Available online at [www.sciencedirect.com](http://www.sciencedirect.com/)

[Electronic Notes in Theoretical Computer Science 332 (2017) 75–93](http://dx.doi.org/10.1016/j.entcs.2017.04.006)

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

From Cut-free Calculi to Automated Deduction: The Case of Bounded Contraction [1](#_bookmark0)

Agata Ciabattoni, Bj¨orn Lellmann[2](#_bookmark0)

*Vienna University of Technology. Austria.*

Carlos Olarte, Elaine Pimentel[3](#_bookmark0)

*Universidade Federal do Rio Grande do Norte. Brazil*

**Abstract**

The addition of the bounded contraction rules to Full Lambek Calculus with exchange and weakening (*FLew* ) gives rise to serious complications for proof search. For example, adding to *FLew* a naive version of these rules brakes cut-admissibility. Although this can be avoided by refining these rules, in this work we show that even “good” proof systems for *FLew* plus bounded contraction do not necessarily lead to good implementations. In order to solve this problem, we propose an extension of the lazy splitting methodology to bounded contractions, showing how to transform our focused, cut-free sequent calculus into a terminating theorem prover. Our system is used to show that the decision problem for *FLew* with bounded contraction is in EXPTIME.

*Keywords:* Proof theory, substructural logics, proof search, bounded contraction.

# Introduction

Cut-free sequent calculi are undoubtedly among the finest tools for proving impor- tant theoretical properties such as consistency, decidability or interpolation for a given logic. Unfortunately, when it comes to practical applications, and in particular to automated reasoning tasks, they are not always well-behaved. Seemingly innocu- ous rules, like contraction, give rise to a dramatic increase of the search space in backward proof search procedures. Indeed, a major part of the proof-theoretic effort

1 This work has been supported by the Marie Curie project GetFun (PIRSES-GA-2012-318986 funded by EU-FP7), the FWF START project Y544-N23, EU H2020-MSCA grant 660047, CAPES and CNPq.

2 Email: [*{*agata,lellmann*}*@logic.at](mailto:agata@logic.at)

3 Email: *{*carlos.olarte, elaine.pimentel*}*@gmail.com

<http://dx.doi.org/10.1016/j.entcs.2017.04.006>

1571-0661/© 2017 The Author(s). Published by Elsevier B.V.

This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

for proposing good logical proof systems suitable for implementation involves tam- ing exactly this rule. For example, good refinements of sequent systems for classical logic absorb contraction into the logical rules; in intuitionistic logic, contraction can be avoided after a careful control on the use of the implication left rule [[5](#_bookmark27)]; and all focused systems, e.g. for linear logic, rely on controlling the duplication of classical resources.

Here we consider weaker variants of the contraction rule, the so-called *bounded contraction* rules, which for a fixed *k*, contract *k* +1 copies of a formula into *k* copies. These rules are special cases of knotted rules [[10](#_bookmark32)] and play an important role, e.g., in finite-valued L- ukasiewic logics [[15](#_bookmark37)] or varieties of residuated lattices [[8](#_bookmark30)].

Simply adding bounded contraction rules to the sequent calculus for affine in- tuitionistic additive, multiplicative linear logic aIMALL [[6](#_bookmark28)] (or, equivalently, Full Lambek Calculus with exchange and weakening *FLew*) brakes cut-admissibility (al-

though not consistency). Equivalent versions of the bounded contraction rules, which preserve cut-elimination when added to *FLew*, have been defined, e.g., in [[2](#_bookmark23)]. However, although the resulting cut-free systems might suffice for theoretical pur-

poses, the additional rules involve splitting the context into a number of parts. Since there are many ways to do so, this results in an exponential branching in the naive proof search procedure, rendering it unfeasible for practical purposes.

In this paper we strive to solve this problem and show how to turn the cut-free sequent calculus for *FLew* with bounded contraction into a terminating theorem prover. For this purpose, we propose a general form of *lazy splitting* which avoids the

splitting of contexts, hence solving the problem of the exponential choice inherent in the rules of bounded contraction. This choice is substituted by a much simpler one: from which context, on the leaves, an atomic formula should be taken. This not only turns bounded contractions into harmless structural rules, but also provides the basis for efficient theorem provers for a number of logics. An implementation of the system, available at <http://subsell.logic.at/flew>, is also presented. We

further show an EXPTIME upper bound on the complexity of *FLew* with bounded

contractions. While a PSPACE lower bound for these logics follows from [[11](#_bookmark33)], their

exact complexity is left open.

The paper is structured as follows: The base logical system *FLew* with bounded contraction is presented in Section [2](#_bookmark1), while its focused, cut-free version is described in Section [3](#_bookmark6). In Section [4](#_bookmark9) we show that the proposed systems yield the seemingly

first purely proof-theoretic proof of decidability and complexity for *FLk*

*ew*

. Section [5](#_bookmark13)

proposes the general lazy splitting procedure. Finally, we conclude and present

some future work in Section [6.](#_bookmark22)

# *FLew* with bounded contraction

Providing feasible *automated deduction procedures* for substructural logics, i.e., log- ics whose sequent calculi lack (or restrict the use of) some of the standard structural rules, is a difficult task. A successful example of such a procedure is the one avail- able under <http://www1.chapman.edu/~jipsen/reslat/>, which implements the

Δ*, Ak*+1 *▶ C*

Δ*, Ak ▶ C* (k + 1) *−* k Δ*,A ▶ A* init Δ *▶T T* Δ*, ⊥▶ C ⊥*

Δ *▶ A* Γ*,A ▶ C*

Δ*,* Γ *▶ C* cut

Δ*, A, B ▶ C* Δ1 *▶ A* Δ2 *▶ B* Δ1 *▶ A* Δ2*, B ▶ C*

*⊗ ⊗*

*L R*

*⊃L*  Δ*, A ▶ B ⊃R*

Δ*,A ⊗ B ▶ C*

Δ1*,* Δ2 *▶ A ⊗ B*

Δ1*,* Δ2*,A ⊃ B ▶ C*

Δ *▶ A ⊃ B*

Δ*, Ai ▶ C* Δ *▶ A* Δ *▶ B* Δ*, A ▶ C* Δ*, B ▶ C*

*∧ ∧*

*Li R*

*∨L*  Δ *▶ Ai ∨Ri*

Δ*, A*1 *∧ A*2 *▶ C*

Δ *▶ A ∧ B*

Δ*,A ∨ B ▶ C*

Δ *▶ A*1 *∨ A*2

Fig. 1. Naive system for *FLew* with bounded contraction (note that the weakening rule is admissible).

algorithm in [[14](#_bookmark36)] to decide validity of equations in residuated lattices (and hence, of formulas in Full Lambek Calculus).

In this work we provide automated deduction procedures for substructural logics obtained by adding to Full Lambek calculus with exchange and weakening (*FLew*, or aIMALL) the axioms *αk ⊃ αk*+1 (where *αn* stands for *α ⊗ · · ·⊗ α*, *n* times,

*α*0 = *T*) [4](#_bookmark4) , for *k ≥* 1, or, equivalently, the rules [[10,](#_bookmark32)[15]](#_bookmark37)

Δ*, Ak*+1 *▶ C*

(k + 1) *−* k

Δ*, Ak ▶ C*

More precisely, we consider formulas built from the grammar

*F* ::= *T |⊥| A | F*1 *⊗ F*2 *| F*1 *∨ F*2 *| F*1 *⊃ F*2 *| F*1 *∧ F*2

and the rules in Fig. [1](#_bookmark2) for *FLew* plus bounded contraction.

As shown in [[10](#_bookmark32)] such systems enjoy cut elimination if and only if *k* = 1. Consider for instance the sequent

*C, A ⊃ C ⊃ B,* (*A ⊃ B*)*k−*1*,* (*A ⊃ B*)*k*+1 *⊃ D ▶ D k >* 1 (1)

A proof of such sequent with cut in the naive system of Fig.[1](#_bookmark2) is

*A ▶ A* init

*C ▶ C* init *B ▶ B* init

*C, C ⊃ B ▶ B ⊃L*

*D ▶* init

*⊃ ⊗*

*D*

*C, A ⊃ C ⊃ B, A ▶ B ⊃L*

*C, A ⊃ C ⊃ B ▶ A ⊃ B ⊃R*

(*A ⊃ B*)*k*+1*,* (*A ⊃ B*)*k*+1 *⊃ D ▶ D*

(*A ⊃ B*)*k,* (*A ⊃ B*)*k*+1 *⊃ D ▶ D* (k + 1) *−* k

*C, A ⊃ C ⊃ B,* (*A ⊃ B*)*k−*1*,* (*A ⊃ B*)*k*+1 *⊃ D ▶ D* cut

It is straightforward to check that this cut cannot be eliminated. Indeed, any cut-free proof of the end sequent would start with an instance of the implication left rule. Using polarities and focusing techniques, we can show that neither the formula *A ⊃ C ⊃ B* nor the formula *A ⊃ B* can be principal. Moreover, applying implication left repeatedly on (*A ⊃ B*)*k*+1 *⊃ D* will consume the rest of the context and the remaining sequent (*A ⊃ B*) *⊃ D ▶ D* is not provable. Observe also that the (k + 1) *−* k rule cannot be used in any moment.

This counter-example is based on the fact that some information is hidden inside the implication. However, there are some other less trivial counter-examples, for

4 Abusing the notation, we will represent by *Ak* both *A* N*k*

*i*=1

*A* and *A,..., A*, *k* times. Observe that the

rule *⊗L* is invertible, hence it is always possible to substitute tensors by commas (and vice-versa).

instance,

*A ⊃ ⊥,* (*A ⊃ B*)*k—*1 *▶* (*A ⊃ B*)*k*+1 (2)

Observe that *A ⊃⊥* implies *A ⊃ B* and then, *A ⊃ B* is represented in the context *k* times. However, the rule (k + 1) *−* k cannot be applied to multisets of formulas and the only way of proving the sequent ([2](#_bookmark5)) is by applying the cut rule which, again, cannot be eliminated.

There is a number of ways of proposing less naive sequent systems for handling bounded contractions. For instance, the algorithm in [[2](#_bookmark23)], transforms (suitable) axioms into structural rules and leads to the following rule for bounded contraction

Δ*,* Δ*k*+1 *▶ C ...* Δ*,* Δ*i*1 *,...,* Δ*ik ▶ C . . .* Δ*,* Δ*k*+1 *▶ C*

1 1 *k*

*k* k*j*

Δ*,* Δ1*,...,* Δ*k ▶ C*

where Δ1*,...,* Δ*k* are non-empty multisets of formulas, *i*1 + *···* + *ik* = *k* + 1 and Δ*k* = Δ*, ...,* Δ, *k* times. Adding these rules to *FLew* preserves cut-elimination. However, they are not good for proof search due to the non-trivial splitting of the

context and the large number of premises.

The following rule reduces the number of branching while preserving cut- elimination

Δ*,* Δ*k*+1 *▶ C ...* Δ*,* Δ*k*+1 *▶ C*

1 *k*

k

Δ*,* Δ1*,...,* Δ*k ▶ C*

**Lemma 2.1** *The rules* k *and* k*j are equivalent in FLew in the presence of the cut rule.*

**Proof.** Clearly k*j* derives k. For the converse direction it is enough to show that

each Δ*,* Δ*i*1 *,...,* Δ*ik ▶ C* (with *i*1 + *···* + *ik* = *k* + 1) can be derived from the

1 *k* W*k*

N *k*+1

premises of k. This follows by cutting the sequent Δ*,*

*i*=1(

Δ*i* ) *▶ C*, with

Δ*i*1 *,...,* Δ*ik ▶* W*k* (N Δ*k*+1) which is derivable in *FLew* extended with k. *2*

1 *k i*=1 *i*

Using the rule k, the sequents ([1](#_bookmark3)) and ([2](#_bookmark5)) are cut-free provable. In ([1](#_bookmark3)), take Δ1 = *{C, A ⊃ C ⊃ B},* Δ2 = *...* = Δ*k* = *{A ⊃ B}* and Δ = (*A ⊃ B*)*k*+1 *⊃ D*; in

([2](#_bookmark5)), take Δ1 = *{A ⊃ ⊥}*, Δ2 = *...* = Δ*k* = *{A ⊃ B}* and Δ = *∅*.

Observe that, when using the k rule, there is an exponential number of choices since one has to cleverly split the context into *k* + 1 parts. Hence the difficulty of finding an arbitrary cut-formula is replaced by a hard context splitting.

It is instructive to consider the relationship between the naive system in Fig. [1](#_bookmark2) and the system obtained by replacing the (k + 1) *−* k rule with the k rule. Assume that we use *k* instances of cut to produce *k* copies of the formula *A*, which will be used for applying the (k + 1) *−* k rule:

Δ*k ▶ A*

*Ak*+1*,* Δ *▶ C*

*Ak,* Δ *▶ C* (k + 1) *−* k

cut

*Ak—*1*,* Δ*,* Δ*k ▶ C*

.

Δ1 *▶ A A,* Δ*,* Δ2*,...,* Δ*k ▶ C*

Δ*,* Δ1*,...,* Δ*k ▶ C*

cut

Observe that choosing the cut formula (i.e., *A*) is the same as choosing the multisets Δ1*,...,* Δ*k* that imply *A*. It turns out that we can choose, instead of a

random *A*, the formula W*k* (N Δ*i*) as the cut formula. Indeed, in the presence

*i*=1

W (

W (

of (k + 1) *−* k, we have the equivalence proof above can be re-written as

*k i*=1

N Δ*i*)*k ≡* (

*k i*=1

N Δ*i*))*k*. Hence the

(Δ1)*k*+1*,* Δ *▶ C . . .* (Δ*k*)*k*+1*,* Δ *▶ C*

*∨L*

W*k*

(Δ )*k*+1*,* Δ *▶ C*

*i*=1

(W*k*

*i ≡*

(Δ*i*))*k*+1*,* Δ *▶ C*

W N *∨R,* init

(W

*k*

*i*=1 (k + 1) *−* k

W*k* N

*∨R,* init

Δ*k ▶*

*i*=1(

Δ*i*)

(W

W*k*

*k i*=1

*k i*=1

(Δ*i*))*k—*1*,* Δ*,* Δ*k ▶ C*

.

(Δ*i*))*k,* Δ *▶ C*

cut

Δ1 *▶*

*i*=1(

Δ*i*)

*i*=1(Δ*i*)*,* Δ*,* Δ2*,...,* Δ*k ▶ C* cut

Δ*,* Δ1*,...,* Δ*k ▶ C*

showing that the rule k is derivable in the naive system. Conversely, observe that

(k + 1) *−* k is an instance of k with Δ1 = *...* = Δ*k* = *A*. Hence, in this case, finding a cut-formula or splitting the context have the same level of difficulty.

**Remark 2.2** This debunks the myth that cut-free systems are necessarily well behaved. Sometimes, choosing a cut formula can be as hard as applying a rule.

# A focused, cut-free system for bounded contractions

We start by observing that, once a formula is contracted using the k rule, it can be contracted an infinite number of times. Hence, we shall consider two different kind of contexts: the *unbounded* context, where formulas can be weakened and contracted, and the *bounded* one, where formulas can be only weakened.

The rules of the proposed cut-free system *FLk*

*ew*

are presented in Figure [2](#_bookmark7). The

connectives *⊗, ∨* are called *positive*, while *⊃, ∧* are called *negative*. The *sequents* in

*k* have the shape

*FL*

*ew*

Γ; Δ *▶ C*

where Γ is the *unbounded context* and Δ is the *bounded* (or linear) one.

We now move in the direction of proposing a notion of *focusing* [[1](#_bookmark24)] for systems with bounded contractions. We start by analyzing the possible orderings of applica- tions of rules. The goal is to organize proofs in order to reduce the non determinism, as in [[13](#_bookmark35)].

**Definition 3.1** Let *S* be a sequent with two formulas *A* and *B* such that the rule *α* (resp. *β*) can be applied on *A* (resp. *B*). We say that a rule *β permutes down α*, notation *β ↓ α*, if for any proof *π* of *S* starting with *α* followed immediately by *β* (reading proofs bottom up), there exists a proof *πj* of *S* where the two rules have been exchanged (considering also the empty case). We write *β ‡ α* when *β ↓ α* and *α ↓ β*.

Given two sets of inference rules *A* and *B*, we say that *A* has *weak permutability*

if, for any two rules *α*, *αj* of *A*, *α ‡ αj*. We say that *A* has *full permutability over*

## Negative Rules

*T ⊥*

Γ; Δ*, F, G ▶ C*

*L*

*⊗*

Γ; Δ *▶T*

*R* Γ; Δ*, ⊥▶ C L*

Γ; Δ*,F ⊗ G ▶ C*

Γ; Δ*, F ▶ G ⊃R* Γ; Δ *▶ F* Γ; Δ *▶ G ∧* Γ; Δ*, F ▶ C* Γ; Δ*, G ▶ C ∨*

*R*

*L*

Γ; Δ *▶ F ⊃ G*

Γ; Δ *▶ F ∧ G*

Γ; Δ*,F ∨ G ▶ C*

Γ*, F, G*; Δ *▶ C*

&

& *∈ {∧, ⊗}* Γ*,F* ; Δ *▶ C* Γ*, G*; Δ *▶ C*

*∨LC*

Γ*,F* & *G*;Δ *▶ C L*

Γ*,F ∨ G*;Δ *▶ C*

Γ*, G*; Δ *▶ C ⊃LCC*  Γ*, G*; Δ *▶ C ⊃LCG*

Γ*,F ⊃ G, F* ;Δ *▶ C* Γ*,F ⊃ G, G*;Δ *▶ C*

## Positive Rules

Γ; Δ1 *▶ F* Γ; Δ2 *▶ G ⊗R* Γ; Δ1 *▶ F* Γ; Δ2*, G ▶ C ⊃L*

Γ; Δ1*,* Δ2 *▶ F ⊗ G* Γ; Δ1*,* Δ2*,F ⊃ G ▶ C*

Γ*, F ⊃ G*; Δ1 *▶ F* Γ*, F ⊃ G*; Δ2*, G ▶ C ⊃LC*  Γ; Δ*, Fi ▶ C ∧*  Γ; Δ *▶ Fi ∨*

*Li Ri*

Γ*,F ⊃ G*; Δ1*,* Δ2 *▶ C*

Γ; Δ*, F*1 *∧ F*2 *▶ C*

## Structural Rules

Γ; Δ *▶ F*1 *∨ F*2

Γ; Δ *▶ p*

init *p ∈ {*Δ*,* Γ*}* Γ*,* Γ1; Δ *▶ C . . .* Γ*,* Γ*k*; Δ *▶ C*

Γ; Δ*,* Γ1*,...,* Γ*k ▶ C*

k

Fig. 2. System *FLk* . In the init axiom, *p* is atomic. We note that rules *⊃LCC* and *⊃LCG*, as well as &*L* are

*ew*

admissible. However, these rules are fundamental for showing that the proof search procedure terminates (see Section [4](#_bookmark9)).

*B* when *A* has weak permutability and, for any pair of rules (*α, β*) *∈A × B*, *α ↓ β*

holds.

The following proposition identifies the classes of rules that have weak or full

permutability in *FLk* .

*ew*

**Proposition 3.2** *Let N be the set of inference rules {TR, ⊥L, ⊗L, ⊃R*

*, ∧R, ∨L,* &*L, ∨LC, ⊃LCC, ⊃LCG} and P be the set of inference rules {⊗R, ⊃L, ⊃LC*

*, ∧Li, ∨Ri}. Then, (1) N has full permutability over P and* k*, (2)* k *has full per- mutability over P and (3) P has weak permutability.*

**Proof.** The proof is straightforward by doing small step permutations or using well known equivalences in intuitionistic logic. We will present some cases of counter- examples for non-permutability. The list with all the cases can be found in the appendix of [[4](#_bookmark26)].

* *∧L /↓* k:

*A ⊃ D*; *Ak*+1 *▶ Dk*+1 *∨ Ck*+1

*A ⊃ D*; *Ak*+1 *∧ Bk*+1 *▶ Dk*+1 *∨ Ck*+1 *∧L*

*B ⊃ C*; *Bk*+1 *▶ Dk*+1 *∨ Ck*+1

*B ⊃ C*; *Ak*+1 *∧ Bk*+1 *▶ Dk*+1 *∨ Ck*+1 *∧L*

*Ak*+1 *∧ Bk*+1*,A ⊃ D, B ⊃ C ▶ Dk*+1 *∨ Ck*+1 k

* k */↓ ∧R*:

*A*; *·▶ Ak*+1 *A ∧ B*; *·▶ Ak*+1

*A, A ∧ B ▶ Ak*+1 k

*A ∧ B ▶ B ∧*2*L,* init *A ▶ A* init

*A, A ∧ B ▶ B ⊗ A ⊗R*

*A, A ∧ B ▶ Ak*+1 *∧* (*B ⊗ A*) *∧R*

*2*

We can hence separate the proof steps into two phases. In the *negative phase* all negative rules are applied eagerly until the left context has only atomic or negative formulas and the succedent formula is positive or atomic. When this process termi- nates, we may possibly apply k a number of times and the *positive phase* starts by choosing a negative formula on the left or a positive one on the right. The focus- ing persists until a positive formula appears on the left context or a negative one appears in the succedent, and we come back to the negative phase.

This determines a focused system for *FLk*

*ew*

*ew*

, called *FFLk*

(see Figure [3](#_bookmark12)). We

separate the left context of sequents in *FFLk*

*ew*

in three: the set Γ will always denote

the unbounded context; Δ is a linear context containing only negative or atomic

formulas; and Θ is a general linear context. We will differentiate focused and unfocused sequents by using different arrow symbols: “*⇒*” for unfocused and “*→*”

for focused. In this way, *FFLk*

*ew*

contains three types of sequents: Γ; Δ; Θ *⇒ G* is an

unfocused sequent; Γ; Δ *→* [*F* ] is a sequent focused on the right; and Γ; Δ*,* [*F* ] *→ G*

is a sequent focused on the left.

The following result is a direct consequence of Proposition [3.2](#_bookmark8).

**Theorem 3.3** *The system FFLk is sound and complete w.r.t. FLk .*

*ew*

*ew*

# Decidability and Complexity

The unfocused and focused systems above give rise to the seemingly first purely

proof-theoretic proof of decidability and complexity for *FLk*

*ew*

(a proof of the fi-

nite model property is in [[7](#_bookmark29)]). Decidability in both systems follows from standard

backwards proof search with a history mechanism to avoid loops.

**Theorem 4.1** *Derivability of sequents in the systems FLk*

*ew*

*and FFLk*

*is decidable.*

**Proof.** By performing a standard backward proof search, storing every sequent encountered in the current branch of the attempted derivation in a *history*, i.e., a sequence of sequents, and only applying rules if none of their premises is in the history. Since all the rules have the subformula property, the number of sequents possibly appearing in a derivation is finite, and hence the proof search terminates.*2*

*ew*

The complexity bounds provided by the previous theorem are however far from optimal. This is because, in the worst case, the procedure needs to visit every possible sequent on a single branch, giving an exponential space bound on naive proof search. This can be improved by considering *forward* instead of backwards proof search as follows.

**Negative Phase**

*⊥ T*  Γ; Δ; Θ*, F, G ⇒ C ⊗*  Γ; Δ; Θ*, F ⇒ G ⊃R*

*L*

Γ; Δ; Θ*, ⊥⇒ C*

*L* Γ; Δ; Θ *⇒T R*

Γ; Δ; Θ*,F ⊗ G ⇒ C*

Γ; Δ; Θ *⇒ F ⊃ G*

Γ; Δ; Θ *⇒ F* Γ; Δ; Θ *⇒ G ∧R* Γ; Δ; Θ*, F ⇒ C* Γ; Δ; Θ*, G ⇒ C ∨L*

Γ; Δ; Θ *⇒ F ∧ G* Γ; Δ; Θ*,F ∨ G ⇒ C*

Γ*, F, G*; Δ; Θ *⇒ C*

Γ*,F* & *G*; Δ;Θ *⇒ C* &*L*

& *∈ {∧, ⊗}* Γ*,F* ; Δ; Θ *⇒ C* Γ*, G*; Δ; Θ *⇒ C*

Γ*,F ∨ G*; Δ;Θ *⇒ C*

*∨LC*

Γ*, G*; Δ; Θ *⇒ C ⊃LCC*  Γ*, G*; Δ; Θ *⇒ C ⊃LCG*

Γ*,F ⊃ G, F* ; Δ;Θ *⇒ C* Γ*,F ⊃ G, G*; Δ;Θ *⇒ C*

**Positive Phase**

Γ; Δ1 *→* [*F* ] Γ; Δ2 *→* [*G*] *⊗*

Γ; Δ1*,* Δ2 *→* [*F ⊗ G*] *R*

Γ; Δ1 *→* [*F* ] Γ; Δ2*,* [*G*] *→ C ⊃*

Γ; Δ*,* [*Fi*] *→ C ∧*

Γ; Δ *→* [*Fi*] *∨*

Γ; Δ *,* Δ *,* [*F ⊃ G*] *→ C L* Γ; Δ*,* [*F ∧ F* ] *→ C Li* Γ; Δ *→* [*F ∨ F* ] *Ri*

1 2 1 2 1 2

**Structural Rules**

Γ*,* Δ1; Δ; *·⇒ Pa .. .* Γ*,* Δ*k*; Δ; *·⇒ Pa*

Γ; Δ*,* Δ1*,...,* Δ*k*; *·⇒ Pa* k

Γ; Δ*,* [*N* ] *→ Pa*

Γ; *N,* Δ; *·⇒ Pa Dl*

*Na,* Γ; Δ*,* [*Na*] *→ Pa*

*Na,* Γ; Δ; *·⇒ Pa Dlca*

Γ; Δ *→* [*P* ]

Γ; Δ; *·⇒ P Dr*

Γ; Δ; *P ⇒ Pa R*

Γ; Δ*,* [*P* ] *→ Pa l*

Γ; Δ; *·⇒ N*

Γ; Δ *→* [*N* ] *Rr*

Γ; Δ*, Na*; Θ *⇒ R* store

Γ; Δ; Θ*, Na ⇒ R*

Γ; [*A* ] *→ A Il* Γ; Δ *→* [*A* ] *Ir* given *Ap ∈* Γ *∪* Δ and Δ *⊆ {Ap}*

*n n p*

Fig. 3. Focused system *FFLk*

*ew*

allowing bounded contractions. Here *Pa* is positive or atomic, *P* is positive,

*N* is negative, *Na* is an implication or a negative atom, *An* a negative atom and *Ap* a positive atom.

**Theorem 4.2** *The problem of deciding whether a formula is a theorem of FLk is*

*ew*

*in* EXPTIME*.*

**Proof.** All the rules of *FLk*

*ew*

have the subformula property, hence only sequents

containing subformulae of the input formula can occur in a derivation of that for-

mula. For an input formula of size *n* the first component of such a sequent is a set of subformulae of the input, the second component is a multiset containing w.l.o.g. at most *k −* 1 copies of each subformula of the input (in case there are *k* copies of a formula we may apply the *k*-rule to transfer all of them into the classical context),

and the right hand side is a single subformula of the input. So there are at most 2*n · kn · n* relevant sequents, each containing at most *n* + *k · n* + 1 formulae, so of size at most *n·* (*n* + *k · n* + 1). To check derivability, we implement a *forward search* pro- cedure: given an input formula *A* construct the set S*A* of relevant sequents as above. Then starting from the initial sequents in S*A* apply all possible rules to the already constructed sequents and add the resulting sequents. After at most card(S*A*)-many steps this reaches a fixpoint. Now check whether the sequent *.*; *. ▶ A* is among the

constructed ones. There is a fixed number of rules in *FLk*

*ew*

, every such rule has at

most *k* premises, and computing the conclusion of such a rule can be done easily,

thus each of the steps of the construction can be done in time exponential in *n*. Since there are only exponentially many such steps, the whole procedure runs in exponential time. *2*

While this gives an upper bound for the complexity of *FLk*

*ew*

, a PSPACE lower

bound follows from the results in [[11](#_bookmark33)]. The exact complexity of *FLk*

*ew*

still seems to

be open.

# Lazy Splitting

In what we have presented so far, proof theory machinery was used in order to present a “good proof system” for bounded contraction, in the sense that it is cut free and it has a notion of normal forms (via focusing). However, the proof systems proposed in the previous sections are still far from being suitable for implementation. In fact, once a negative phase finishes, one has to decide either to apply the k rule or a positive rule. But this entails a huge problem for proof search: the logical context can be split in an exponential number of ways due to the rules k, *⊗R, ⊃L* and *⊃LC*. Observe that the focusing strategy shows that *if* a sequent is provable and *if* the proof uses the k rule, *then* you may apply it before the positive rules. But if this is not the case, then there will be useless attempts of splitting the context,

and this is extremely inefficient. Thus, a naive implementation of *FFLk*

*ew*

does not

work even for simple sequents, as shown in Section [5.1](#_bookmark19).

In linear logic, *lazy splitting* systems have been proposed in order to minimize the non-determinism during proof search (see e.g., [[9,](#_bookmark31)[12,](#_bookmark34)[3](#_bookmark25)]). The idea is to separate the linear context into two: the formulas that will/will not be used in a branch of a derivation. In this way, one avoids the splitting of contexts: a branch of a derivation goes up with all the resources, consumes what is needed and then allows the “re-use” of the spare resources in the other branch. Hence the name *lazy*, as one postpones the decision of splitting, thus improving proof search.

Formally, in the system proposed in [[12](#_bookmark34)], the rule *⊗R* becomes:

Γ; *·*;Δ :: *E ▶ F /* Δ*j* :: *Ej* Γ; Δ*j*; *Ej ▶ G / Ejj*

Γ; Δ; *E ▶ F ⊗ G / Ejj ⊗R*

where Γ represents the classical context and Δ the linear one. Moreover, *E* is a multiset of formulas representing an *excess* of resources (the *input*) that the sequent

may *return* as an *output*. The notation Ψ *▶ F /* Δ :: *E* means that, in order to prove *F* from the context Ψ, the formulas in the list of multisets Δ :: *E* were not used (*excess*). The distinction between Δ (*non-returnable*) and *E* (*returnable*) is used to define the scope of the formulas. This can be better understood with the analysis of the right rule for linear implication

Γ; Δ*,F* ; *E ▶ G /Ej*

Γ; Δ; *E ▶ F −◦ G /Ej −◦R*

Observe that the scope of the formula *F* is the premise sequent. Hence, *F* cannot be returned in the conclusion (via *Ej*) and it must be placed into the non-returnable context Δ.

The same techniques can be applied in *FLk*

*ew*

for the *linear logic connectives*, but

the rule k still needs to decide how to split the context. Hence, even for “simple”

provable sequents with few extra hypotheses (that may be weakened), a prover using focusing and lazy splitting on linear connectives is still not practical as shown in Section [5.1](#_bookmark19).

In the following we will present a new and non-trivial extension of lazy splitting to bounded contractions. More interestingly, we shall show that the lazy version of k can be *eagerly* applied without loosing provability. Hence, the results in Proposition

[3.2](#_bookmark8) (k has full permutability over *P*) lead, in fact, to a perfect implementation strategy.

Informally, the idea is to separate the classical context into 3 contexts (that should be thought as classical “wannabe” contexts), indicating the number of times that a formula was used in a derivation: none, once or many. If a formula was not used at all in an application of k, it is a *returnable excess* or *output*. If it was used exactly once, it is a *non-returnable excess*, that is, it is linear: this formula will be copied to the other branches of k but it will not be returned as an output. And if it was used 2 or more times, it is *classical* and it cannot be used in the other branches of k. The laziness for bounded contractions thus comes in *k* steps, where formulas are allowed to move from one context to a “more classical” one.

*ew*

The rules for *LFLk*

*ew*

, the lazy system for *FLk*

, are depicted in Figure [4](#_bookmark16) and

explained in the following.

Sequents in *LFLk*

*ew*

have the shape

*L* ; Δ; *E ▶ G / Lj* ; *Ej*

where *G* is a formula, *E, Ej* are lists of multisets of formulas, Δ is a mul- tiset of formulas and *L, Lj* are (possibly empty) lists of triples of the form Γ ; ΩΔ :: Ω*E* ; *N*Δ :: *NE* , where the subscripts Δ and *E* refer to the returnable

/non returnable part of the contexts. When Δ*,E* are not important, we simply write Ω*,N* instead of ΩΔ :: Ω*E* and *N*Δ :: *NE*. Moreover, when Γ*,* Ω and *N* are not important we write [*A*] . [5](#_bookmark14)

5 A useful mnemonic. We use capital letters to denote (Γ)lassical, used (Ω)nce, (*N* )ot used, (*E*)xcess

The context Γ ; Ω ; is classical but each part obeys different rules. After the

*N*

application of the lazy version of k:

* Γ contains all the formulas that have been used two or more times.
* The context Ω contains the formulas that were used only once.
* *N* contains all the formulas that have not been used, that is, the *output*.

The contexts Δ and *E* follow the design principles for the lazy system for linear logic in [[12](#_bookmark34)]. The next example clarifies better the use of the new contexts.

**Example 5.1** Here we will consider the system *LFL*2

*ew*

(that is, with *k* = 2). For

proving the sequent *a*2*,b ▶ a*3 *⊗ b*, we start by applying the lazy version of the k

rule:

*π*1

|  |  |  |
| --- | --- | --- |
|  | :: *·* ; *·* ; | *a*2*,b* :: *·* |
| *·* ; *·* ; *·* |

|  |  |  |
| --- | --- | --- |
|  | :: Γ1 ; Ω1 ; | *N* 1 |
| *·* ; *·* ; *·* |
|  |

; *·*; *·▶ a*3 *⊗ b / *  

; *a*2*, b*; *·▶ a*3 *⊗ b /*

*·* ; *·* ; *·*

; *·* Υ2

k

*·* ; *·* ; *·*

; *·*

As usual, the “*/*[*A*]; *E*” part (the output) is computed by rules in a top-down fashion. Hence, Γ1*,* Ω1*,N* 1 will be determined once *π*1 is finished. Since *E* and are empty in the conclusion, all the returned output will also be empty. Observe that, in the left premise, we assume that the whole linear context *{a*2*, b}* is going to be used as part of the bounded contraction ( *N* context). The proof *π*1 continues

*L*

as follows (omitting the “ ::” part):

*·* ; *·* ; *·*

*j*

*π*

1

*·* ; *a* :: *·* ;

*·* ; *a* :: *·* ;

; *·*; *·▶ a*2 *⊗ b /*

; *·*

; *·*; *·▶ a*3 *⊗ b /* ; *·*

Γ1 ; Ω1 ; *N* 1

*a*2*,b* :: *·*

*·* ; *·* ;

*a, b* :: *·*

; *·*

*init*

*a, b* :: *·*

; *·*; *·▶ a /*

*a*2*,b* :: *·*

*·* ; *·* ;

*N* 1

Γ1 ; Ω1 ;

*⊗R*

Note that, in the left derivation, one *a* is used in the initial axiom and hence it is moved to the Ω context in the output. The other formulas *a, b* are set in the *N* context and both Ω*,N* are passed to the right premise in the *⊗R* rule – they are the excess.

Derivation *πj*

1

proceeds as follows:

*⊗R*

*·* ; *a* :: *·* ;

Ψ1 Ψ2

*I*

; *·*; *·▶ a / a* ; *·* ; *a, b* :: *·* ; *·* ; *·*; *·▶ a ⊗ b /* Γ1 ; Ω1 ; ; *·*

; *·*; *·▶ a*2 *⊗ b /* Γ1 ; Ω1 ; ; *·*

*a, b* :: *·*

*·* ; *a* :: *·* ;

*N* 1

*a, b* :: *·*

*a* ; *·* ;

*a, b* :: *·*

*N* 1

*⊗R*

where

Ψ1 =

*I*

; *·* Ψ2 =

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ; *·* ; | *a, b* :: *·* | ; *·*; *·▶ a /* |  | ; *·* ; | *a, b* :: *·* |
| *a* :: *·* | *a* :: *·* |

*I*

; ; ; *·*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ; *·* ; | *a, b* :: *·* | ; *·*; *·▶ b /* |  |  | *b* :: *·* |  |  |
| *a* :: *·* | *a* :: *·* | *a* :: *·* |

Now we know that Γ1 = *{a},* Ω1 = *{b* :: *·}* and *N* 1 = *{a* :: *·}*.

Let’s come back to the right premise Υ2. Since the output of the Υ1 premise

formulas and (A)ll.

was

, the classical input for Υ2 will be

and the linear

input will be *{b* :: *·}*. Hence, Υ2 =

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ; | *b* :: *·* | ; |  |
| *a* :: *·* | *a* :: *·* |

; *b*; *· ▶*

; *·*.

With a similar derivation as the one just presented, it is easy to see that Γ2 = *{a}*, Ω2 = *N* 2 = *∅*. Therefore, everything is consumed and the output of the rule k will be empty.

*a* :: *·*

*·* ; *·* ;

This mimics the following derivation in the system *FL*2

*ew*

*a*; *b ▶ a*3 *⊗ b a*; *b ▶ a*3 *⊗ b*

*·*; *a*2*,b ▶ a*3 *⊗ b* k

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *·* ; *·* ; | *a* :: *·* | |  | |
| *a*3 *⊗ b /* | | Γ2 ; Ω2 ; | | *N* 2 |

where the *a*’s are split by the bounded contraction and *b* remains in the linear context.

As we could see, the system “counts” the number of times a formula was used during a proof: none, once or several times. If the formula was not used at all (i.e.,

it remains in the context), then it will be placed in the classical context ( *N* )

*N*

of the next premise of the k rule. If it was used once ( Ω context), then it must be part of the linear context in all premises. Finally, if it was used twice (or more), it will not be passed to the remaining premises.

Now let us explain the other rules in the *LFLk*

*ew*

system. The positive rules are

similar to those in [[12](#_bookmark34)]. For instance, the rule *⊗R* uses both Δ and *E* (Δ :: *E*) in

order to prove *F* . The formulas not used in the proof of *F* , i.e, Δ*j* :: *Ej*, are used in the proof of *G*. The final output of the proof of *F ⊗ G* is *Ejj* (since Δ*jj* cannot be returned).

Consider now the classical context in *⊗R*. The proof of *F* may move some

*Lj*

formulas in

*L*

producing

and, from *Lj* , the proof of *G* may also move some

other formulas producing *Ljj* . This “moving of formulas” is determined by the take

procedure (Def. [5.2](#_bookmark15)).

The negative rules *∧R* and *∨L* need some adjustments w.r.t. the system in [[12]](#_bookmark34) due to the presence of weakening. Consider the following rule form [[12](#_bookmark34)]:

Γ; Δ; *E ▶ F / Ej* Γ; Δ; *E ▶ G / Ej*

Γ; Δ; *E ▶ F ∧ G / Ej*

In the case of linear logic, the proof of *F* and *G* must output exactly the same excess of formulas *Ej*. Note that this is not the case in *FLew* since the rules internalize weakening. For instance, the sequent *a, b ▶ a ∧ b* is provable and, while the proof

of *a* outputs *b*, the proof of *b* outputs *a*. Hence, what we need as output is the multiset intersection (A) of the outputs of the two derivations. A similar situation happens with the classical context. For example, in a proof of *a ∧ a*2 from the context *·* ; *·* ; *a* :: *·* , the proof of *a* outputs *·* ; *a* :: *·* ; *·* and the proof of *a*2 outputs

*a* ; *·* ; *·* . Then, the proof of *a ∧ a*2 outputs *·* ; *a* :: *·* ; *·* A˜

where A˜ chooses the left most position of *a*. Formally,

*a* :: *·* ; *·* ; *·* = *a* :: *·* ; *·* ; *·*

**Negative Rules**

*⊥L TR*

*L* ; Δ*, ⊥*; *E ▶ C / L* ; *E L* ; Δ; *E ▶T / L* ; *E*

*⊗L*

*L* ; Δ*, F, G*; *E ▶ C / Lj* ; *Ej*

*L* ; Δ*,F ⊗ G*; *E ▶ C / Lj* ; *Ej*

*⊃R*

*L* ; Δ; *E ▶ F ⊃ G / Lj* ; *Ej*

*L* ; Δ*,F* ; *E ▶ G / Lj* ; *Ej*

*∧R*

*L* ; Δ; *E ▶ F / L*1 ; *E*1 *L* ; Δ; *E ▶ G / L*2 ; *E*2

*L* ; Δ; *E ▶ F ∧ G /* ; *E*1 A *E*2

*L*1A˜*L*2

*∨L*

*L* ; Δ*,F* ; *E ▶ C / L*1 ; *E*1 *L* ; Δ*, G*; *E ▶ C / L*2 ; *E*2

*L* ; Δ*,F ∨ G*; *E ▶ C /* ; *E*1 A *E*2

*L*1A˜*L*2

**Positive Rules**

*⊗R*

*L* ; *·*;Δ :: *E ▶ F / Lj* ; Δ*j* :: *Ej Lj* ; *·*; Δ*j* :: *Ej ▶ G / Ljj* ; Δ*jj* :: *Ejj*

*L* ; Δ; *E ▶ F ⊗ G / Ljj* ; *Ejj*

*⊃L*

*L* ; *·*;Δ :: *E ▶ F / Lj* ; Δ*j* :: *Ej Lj* ; *G*; Δ*j* :: *Ej ▶ C / Ljj* ; Δ*jj* :: *Ejj*

*L* ; Δ*,F ⊃ G*; *E ▶ C / Ljj* ; *Ejj*

*L* ; Δ*, F*1 *∧ F*2; *E ▶ C / Lj* ; *Ej*

*L* ; Δ*, Fi*; *E ▶ C / Lj* ; *Ej*

*∧Li*

*L* ; Δ; *E ▶ F*1 *∨ F*2 */ Lj* ; *Ej*

*L* ; Δ; *E ▶ Fi / Lj* ; *Ej*

*∨Ri*

**Structural Rules**

take(*p,* Δ*, L* ; *E*)= *Lj* ; *Ej*

*L* ; Δ; *E ▶ p / Lj* ; *Ej* init

*L* ; Δ*,F* ; *E ▶ C / Lj* ; *Ej*

; Δ*,F* ; *E ▶ C / Lj* ; *Ej*

*L \ F*

*L* ; Δ; *E ▶ C / Lj* ; *Ej DL L* ; Δ; *E, F ▶ C / Lj* ; *Ej DE*

Υ1 Υ2 *···* Υ*k* k

*L* ; Δ; *E ▶ C / Lk* ; *Ek*

Υ1 =

Δ :: *E*

*L* :: *·* ; *·* ;

; *·*; *·▶ C /*

; ; *·*

Υ*i* =

Δ*i—*1 :: *Ei—*1

*Li—*1 :: *·* ; *·* ;

|  |  |  |  |
| --- | --- | --- | --- |
| *L*1 :: Γ1 ; | ΩΔ :: Ω*E* |  | Δ1 :: *E*1 |

; ΩΔ; Ω*E ▶ C /*

; Ω*i ,* 2 *≤ i ≤ k*

Fig. 4. Lazy-splitting *LFLk* system. In the init axiom, *p* is atomic. Rules &*L*, *VLC* , *>LCC* , *>LCG*

Δ*i* :: *Ei*

*Li* :: Γ*i* ; Ω*i* ;

*E*

*ew*

in Figure [2](#_bookmark7) apply also on the Γ context. C represents multiset intersection. The definitions of C˜ and

take(*F,* Δ*, L* ; *E*) are given in Definition [5.2](#_bookmark15). In rule *DL*, *L \ F* means take(*F, ·, L* ; *·*). Rule init assumes

that *p* occurs either in Δ, or *E*. In k, *{*Δ*,E}* must be non-empty.

*L*

**Definition 5.2** Let *F* be a formula, be a classical context and *E* and Δ be

*L*

multisets of formulas. Assume that *F* occurs either in *L* , *E* or Δ. If *F* occurs in Δ, then take(*F,* Δ*, L* ; *E*) = *L* ; *E*. If *F* does not occur in Δ, then take chooses one of the occurrences of *F* and produces *Lj* ; *Ej* as follows:

*Lj*

*L*

* If *F* is taken from *E*, then

*L*

=

and *Ej* = *E \ {F}*

* Let

=

where *Lh* and *Lt* are (possibly empty) lists

of triples. If *F* is taken from Γ ; Ω ; *N* , then *E* = *Ej* and we have three

*Lh* :: Γ ; Ω ; *N* :: *Lt*

*Lj*

choices. If *F* is taken from Γ, then

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *Lh* :: | Γ *∪ {F}* | | ; | Ω *\ {F}* | | | ; *N* :: *Lt* | | |
| *Lh* :: Γ ; | | | Ω *{F}* | | | ; | *N \ {F}* | | :: *Lt* |  |

=

*Lj*

=

*Lj*

= *L* . If *F* is taken from Ω then

. Finally, if *F* is taken from *N* , then

. [6](#_bookmark17)

Let

= Γ1 ; Ω1 ;

and

= Γ2 ; Ω2 ;

be classical contexts containing

the same multiset of formulas, where each formula *F* may occur at possibly different

[*A*2]

[*A*1]

[*A*3]

*N*2

[*A*2]

*N*1

[*A*1]

positions (1 for *N* , 2 for Ω and 3 for Γ). The context as follows.

= A˜

is obtained

1. Start with = *·* ; *·* ; *·* .

[*A*3]

1. Let *F* be a formula in and let *n, m ∈ {*1*,* 2*,* 3*}* be the left most positions

[*A*2]

[*A*1]

[*A*1]

of *F* in

and

respectively.

1. Add *F* at position *max*(*n, m*) in [*A*3] .
2. Delete one occurrence of *F* at position *n* (resp. *m*) in

(resp. ).

[*A*2]

[*A*1]

1. If */*= *·* ; *·* ; *·* , goto to (ii).

*L*

*Lj*

[*A*1]

Given two lists of triples

*L*

*Lj*

and

of the same length and with each correspondent

triple containing the same multiset of formulas, we define applying A˜ to each correspondent triple.

A˜

by point-wise

Observe that the changes in the context are governed by the initial rule, due

*L*

to the take procedure, as follows: if *p* occurs in Δ (which is non returnable), *p* is taken from that context and the input and output coincide; otherwise, *p* is removed

from *E* or it is shifted to the left in the context.

*L*

**Theorem 5.3 (Soundness and Completeness)** *The sequent ·*;Δ *▶ G is prov-*

*·* ; *·* ; *·*

*ew*

*able in FLk*

*ew*

*if and only if*

; Δ; *·▶ G is provable in LFLk .*

**Proof.** The proof follows by mutual induction on sequents. We will show here how

to mimic a proof of *·*;Δ *▶ G* into *LFLk*

*ew*

and vice-versa.

Starting from the sequent *·*;Δ *▶ G* in *FLk*

*ew*

, one applies negative rules until only

k or positive rules can be applied. These will match exactly the moves for applying

*·* ; *·* ; *·*

*ew*

negative rules for proving linear context is considered.

; Δ; *·▶ G* in *LFLk*

, since only the non returnable

6 We stress out that we always take/add a formula *F* from/to the context respecting the Δ*,E* contexts. Hence, for example, if *F ∈ N*Δ, then *N \ {F}* actually means *N \ {F* :: *·}*.

Consider now that the rule k is applied (for the first time):

*π*1 *πk*

Δ1; Δ *▶ G ·· ·* Δ*k*; Δ *▶ G*

*·*; Δ*,* Δ1*,...,* Δ*k ▶ G* k

We may assume, for every 1 *≤ i ≤ k* and without loss of generality, that formulas in Δ*i* (resp. Δ) were used two or more times (resp. at most once) in *πi*, since Δ is linear (with weakening). Moreover, if *F ∈* Δ*i* was used at most once, it can be moved to Δ (again, due to weakening).

On mimicking *π*1, in order to have a proof *πj*

1

of the sequent

; *·*; *·▶ G /* ; *·*

Δ*,* Δ1*,...,* Δ*k* :: *·*

*·* ; *·* ;

*N* 1

Γ1 ; Ω1 ;

we have that: Γ1 = Δ1 since formulas in Δ1 were used two or more times, Ω1 are formulas in Δ used exactly once in *π*1, and *N* 1 = Δ2*,...,* Δ*k,* (Δ *−* Ω1) :: *·*, since

those formulas were never used in *π*1. Due to weakening, we may adapt *πj*

1

so that

Ω1 = Δ.

Following the same argument, we can mimic *πi* to a proof of the sequent

; Δ; *·▶ G /* ; *·*

Δ*i,...,* Δ*k* :: *·*

*·* ; *·* ;

|  |  |
| --- | --- |
| Γ*i* ; Ω*i* ; |  |
| Δ*i*+1*,...,* Δ*k* :: *·* |

Hence the output of

; Δ*,* Δ1*,...,* Δ*k* :: *·*; *·▶ G* will be

; *·*.

The positive rules are handled in the exact same way as in [[12](#_bookmark34)], so we will omit the details.

*·* ; *·* ; *·*

*·* ; *·* ; *·*

For the converse, let’s take a closer look at the (lazy) rule k

Υ1 Υ2 *·· ·* Υ*k* k

*L* ; Δ; *E ▶ C / Lk* ; *Ek*

On trying to prove the formula *C*, we move the whole linear context Δ*,E* to the

Δ :: *E*

*L* :: *·* ; *·* ;

“classical wannabe” context . Some formulas in Δ*,E* are linearly

consumed (ΩΔ*,* Ω*E*), some are classically consumed (Γ1) and some are not used at all (Δ1*, E*1). We now try to prove the same *C* with the same resources *but* the classical formulas in Γ1.

; ΩΔ*,* Ω*E*; *·▶ C*

Δ1 :: *E*1

*L* :: *·* ; *·* ;

This will consume formulas in ΩΔ*,* Ω*E,* Δ1*, E*1: some classically (Γ2), some linearly

(Ω2) and some will not be used at all (Δ2*, E*2*,* Ω2 , where Ω2 = ΩΔ*,* Ω*E −* Ω*i*). And

*E E*

so on. This process mimics perfectly the application of the rule k in *FLk*

*ew*

, where

part of the linear context becomes classical and it is not delivered to the other

branches of derivation. *2*

In Appendix C of [[4](#_bookmark26)] we present some examples of derivations in both systems, to illustrate better the use of contexts and rules.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequent** | *FFL*2  *ew* | *lazy-FL*2  *ew* | *LFL*2  *ew* |
| *p ⊃ ⊥,* (*p ⊃ q*) *▶* (*p ⊃ q*)3 | 2ms | 2ms | 7 ms |
| *r, p ⊃ r ⊃ q, p ⊃ q ▶* (*p ⊃ q*)3 | 47ms | 18ms | 8ms |
| *r, p ⊃ r ⊃ q,* (*p ⊃ q*)3 *⊃ s ▶ s* | 77ms | 33ms | 8ms |
| *a, r, p ⊃ r ⊃ q,* (*p ⊃ q*)3 *⊃ s ▶ s* | 4.2s | 576ms | 55ms |
| *a, b, r, p ⊃ r ⊃ q,* (*p ⊃ q*)3 *⊃ s ▶ s* | - | 168s | 55ms |

Fig. 5. Tests of the systems *FFLk*

*ew*

*ew*

, *lazy-FL*2

(*FFLk*

+ lazy rules for linear logic connectives) and

*LFFLk*

*ew*

(Figure [4](#_bookmark16)). “-” means more than 5 minutes.

*5.1 Eager lazy* k *rule and implementation*

We have implemented in Maude [http://maude.cs.illinois.edu](http://maude.cs.illinois.edu/) the system

*ew*

*FFL*2

*ew*

*ew*

(for *k* = 2), *FFL*2

with lazy splitting for the linear logic connectives only

(*lazy-FL*2

*ew*

*ew*

system) and the system *LFL*2

in Figure [4](#_bookmark16). In this section, we report

some experiments on these systems. All the tests were performed on an iMac, 2.9

GHz Intel Core i5 with 8 GB of RAM running Maude 2.7. The source code and examples can be downloaded from <http://subsell.logic.at/flew>. We stress out that, although the examples were implemented for *k* = 2, the implementation for any *k* is a trivial adaptation of the systems implemented so far.

Figure [5](#_bookmark18) summarizes the results of the tests. We tested the canonical examples for cut-elimination as those described in Section [2](#_bookmark1). As expected, the lazy strategies improved considerably the efficiency of the system. The advantages of using the lazy version of k can be appreciated in the last entries of the table. In those experiments, we added into the context some “useless” formulas, that can be weakened. It can

be noted that the uncontrolled use of k in the system *lazy-FL*2

*ew*

affects considerable

the performance of the solver.

Some engineering can be used in order to improve the proof search procedure.

For instance, the rule init in System *LFLk*

*ew*

introduces a non-determinism (predicate

taken). As pointed out in Definition [5.2,](#_bookmark15) we can eliminate some of the choices by

taking the atom *p* from the context Δ that cannot be returned. If the atom is not in Δ, it should be taken from the context Γ and the output coincides with the input. In other case, it must be taken from the other contexts.

Other improvements can be also applied. For instance, it is easy to prove that the following context transformations are safe and improve the search procedure (since they reduce the number of formulas in the context):

.

* Γ ; Ω ;

*~* Γ *∪ {F}* ; Ω ;

*N, F*

*N, F, F*

* Γ ; Ω ;

*N*

* Γ*,F* ; Ω ;
* Γ*,F* ; Ω ;

; Δ*, F,F* ; *E ~* Γ*,F* ; Ω ;

; Δ*,F* ; *E ~* Γ*,F* ; Ω ;

*N*

*N*

*N*

; Δ; *E, F ~* Γ*,F* ; Ω ;

*N*

*N*

*N*

; Δ; *E*.

; Δ; *E*.

; Δ; *E*.

* Γ*, F,F* ; Ω ;

*N*

; Δ; *E ~* Γ*,F* ; Ω ;

; Δ; *E*.

More interestingly and actually quite surprising, Proposition [5.5](#_bookmark21) below shows a sufficient condition for determining if an application of the rule k can be avoided.

For that, we introduce the system *KLFLew* that differs from *LFLk*

*ew*

only in the rule

k.

**Definition 5.4** (System *KLFLew*) The system *KLFLew* shares with *LFLk*

*ew*

all the

proof rules but k, that is substituted by the following rules

Υ1 *·· ·* Υ*k*

k

1

*L* ; Δ; *E ▶ C / Lk* ; *Ek*

Υ1

2

k

*L* ; Δ; *E ▶ C / L*1 ; *E*1

where Υ1*,...,* Υ*k* are as in Figure [4](#_bookmark16) and(resp.) means that Γ1 */*= *∅* (resp.

Γ1 = *∅*). Similar to *LFLk*

*ew*

, *{*Δ*, E}* must be non-empty.

Rule k1 is the same as k in the *LFLk*

*ew*

system. Rule k2 detects whether the

application of k is “useless”. This happens when the formulas were used at most

once and then, the Γ1 context remains empty. In that case, a proof of Υ1 is indeed a proof of the sequent in the conclusion. For instance, consider a proof of the sequent

*a, b ▶ a ⊗ b* where k (in *LFLk*

*ew*

) is not needed. If we (eagerly) apply k, we obtain

the following (omitting the redundant premises):

*π*1

; *·*; *·▶ a ⊗ b /*

*·* ; *·* ; *·*

*a, b* :: *·*

*·* ; *·* ;

|  |  |  |
| --- | --- | --- |
| *·* ; | *a, b* :: *·* | ; *·* |

*π*2

; *a, b*; *·▶ a ⊗ b /* [*L*2] ; *E*2 *...*

k

[*·*] ; *a, b*; *·▶ a ⊗ b /* [*·*] ; *·*

Note that *π*1 (up to some minor syntactic conversions) is indeed the proof of the sequent in the conclusion (which is the same as the ones on the other premises).

Ω

Since *π*1 only uses once the formulas *a, b*, they are returned in the remains empty. Then, we have the following proof in *KLFLew*

Γ

*π*1

context and

k2

[*·*] :: *·* ; *·* ;

; *·*; *·▶ a ⊗ b /*

[*·*] ; *a, b*; *·▶ a ⊗ b /* [*·*] ; *·*

*a, b* :: *·*

[*·*] :: *·* ; *a, b* :: *·* ; *·*

that ignores the useless application of k.

**Proposition 5.5** *The sequent L* ; Δ; *E ▶ G is provable in LFLk*

*ew*

*iff it is provable*

*in KLFLew.*

**Proof.** (*⇒*) If there is a proof of a sequent Υ in *KLFLew*, it is easy to produce a

proof ofΥ in *LFLk* by simply dropping the applications of k2 and substituting any

*ew*

instance of k1 with k in *LFLk* .

*ew*

(*⇐*) We proceed by induction on the length of the derivation of the sequent Υ

in *LFLk* . We need to show that k*i* can be always introduced before any other

*ew*

connective (belonging to the positive phase). Consider for instance the following

derivation where k is not used in *πi* – since k permutes down due to Proposition [3.2](#_bookmark8) (we omit some of the outputs):

*π*1 *π*2

*L* ; *·*;Δ :: *E ▶ G / Lj* ; Δ*j* :: *Ej Lj* ; *·*; Δ*j* :: *Ej ▶ Gj*

*L* ; Δ; *E ▶ G ⊗ Gj ⊗R*

In *KLFLew*, we have the following derivation (we also omit some of the outputs):

*j j*

*π*

*π*

1 2

; *·*; *·▶ Gj*

; *·*; *·▶ G ⊗ Gj*

k2

*L* ; Δ; *E ▶ G ⊗ Gj*

Δ :: *E*

*L* :: *·* ; *·* ;

; *·*; *·▶ G*

*L* :: *·* ; *·* ;

Δ :: *E*

|  |  |  |  |
| --- | --- | --- | --- |
| *Lj* :: *·* ; | Δ1 :: *E*1 | ; | Δ2 :: *E*2 |

*⊗R*

Derivations *πj*

1

and *πj*

can be easily built from *π*1 and *π*2 (up to some simple

syntactic conversions). We note that Δ1 *∪* Δ2 = Δ and similarly for *E*. The interesting part is that the output of the derivation on the left must be *Lj* ;Δ :: *E* since none of the formulas in Δ*,E* was moved into the Γ contexts. This justifies the use of k2. *2*

2

# 6 Conclusion and future work

In this work we introduced focused proof systems *FFLk*

*ew*

(*k >* 1) for Full Lambek

Calculus with exchange and weakening extended with bounded contractions. The

system was used to show that the validity problem is in EXPTIME. Although, proof theoretically speaking, these are satisfactory results, they do not lead to good implementation strategies.

We then proposed a new and non-trivial extension of lazy splitting to bounded

contractions, showing soundness and completeness of the resulting systems *LFLk* .

*ew*

This notion of laziness for bounded contractions is crucial for implementing good

provers for all the logics considered in this paper.

There are a number of ways of continuing this work. The first goal is to determine the exact complexity of *FLew* plus bounded contractions. That is, knowing that checking validity in such systems is PSPACE-hard [[11](#_bookmark33)], Theorem [4.2](#_bookmark11) implies that such complexity can be either PSPACE-complete or EXPTIME-complete.

Also, all the results presented in this paper strongly rely on the fact that the base logical system allows weakening. Without it, the situation is more complicated. For example, there is no more a clear separation between formulas behaving classically or linearly, and hence the notion of lazy splitting has to be completely reformulated.

We are currently working on implementing the system *KLFLew* (Definition [5.4](#_bookmark20))

(a prototypical version can be found at <http://subsell.logic.at/flew>). The

main challenge is guaranteeing termination (note that one implication on the Γ context may generate a loop). The implementation of the system in Figure [4](#_bookmark16) guar- antees termination by keeping the history of sequents and detecting loops (see The-

orem [4.1](#_bookmark10)). Loop detection in *KLFLew* is harder: after applying *−◦L*, some formulas

can be moved into the linear context and then, the lazy version of k can be eagerly

applied again. Hence, the sequents in the history are not exactly the same since the list of [*A*] contexts grows after applying k. We can rely on Theorem [4.2](#_bookmark11) to bound the size of such list. However, we believe that something better can be done: if a new [*A*] context is added, and it shares with the previous ones the same formulas

Ω

*N*

in the

and

contexts, then a loop is detected.

The bounded contraction rules are a proper subset of the sequent rules that can be extracted from the Hilbert axioms in the class *N*2 in the hierarchy of [[2]](#_bookmark23) (see <https://www.logic.at/tinc/webaxiomcalc/axiomcalc.html> for an imple- mentation of this algorithm). A great step forward would be to transform all se- quent calculi for substructural logics generated by the algorithm in [[2](#_bookmark23)] into efficient automatic provers.

# References

1. Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.
2. Agata Ciabattoni, Nikolaos Galatos, and Kazushige Terui. From axioms to analytic rules in nonclassical logics. In *23th Symp. on Logic in Computer Science*, pages 229–240. IEEE Computer Society Press, 2008.
3. Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 232(1-2):133–163, 2000.
4. Agata Ciabattoni, Bj¨orn Lellmann, Carlos Olarte, and Elaine Pimentel. From cut-free calculi to automated deduction: the case of bounded contraction (technical report), 2016. Available on authors’ web pages.
5. Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *J. of Symbolic Logic*, 57(3):795– 807, September 1992.
6. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
7. N. Galatos and P. Jipsen. Residuated frames with applications to decidability. *Transactions of the* *AMS*, 365(3), 2013.
8. N. Galatos, P. Jipsen, T. Kowalski, and H. Ono. *Residuated Lattices: an algebraic glimpse at* *substructural logics*. Elsevier, 2007.
9. Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic.

*Information and Computation*, 110(2):327–365, 1994.

1. R. Hori, H. Ono, and H. Schellinx. Extending intuitionistic linear logic with knotted structural rules.

*Notre Dame Journal of Formal Logic*, 35(2):219–242, 1994.

1. Rostislav Horcik and Kazushige Terui. Disjunction property and complexity of substructural logics.

*Theoretical Computer Science*, 412:3992–4006, 2011.

1. Pablo L´opez and Ernesto Pimentel. Resource management in linear logic search revisited. In Harald Ganzinger, David A. McAllester, and Andrei Voronkov, editors, *LPAR’99*, volume 1705 of *LNCS*, pages 304–319. Springer, 1999.
2. Dale Miller and Alexis Saurin. From proofs to focused proofs: a modular proof of focalization in linear logic. In J. Duparc and T. A. Henzinger, editors, *CSL 2007: Computer Science Logic*, volume 4646 of *LNCS*, pages 405–419. Springer, 2007.
3. M. Okada and K. Terui. The finite model property for various fragments of intuitionistic linear logic.

*J. Symbolic Logic*, 64(2):790–802, 1999.

1. Andreja Prijatelj. Bounded contraction and gentzen-style formulation of lukasiewicz logics. *Studia Logica*, 57(2/3):437–456, 1996.