. s_0 : \forall \alpha. (T_n \multimap \dots \multimap T_1 \multimap T_0)^\neg.$$

On the other hand, the following judgement is derivable in LLF:

$$z : \forall \alpha. (T_n \multimap \dots \multimap T_1 \multimap T_0) \vdash \lambda x_n \dots \lambda x_1. \Lambda \alpha. (z[\alpha] x_n \dots x_1) : (T_n \multimap \dots \multimap T_1 \multimap \forall \alpha. T_0).$$

Using Lemma 11 and then Lemmas 9 and 10 we get:

$$\Gamma_1 \mid \lambda x_n \dots \lambda x_1. \Lambda \alpha. s_0 : (T_n \multimap \dots \multimap T_1 \multimap \forall \alpha. T_0)^\neg,$$

which by definition is equal to

$$\Gamma \mid \Lambda \alpha. s_0 : \forall \alpha. T_0^\neg,$$

so we have proved what we wanted.

**Case ENTER:**  $\Gamma$  is of the form  $\Gamma'^{+1}, \Delta^0$  and the premise is  $\Gamma' \vdash_a t : !T$ . By Lemma 1 we know that  $FV(!T) \subseteq \text{dom}(\Gamma')$ , so w.l.o.g. we can assume that: if  $\alpha \in \text{dom}(\Delta^0)$ , then  $\alpha \notin FV(!T)$ . Let us assume for instance  $\Delta^0 = x : (S, 0), \alpha : (\star, 0)$  (the general case can anyway be handled in a similar way). By induction hypothesis we have:  $\Gamma' \mid t : !T^\neg$ , so  $\Gamma'^{+1} \mid t : T^\neg$ . Moreover one can derive in LLF  $y : !T \vdash \Lambda \alpha. \lambda x : |S|. y : \forall \alpha. S \multimap T$ . So by applying Lemma 11 we get

$$\Gamma'^{+1} \mid \Lambda \alpha. \lambda x : |S|. t : \forall \alpha. S \multimap T^\neg.$$

Hence

$$\Gamma'^{+1}, \alpha : (\star, 0), x : (S, 0) \mid t : T^\neg,$$

which is what we wanted.

**Case LEAVE** : it follows directly from the definition.  $\square$

**COROLLARY 13.** *Let  $T$  be an LLF type and  $\Gamma$  an LLF context. If  $\Gamma^0 \vdash_a s : T$  then  $\Gamma \vdash s : T$ .*

**Proof :** As  $\Gamma^0 \vdash_a s : T$ , with the the soundness Theorem 12 we get  $\Gamma^0 \mid s : T^\neg$ . Moreover, this judgement is of the form

$$\Gamma^0 \mid s : T^\neg \vdash \Lambda \alpha_k \dots \Lambda \alpha_1. \lambda x_n \dots \lambda x_1. s : \forall \alpha_k \dots \forall \alpha_1. (T_n \multimap \dots \multimap T_1 \multimap T).$$

On the other hand, the following judgement is derivable in LLF:

$$x_1 : T_1, \dots, x_n : T_n, y : \forall \alpha_k \dots \forall \alpha_1. (T_n \multimap \dots \multimap T_1 \multimap T) \vdash y[\alpha_k] \dots [\alpha_1] x_n \dots x_1 : T.$$

By applying the rule LLF-CUT on these two judgements we get:

$$x_1 : T_1, \dots, x_n : T_n \vdash$$

$$(\Lambda \alpha_k \dots \Lambda \alpha_1. \lambda x_n \dots \lambda x_1. s)[\alpha_k] \dots [\alpha_1] x_n \dots x_1 : T.$$

By using closure properties in LLF we obtain:

$$x_1 : T_1, \dots, x_n : T_n \vdash s : T. \quad \square$$

## 5. Type checking and type inference

In this section we describe the main contribution of this paper, efficient constraint-based algorithms for the type checking and type inference problems.

### 5.1 A generic !-rule

The algorithmic system is not quite ready for type checking and type decoration, in particular because of the rules dealing with the ! connective, which are not syntax-directed: ENTER, LEAVE, DER, DIG.

For this reason we will consider a new generic rule, (ALL-!), which will represent any possible sequence of applications of these four rules. If  $a \in \mathbb{N}$  we write  $!^a$  for  $\underbrace{! \dots !}_a$ .

The new rule is given in Fig. 6, where condition (\*) is as follows.

$$(*) \left\{ \begin{array}{l} T \text{ is not of the form } !T', \\ p \geq 0, q \geq 0 \\ \Gamma^{+c} \text{ ok} \\ (\Gamma^{-1} \text{ ok}) \vee (p \neq 0) \vee (q = c = 0) \end{array} \right.$$

Note that ALL-! represents in fact a family of rules, depending on the values of  $p, q$  and  $c$ . Also note that ALL-! allows weakening with an arbitrary algorithmic context  $\Delta$ , not only of the form  $\Delta^0$ .

**LEMMA 14.** *The rule ALL-! is derivable in algorithmic LLF.*

**Proof :** Consider a valid instance of ALL-! with values  $c, p, q$ . In view of the derived rule WEAK we may assume that  $\Delta$  is empty. We want to find a sequence of rules ENTER, LEAVE, DER, DIG allowing to derive it. We know that condition (\*) is satisfied and  $FV(\Gamma) = FV(t)$ . Let us examine the various possibilities:

- if  $p \neq 0$ :

As  $p \geq 0$  this means that  $p > 0$ . Let us first show that we can derive from  $\Gamma \vdash_a t : !^p T$  the judgement  $\Gamma^{+c} \vdash_a t : !^p T$ . We distinguish two subcases:

1. if  $c \geq 0$ :

By applying  $c$  times the rule DIG we get  $\Gamma \vdash_a t : !^{p+c} T$ . Then by using  $c$  times the rule ENTER we obtain  $\Gamma^{+c} \vdash_a t : !^p T$ .

2. if  $c < 0$ :

We apply in this case  $(-c)$  times rule LEAVE and  $(-c)$  times rule DER, which yields the same judgement.

Then, from judgement  $\Gamma^{+c} \vdash_a t : !^p T$ , if  $q \leq p$  (resp.  $p \leq q$ ) by repeated instances of DER (resp. DIG) we derive the judgement  $\Gamma^{+c} \vdash_a t : !^q T$ .

- otherwise, if  $(\Gamma^{-1} \text{ ok})$  holds:

We can apply an instance of the rule LEAVE and we obtain the judgement:  $\Gamma^{-1} \vdash_a t : !^{p+1} T$ . We are then in the situation of the previous case and proceed as described there.

- otherwise, if  $(q = c = 0)$  holds:

Assuming that the previous assumptions do not hold we thus have  $p = q = c = 0$  and the instance of the rule leaves the judgement unchanged.  $\square$

**LEMMA 15.** *Any sequence of rules ENTER, LEAVE, DER, DIG in algorithmic LLF can be represented by one instance of rule ALL-!.*

**Proof :** Since rule ALL-! allows weakening with arbitrary algorithmic contexts we may assume without loss of generality that none of the rules in the sequence uses weakening, i.e.,  $\Delta^0$  is empty in each case.

We then proceed by induction on the length of the sequence of rules considered.

$$\frac{\Gamma \vdash_a t : !^p T \quad FV(t) = FV(\Gamma) \quad \text{condition (*)}}{\Gamma^{+c}, \Delta \vdash_a t : !^q T} \text{(ALL-!)}$$

**Figure 6.** Generic ! rule

In the base case, that of the empty sequence, we use the instance with  $c = 0$  and  $p = q = 0$ , which is valid (both if  $p = 0$  and if  $p \neq 0$ ).

Consider a valid sequence of length  $(n + 1)$  with  $n \geq 0$ . By IH the sequence can be represented by an instance of ALL-! with certain values  $c, p, q$ . Let us then examine the various cases, depending on the nature of the  $(n + 1)$ -th rule of the sequence:

- Rule LEAVE: By applying successively rules ALL-! and LEAVE we get:

$$\frac{\Gamma \vdash_a t : !^p T \quad FV(\Gamma) = FV(t) \quad \text{condition (*)}}{\frac{\Gamma^{+c} \vdash_a t : !^q T}{\Gamma^{+c-1} \text{ ok}} \quad \Gamma^{+c-1} \text{ ok}} \Gamma^{+c-1} \vdash_a t : !^{q+1} T$$

We need to check that the corresponding instance of (ALL-!) with values  $(c - 1), p, (q + 1)$  satisfies the condition (\*). Now,  $\Gamma^{+(c-1)} \text{ ok}$  is an explicit side condition. The first premise yields  $p \geq 0, q \geq 0$ , hence in particular  $p \geq 0, q + 1 \geq 0$ . Let us consider the different cases of condition (\*) in the first premise. If  $\Gamma^{-1} \text{ ok}$  or  $p \neq 0$  then we are clearly done. So suppose that  $q = c = 0$ . But in that case, we have  $\Gamma^{-1} \text{ ok}$  and we are also done.

- Rule ENTER: By applying successively rules ALL-! and ENTER we get:

$$\frac{\Gamma \vdash_a t : !^p T \quad FV(\Gamma) = FV(t) \quad \text{condition (*)}}{\frac{\Gamma^{+c} \vdash_a t : !^q T}{\Gamma^{+c+1} \vdash_a t : !^{q-1} T} \quad q > 0}} \Gamma^{+c+1} \vdash_a t : !^{q-1} T$$

The values to satisfy (\*) are  $(c+1), p, (q-1)$ . Here,  $\Gamma^{+(c+1)} \text{ ok}$  follows from  $\Gamma^{+c} \text{ ok}$ . From  $p \geq 0, q \geq 0$  and  $q > 0$  we get hence in particular  $p \geq 0, q - 1 \geq 0$ . Now let us consider the different cases of condition (\*) in the first premise. If  $\Gamma^{-1} \text{ ok}$  or  $p \neq 0$  then we are clearly done. But the third case  $q = c = 0$  is impossible in view of  $q > 0$ .

- Rule DIG

$$\frac{\Gamma \vdash_a t : !^p T \quad FV(\Gamma) = FV(t) \quad \text{condition (*)}}{\frac{\Gamma^{+c} \vdash_a t : !^q T}{\Gamma^{+c} \vdash_a t : !^{q+1} T} \quad \text{ALL-!} \quad q > 0}} \Gamma^{+c} \vdash_a t : !^{q+1} T \text{ DIG}$$

The values to satisfy (\*) are  $(c + 1), p, (q + 1)$ . Here,  $\Gamma^{+c} \text{ ok}$  has been assumed. The rest is like the previous case ENTER.

- Rule DER

$$\frac{\Gamma \vdash_a t : !^p T \quad FV(\Gamma) = FV(t) \quad \text{condition (*)}}{\frac{\Gamma^{+c} \vdash_a t : !^q T}{\Gamma^{+c} \vdash_a t : !^{q-1} T} \quad \text{ALL-!} \quad q > 0}} \Gamma^{+c} \vdash_a t : !^{q-1} T \text{ DER}$$

Here,  $\Gamma^{+c} \text{ ok}$  is part of the assumption. Again, the rest is like case ENTER.

## 5.2 Constraints generation

Solving the type checking problem is now a matter of solving a system of arithmetical constraints which arises from backward applications of the algorithmic rules using unknowns instead of actual values for the numerical parameters contained in those rules.

### 5.2.1 Type schemas

To describe this procedure in some more detail let us introduce LLF-*type schemas* by the following grammar where  $q$  ranges over integer unknowns.

$$T ::= A \mid T_1 \multimap T_2 \mid \forall \alpha. T \mid !^q T$$

With each system F type  $A$  we can associate a type schema  $A^T$  by replacing  $\multimap$  with  $\multimap$  and inserting  $!^q$  in front of every sub-type-expression of  $A$  (choosing fresh unknowns each time). For example, if  $A$  is  $\forall \alpha. \alpha \rightarrow \alpha$  then  $A^T$  is  $!^a (\forall \alpha. !^b (!^c \alpha \multimap !^d \alpha))$  where  $a, b, c, d$  are integer unknowns. It is clear that any LLF type  $T$  with  $|T| = A$  arises from  $A^T$  by substituting nonnegative integers for the unknowns.

We gloss over the fact that  $A \mapsto A^T$  is not strictly speaking a function because it depends on the choice of the unknowns and their freshness.

Now suppose that we are given an instance  $(\Gamma, t, T)$  of the type checking problem with  $\Gamma$  an LLF-context,  $t$  a term, and  $T$  an LLF type.

### 5.2.2 Skeleton derivation

We begin by forming the type schemas  $|T|^T$  and  $|\Gamma(x)|^T$  for each  $x \in \text{dom}(\Gamma)$ . Let  $\Gamma^T$  be the schematic LLF-context given by  $\Gamma^T(x) = |\Gamma(x)|^T$  and  $\mathcal{E} = FTV(T) \cup FTV(t)$ . We now construct a schematic “skeleton” derivation of the algorithmic judgement  $(\Gamma^T + \mathcal{E})^0 \vdash_a t : |T|^T$  by backwards application of the algorithmic rules with an instance of the generic rule ALL-! before and after each of the syntax-directed rules VAR, APP, LAM, TAPP, TLAM which in turn are applied according to the structure of  $t$ . We introduce a fresh set of unknowns after/before each application of rule ALL-! and use appropriate equality constraints to relate premise and conclusion of each rule instance.

We also introduce constraints which restrict the parameters in the bottom judgement  $(\Gamma^T + \mathcal{E})^0 \vdash_a t : |T|^T$  in such a way that it corresponds to  $(\Gamma + \mathcal{E})^0 \vdash_a t : T$ . However, instead of stipulating the precise number of !-s it suffices here to distinguish between  $!^0$  and  $!^a$  for  $a \neq 0$ . This is because in LLF in view of rules DER and DIG,  $!^a A$  and  $!^b A$  for  $a \neq 0$  and  $b \neq 0$  are equivalent by rules DER, DIG and CUT.

We explain below how the side conditions (\*) translate into additional constraints on the unknowns. Rather than continuing to describe the construction of the skeleton derivation abstractly, we refer to the concrete example shown in Fig. 7. In this example, the type checking problem is  $\Gamma = x:!(B \multimap C), y:!(A \multimap B), z:!A$  and  $t = x(yz)$  and  $T = !C$ . Herein,  $A, B, C$  are base types that are neither declared in the algorithmic typing contexts nor further decomposed. We omit here for simplicity the declarations  $\alpha:(*, m)$  for type variables in the contexts since they do not play any rôle in the example.

The constraints are shown in Figure 8. They comprise nonnegativity constraints on all the unknowns appearing as “exponents” to ! and as level annotations in algorithmic LLF-contexts. The constraints marked “initial” make sure the bottom judgement has the required form (up to equivalence). The remaining ones stem from the side conditions of the generic rule ALL-! as we describe now.

### 5.2.3 Transformation of condition (\*) into arithmetic constraints

The unknowns contained in a skeleton derivation must be chosen in such a way that the side conditions (\*) arising from instances of rule ALL-! are met. Moreover, the unknowns are constrained to be integers not arbitrary rationals or even reals.

We now show how these conditions can be transformed into a tractable fragment of quantifier-free rational arithmetic and moreover that solutions to those constraints are invariant under scaling, so that satisfiability over the rationals is equivalent to solvability over the integers. This is crucial because the usual decision procedures work over the rationals rather than over the integers and in general satisfiability of arithmetic constraints over the integers is NP-hard.

The parameter  $c$  in the rule ALL-! is represented by an additional unknown not subject to a nonnegativity constraint. In the example, these additional unknowns are  $C_{\{1,\dots,5\}}$  corresponding to five rule instances.

The condition “ $T$  is not of the form ! $T'$ ” is implicit in the exponential notation for iterated !-s. The conditions “ $\Gamma^{+c}$  ok” are implicit in the nonnegativity constraints for the unknowns appearing in the conclusion of the rule.

The condition  $\Gamma^{-1}$  ok can be represented as a conjunction of inequalities of the form  $a \neq 0$  with  $a$  subject to nonnegativity. This is acceptable since all unknowns range over integers and  $\Gamma^{-1}$  ok  $\iff \Gamma^{+c}$  ok for all  $c \geq -1$ .

After distributing disjunctions over conjunctions we then end up with a polynomially-sized conjunction of disjunctions of the forms  $a \geq 0$  (nonnegativity)  $a + b = c$  (coupling of premise and conclusion) and  $a \neq 0 \vee b \neq 0 \vee c = 0$ .

Finally, we remark that the unknowns corresponding to variables that do not occur in the term can be left unconstrained since rule ALL-! allows arbitrary weakening.

Fig. 8 shows the set of constraints in the running example prior to distributing.

### 5.3 Efficient decidability of type checking

Since the resulting system of constraints does not contain nonzero constants, it is stable under scaling, i.e. multiplying a solution by a constant yields a solution. As a result, integer satisfiability is equivalent to satisfiability over the rationals.

Regarding satisfiability over the rationals we then note that the system of constraints falls into the class of “Horn-DLR” introduced and shown to be decidable in polynomial time in [14]. More precisely, a Horn-DLR is a conjunction of disjunctions where each disjunction comprises disequations ( $\neq$ ) and *up to one* inequality ( $\leq$ ). Summing up, we can therefore deduce:

**THEOREM 16** (Decidability of LLF type checking). *Let  $\Gamma$  be an LLF-context,  $t$  a term,  $T$  an LLF-type. It is decidable in polynomial time whether  $\Gamma \vdash t : T$  holds.*

**Proof :** By soundness and completeness of algorithmic type checking it is enough to decide whether  $\Gamma^0 \vdash_a t : T$ . This, however, is equivalent to the satisfiability (over the rationals by scaling) of the constraint system arising from the corresponding skeleton derivation. This in turn is possible to decide in polynomial time by the decision procedure for Horn-DLR.  $\square$

In our example, we have, however, not used the polynomial-time algorithm from [14] but rather the practical implementation Yices [8] which copes with arbitrary boolean combinations of rational inequalities, equations, and disequations (albeit without polynomial-time guarantee).

The Yices solver outputs the following solution:

	$a$	$b$	$c$	$d$	$e$	$f$	$g$	$h$	$i$	$j$	$k$	$l$	$C$
1	1	1	0	1	0	1	0	1	1	1	0	1	-2
2			2		2		2	0	0		1		2
3					1	1	1						1
4					0		0						1
5													1

we remark that although the “exponents” to ! can be restricted to 0, 1 this is not so for the unknowns appearing in context levels. Indeed, there is no integer solution of the constraint system that would set  $g_2$  to 0 or 1. As a result, it is not obviously possible to reduce type checking to propositional logic.

### 5.4 Type inference problem

Given a system F typing judgement  $\Gamma \vdash t : A$ , we note that the system of constraints arising from the skeleton derivation ending in  $(\Gamma^T + \mathcal{E})^0 \vdash_a t : A^T$  yields a concise description (polynomial-sized) of all possible LLF-annotations of that judgement, so that the constraints system may be regarded a solution of the type inference problem.

## 6. Conclusion and related work

We have solved the type checking and type inference problems for LLF, a version of intuitionistic, affine linear logic with connectives  $\multimap, !, \forall$ . Our type checking algorithm accepts a well-typed Church style system F term and decides, for an alleged typing in LLF refining the given system F typing, whether it is indeed valid. Our algorithm uses an algorithmic presentation of LLF that uses levels in contexts and is similar to an algorithmic system for the modal logic S4 [17] and a system also for linear logic developed by Martini and Masini [16]. The algorithmic system can also be seen as a judgemental version of the type inference algorithms by Baillot and Terui [1, 2]. The former was for elementary linear logic which has a somewhat simpler proof theory resulting in a constraint system without any disjunctions; the latter was for a restricted version of light linear logic with ! appearing to the left of implication only. It would be interesting to see whether the judgemental version used here could be used to simplify constraint generation in those algorithms as well.

We also mention [6] and [4] where constraint-based algorithms for type inference for elementary linear logic are given which are, however, not polynomial time. In [5] a proposal to decorate simply typed lambda-terms in intuitionistic linear logic is also described, but it does not lead to a polynomial time procedure either. The related notion of “linear decoration” was introduced and pioneered by Schellinx and his coauthors [7, 19]. Here, the task was to determine which of the !-operators introduced by Girard’s translation from intuitionistic to linear logic can be omitted. This has applications to the study of cut elimination, but does not solve the type checking problem where a particular typing is prescribed.

It should be straightforward to adapt our algorithm to the S4 type system studied in [17] since the only difference is unrestricted duplication which makes the system easier if anything. In particular, the somewhat annoying restriction on the cut rule LLF-CUT could be relieved in that case.

We remark that the type checking algorithm contained in [17] does not subsume that version since it requires usages of rules (in our notation) LEAVE, ENTER to be explicitly marked in the term whereas we can infer their placement.

It would be interesting to investigate to what extent the restriction on rule LLF-CUT can be lifted in the linear system as well. This would require the type inference to discover opportunities for sharing identical (up to  $\alpha$ -equivalence) subterms. Preliminary experiments with a representation of terms with maximal sharing look promising but are left for future work.

## Acknowledgments

The first author was partially supported by project ANR-08-BLANC-0211-01 "COMPLICE".

## References

- [1] V. Atassi, P. Baillot, and K. Terui. Verification of Prime reducibility for system F terms: Type inference in dual light affine logic. *Logical Methods in Computer Science*, 3(4), 2007.
- [2] P. Baillot and K. Terui. A Feasible Algorithm for Typing in Elementary Affine Logic. In *Proceedings of TLCA'05*, volume 3461 of *LNCS*, pages 55–70. Springer, 2005.
- [3] E. Barendsen and S. Smetsers. Uniqueness type inference. In M. V. Hermenegildo and S. D. Swierstra, editors, *PLILP*, volume 982 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 1995. ISBN 3-540-60359-X.
- [4] P. Coppola and S. Martini. Typing lambda terms in elementary logic with linear constraints. In *Proceedings of TLCA'01*, volume 2044 of *LNCS*, pages 76–90, 2001.
- [5] P. Coppola and S. Martini. Optimizing optimal reduction. a type inference algorithm for elementary affine logic. *ACM Transactions on Computational Logic*, 7(2):219–260, 2006.
- [6] P. Coppola and S. Ronchi Della Rocca. Principal typing for lambda calculus in elementary affine logic. *Fundam. Inform.*, 65(1-2):87–112, 2005.
- [7] V. Danos, J.-B. Joinet, and H. Schellinx. On the linear decoration of intuitionistic derivations. *Archive for Mathematical Logic*, 33(6), 1994.
- [8] B. Dutertre and L. de Moura. The YICES SMT Solver. Available at: [yices.csl.sri.com/tool-paper.pdf](http://yices.csl.sri.com/tool-paper.pdf).
- [9] R. Ennals, R. Sharp, and A. Mycroft. Linear types for packet processing. In D. A. Schmidt, editor, *ESOP*, volume 2986 of *Lecture Notes in Computer Science*, pages 204–218. Springer, 2004. ISBN 3-540-21313-9.
- [10] M. Fluet, G. Morrisett, and A. J. Ahmed. Linear regions are all you need. In P. Sestoft, editor, *ESOP*, volume 3924 of *Lecture Notes in Computer Science*, pages 7–21. Springer, 2006. ISBN 3-540-33095-X.
- [11] J.-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
- [12] S. Guerrini, S. Martini, and A. Masini. An analysis of (linear) exponentials based on extended sequents. *Logic Journal of the IGPL*, 6(5): 735–753, 1998.
- [13] M. Hofmann. A type system for bounded space and functional in-place update. *Nordic Journal of Computing*, 7(4):258–289, 2000.
- [14] P. Jonsson and C. Bäckström. A linear-programming approach to temporal reasoning. In *Proceedings of the 13th (US) National Conference on Artificial Intelligence (AAAI-96)*, pages 1235–1240, Portland, OR, USA, 1996.
- [15] I. Mackie. Lilac: A functional programming language based on linear logic. *J. Funct. Program.*, 4(4):395–433, 1994.
- [16] S. Martini and A. Masini. On the fine structure of the exponential rule. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic (Proc. of the Workshop on Linear Logic, Cornell University, June 1993)*, number 222. Cambridge University Press, 1995. URL [citeseer.ist.psu.edu/martini93fine.html](http://citeseer.ist.psu.edu/martini93fine.html).
- [17] F. Pfenning and H.-C. Wong. On a modal lambda calculus for S4. *Electr. Notes Theor. Comput. Sci.*, 1, 1995.
- [18] L. Roversi. A compiler from Curry-typed  $\lambda$ -terms to linear- $\lambda$ -terms. In *Theoretical Computer Science: Proceedings of the Fourth Italian Conference*, pages 330 – 344, L'Aquila (Italy), October 1992. World Scientific.
- [19] H. Schellinx. *The Noble Art of Linear Decorating*. ILLC Dissertation Series 1994-1, Institute for Language, Logic and Computation, University of Amsterdam, 1994.
- [20] P. Wadler. Is there a use for linear logic? In *Proceedings of Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'91*, volume 26(9) of *SIGPLAN Notices*, pages 255–273, 1991.
- [21] D. Walker. Substructural Type Systems. In B. C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 1. MIT Press, 2005.

$$\begin{array}{c}
\frac{x:(\uparrow^{\alpha_1}(\uparrow^{b_1} B \multimap C), e_3) \vdash_a x : \uparrow^{j_1}(\uparrow^{i_1} B \multimap \uparrow^{h_2} C)}{\text{VAR}_1} \quad \frac{x:(\uparrow^{\alpha_1}(\uparrow^{b_1} B \multimap C), e_2) \vdash_a x : \uparrow^{i_1} B \multimap \uparrow^{h_2} C}{\text{ALL}^{-1}_2} \quad \frac{y:(\uparrow^{d_1}(A \multimap B), e_4) \vdash_a y : \uparrow^{i_1}(\uparrow^{k_1} A \multimap \uparrow^{i_2} B)}{\text{VAR}_2} \quad \frac{z:(\uparrow^{f_3} A, g_4) \vdash_a z : \uparrow^{h_2} A}{\text{ALL}^{-1}_5} \\
\frac{y:(\uparrow^{d_1}(A \multimap B), e_3) \vdash_a y : \uparrow^{k_1} A \multimap \uparrow^{i_2} B}{\text{ALL}^{-1}_4} \quad \frac{y:(\uparrow^{d_1}(A \multimap B), e_3), z:(\uparrow^{f_1} A, g_3) \vdash_a yz : \uparrow^{i_2} B}{\text{APP}} \quad \frac{z:(\uparrow^{f_3} A, g_3) \vdash_a z : \uparrow^{k_1} A}{\text{APP}} \\
\frac{y:(\uparrow^{d_1}(A \multimap B), e_2), z:(\uparrow^{f_1} A, g_2) \vdash_a yz : \uparrow^{i_1} B}{\text{ALL}^{-1}_3} \quad \frac{y:(\uparrow^{\alpha_1}(A \multimap B), e_1), z:(\uparrow^{f_1} A, g_1) \vdash_a x(yz) : \uparrow^{h_2} C}{\text{APP}} \\
\frac{x:(\uparrow^{\alpha_1}(\uparrow^{b_1} B \multimap C), e_2), y:(\uparrow^{d_1}(A \multimap B), e_2), z:(\uparrow^{f_1} A, g_2) \vdash_a x(yz) : \uparrow^{h_2} C}{\text{ALL}^{-1}_1} \quad \frac{x:(\uparrow^{\alpha_1}(\uparrow^{b_1} B \multimap C), e_1), y:(\uparrow^{d_1}(A \multimap B), e_1), z:(\uparrow^{f_1} A, g_1) \vdash_a x(yz) : \uparrow^{h_1} C}{\text{APP}}
\end{array}$$

Figure 7. Example skeleton derivation

$$\begin{array}{l}
\text{Initial} \\
\text{Nonnegativity} \\
!_1 \\
!_2 \\
\text{VAR}_1 \\
!_3 \\
!_4 \\
\text{VAR}_2 \\
!_5 \\
\text{VAR}_3
\end{array}
\begin{array}{l}
c_1 = e_1 = g_1 = 0, a_1 \neq 0, b_1 \neq 0, d_1 \neq 0, f_1 \neq 0, h_1 \neq 0 \\
\{a, \dots, k\}_{\{1, \dots, 4\}} \geq 0. \\
c_1 = c_2 + C_1, e_1 = e_2 + C_1, g_1 = g_2 + C_1, (c_2 \neq 0 \wedge e_2 \neq 0 \wedge g_2 \neq 0) \vee h_2 \neq 0 \vee h_1 = C_1 = 0 \\
c_2 = c_3 + C_2, c_3 \neq 0 \vee j_1 \neq 0 \vee C_2 = 0 \\
j_1 = a_1, c_3 = 0, i_1 = b_1, h_2 = 0 \\
e_2 = e_3 + C_3, g_2 = g_3 + C_3, (e_3 \neq 0 \wedge g_3 \neq 0) \vee i_2 \neq 0 \vee i_1 = C_3 = 0 \\
e_3 = e_4 + C_4, e_4 \neq 0 \vee l_1 \neq 0 \vee C_4 = 0 \\
l_1 = d_1, e_4 = 0, k_1 = 0, i_2 = 0 \\
g_3 = g_4 + C_5, g_4 \neq 0 \vee k_2 \neq 0 \vee k_1 = C_5 = 0 \\
k_2 = f_3, g_4 = 0
\end{array}$$

Figure 8. Constraints obtained from skeleton derivation in Fig. 7