Proof-theory for term representation
Internship report

Ulysse Gérard, supervised by Dale Miller, team Parfisal*

November 7, 2015

The general context

Since the introduction of typed λ-calculus and the discovery of the Curry-Howard isomorphism, term calculi and proof systems evolved in an interlaced way. This fruitful symbiosis has brought numerous applications. On the one hand, typing systems helped develop mathematically well founded high-level programming languages with rich features, and the λ-calculus turned out to be a useful tool to study computation and complexity. On the other hand, formal logic led to the development of logic programming, a new paradigm of which *Prolog* is the most famous representative.

This internship lies within the framework of logic with *focusing* and the proof-theory point of view is adopted here, not the one of typing systems. Since its appearance in the early 1990's with Andreoli's LLF system [And92], focusing has had many important applications in computational logic. Many systems appeared realizing different kinds of focusing such as LJT and LJQ [Her95b, DL06]. However, more recently, in the past ten years, a new kind of focused systems for linear, classical and intuitionistic logic stood out from the crowd [LM09]. Named LKF and LJF, these systems are generalizations of the previous ones. Their flexibility comes from the use of polarized formulas and focusing on the right and left hand-sides of sequents. Simultaneously appeared multi-focusing systems [CMS08] which allow the identification of sequent calculi to known parallel structures such as proof nets [Gir87] and expansion trees [CHM13].

If LJT and LJQ were initially proposed with two term calculi associated to them, the annotation of LJF with terms is a very recent work by Brock-Nannestad, Guenaud and Gustafsson [BNGG15] which shows many interesting features. That was the starting point of this internship, which led us to annotate a multi-focused version of LJF.

The research problem

The internship covered various aspects around terms representation and proof theory. The goal was to adopt a fresh point of view in the study of links between terms and logic. One of the main foci was the study of sharing and parallelism: what can we learn about terms from proof theory, and vice versa? This work may have repercussions on concrete projects such as compilation, logic programming and proof certification (in the framework of ProofCert), but it is also interesting in a purely theoretical point

*This internship was found by the ERC grant ProofCert. Details about the project can be found at `https://team.inria.fr/parsifal/proofcert/`.

of view as it is a step further in our comprehension of the interactions between logic and computation. Logic is a fundamental discipline and focusing a fundamental logical technique, thus there are big hopes to find interesting behaviors of the terms associated to these systems.

Your contribution

Many paths were followed during the internship, some more interesting than others. Not all of them will be exposed in this report. The main contribution is the creation of a term calculus for a multi-focused system. In this setting parallelism can be seen directly in normal (cut-free) terms. To achieve this goal, a progressive study of existing proof systems with more or less expressive terms has been carried out. We dissected the existing calculi to understand the way they work, their advantages, their flaws. Then we modeled and reused them in a more complex calculus, the one which annotates multi-focusing. Multi-focusing is an addition to focusing, therefore no features of the previous calculi are lost in the process, but its complexity and expressivity increases.

Arguments supporting its validity

The new term calculus presented in this report corresponds to a very interesting sequent calculus (with multi-focusing). In fact it shares a lot of features with other structures like proof-nets and expansion trees even if sequent calculi are not parallel in essence – while proof nets have been designed to eliminate some forms of bureaucracy in proofs such as the choice of order of application of inference rules. It is therefore natural to expect to find some parallelism in this calculus and in the terms associated to it.

Therefore, the major part of internship was about design. Multi-focused version of LJF with terms are new systems whose shape is still subject to changes and the main challenge was to find coherent structural rules and terms. While we are confident that this term calculus really implements the kind of sharing and parallelism that we were looking for, more formal evidences of that should now be developed.

Summary and future work

As stipulated in the previous paragraph, the established calculus seems fairly robust, but a priority future work would be to study how cut-elimination and β-reduction would apply to it and if they are compatible with the sharing and parralel features of our calculus. That being said, the research conducted during this internship opens lots of interesting fields of thought covering concrete subjects such as compilation, logic programming and automatic proof search and certification as well as more theoretical concerns about proof-theory and term representations.

Contents

1 Introduction

Introduced by Alonzo Church in the thirties, the lambda-calculus (λ-calculus) first appeared as a tool to study computation – it is actually equal to a Turing machine in expressiveness. However it also played an important role in the development of functional programming languages such as ML or Haskell which essentially implement an augmented (with constants and data types among others) version of it.

One of the first interactions between λ-calculus and logic appeared with the simply-typed λ-calculus of Alonzo Church in 1940. Created to avoid paradoxical uses of the untyped λ-calculus by restricting function applications to acceptable types of inputs, it has since known a huge success, an uncountable number of reformulations and a great variety of extensions and this especially after the discovery of the Curry-Howard isomorphism.

Matching types with formulas, cuts with reductions, proofs with programs, the Curry-Howard correspondence highlighted the tight link that exists between logic and typing systems, and nowadays it is traditional to present such systems in the form of sequent calculi similar to the ones used in logic.

The purpose of this internship was to understand and refine the term representations associated with newest logical systems implying focusing, and especially the multi-focused ones. How can we define a term representations for such systems, what are the computational impacts and which tools and uses could emerge from them? Those are some of the questions that will guide us in the course of this report.

We will first show basic notions about the logical systems that we are going to use by recalling their precursors, namely *natural deduction* and *sequent calculus*. Then focusing will be introduced before addressing the crux of the subject: how to add terms to such systems, and what are the interesting properties of these typing systems.

2 Logic reminder

This part will recall some basic notions about natural deduction, sequent calculus and their interactions with λ-terms. It has been kept deliberately simple and concise as only the notions that are of interest for the remainder of this report are developed.

As we are focused on "very simply typed" λ-calculi (only base and arrow types), the sole logic connective that we will use is $\supset$, namely the intuitionistic implication, and others won't be mentioned (except in the section 3.2 about LJF). Indeed, the logical systems presented here, such as LJ for intuitionistic logic, are truncated and thus not complete with regards to intuitionistic logic. The generalization of the ideas exposed in this report to full intuitionistic logic should be relatively straightforward and similar to what has already been done for other type systems.

2.1 General definitions

First of all we will need a notion of formula to describe our systems. The propositional calculus we will be using is most simple:

Definition 2.1. Let $\mathcal{A}$ be a set of symbols ($a, b, c, ...$) whose elements are called *atoms*, then *formulas* (denoted by capital letters $A, B, C, ...$) are described by the following grammar:

$$A, B ::= a \in \mathcal{A} \mid A \supset B$$

Here $A \supset B$ is read "A implies B" as $\supset$ is the intuitionistic implication.

We also need some basic terms to annotate the systems:

Definition 2.2. Let $\mathcal{X}$ be a set of symbols ($x, y, z, ...$) whose elements are called *term variables*, the terms of our basic calculus are made of variables, applications and abstractions:

$$t, u, v ::= x \in \mathcal{X} \mid u\ v \mid \lambda x.t$$

We describe here *administrative normal forms* that will be used later on the report, it is a normal form for terms organizing sharing (naming of subterms) in a very specific way:

Definition 2.3. Terms in *Administrative normal form* (or A-normal form) are described by the following grammar:

$$\begin{aligned} val &::= x \in \mathcal{X} \mid \lambda x.t \\ vlist &::= val\ vlist \mid \varepsilon \\ t, u &::= val \mid \mathtt{let}\ x = val\ vlist\ \mathtt{in}\ u \end{aligned}$$

Finally we need to formalize the containers of formulas that we will use:

Definition 2.4.

- We call $t : A$ a *type assignment* when t is a term and A a formula.
- We call a *declaration* a type assignment of the form $x : A$ with $x \in \mathcal{X}$ and A a formula.
- *Context* and *store* may refer to different objects. They can either be sets, multisets, lists or any other container. Contexts contains formulas while stores contains declarations. The kind of context or store used will be specified for each studied system.

Definition 2.5. Given a term t, the set of *free variables* of t, denoted $\mathsf{FV}(t)$, is inductively defined on the structure of the term:

$$\mathsf{FV}(x) = x \text{ for } x \in \mathcal{X} \qquad \mathsf{FV}(\lambda x.t) = \mathsf{FV}(t) \backslash \{x\} \qquad \mathsf{FV}(tu) = \mathsf{FV}(t) \cup \mathsf{FV}(u)$$

And the set of *bound variables* of t, $\mathsf{BV}(t)$ is defined on the same way:

$$\mathsf{BV}(x) = \emptyset \qquad \mathsf{BV}(\lambda x.t) = \mathsf{BV}(t) \cup \{x\} \qquad \mathsf{BV}(tu) = \mathsf{BV}(t) \cup \mathsf{BV}(u)$$

Definition 2.6. For $x \in \mathcal{X}$ and t, u two terms, we name *substitution of* x *in* t *by* u and denote $t\{x/u\}$ the operation of replacing all free occurrences of x in t by u.

We will also see *explicit substitutions* in the remainder of this report. Denoted $t[x/u]$ or $\mathtt{let}\ x = u\ \mathtt{in}\ t$, these are terms augmenting the syntax of our simple calculus and not writing tricks such as the implicit substitutions we just defined. When used, the grammar of the augmented calculus will be made explicit.

Definition 2.7. As usual we call β-*reduction* the rewriting rule

$$(\lambda x.t)\ u \to_\beta t\{x/u\}$$

and η-*expansion* the following

$$t \to_\eta \lambda x.(t\ x) \text{ with } x \text{ not free in } t$$

and α-*conversion* the act of renaming (with a fresh name) every occurrences of a bound variable in a term.

We will work under the "Barendregt conventions":

- No variable is both free and bound.
- Bound variables have all different names.

Readers looking for a more comprehensive introduction to typed λ-calculus may have a look at Barendregt's writings [Bar93][BG].

2.2 Natural deduction

Natural deduction is a proof system introduced by Gentzen for propositional logic. His goal was to establish a more natural treatment of deduction in logic. But it is also well known that a fragment of this system matches the simply typed λ-calculus. This fragment is composed of the axiom rule and both the introduction and the elimination rules for implication:

$$\frac{}{A \in \Gamma \vdash A}\ \mathrm{Ax} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ \supset \text{ intro} \qquad \frac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \supset \text{ elim}$$

where the context Γ is a set of formulas.

This system can directly be annotated with terms, giving us Church's simply-typed λ-calculus [Chu40]:

$$\frac{}{x : A \in \Gamma \vdash x : A}\ \mathrm{Ax} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B}\ \supset \mathrm{i}$$

$$\frac{\Gamma \vdash u : A \supset B \qquad \Gamma \vdash v : A}{\Gamma \vdash u\ v : B}\ \supset \mathrm{e}$$

where the store Γ is a set of declarations.

Definition 2.8. In natural deduction the proofs ending with an elimination rule whose main premise is the output of an introduction rule for the same symbol are called *cuts*.

One of the main features of natural deduction is the possibility to transform any proof in a cut-free proof of the same formula with an algorithm that we will called the cut-elimination procedure. Noticeably, a step in this procedure corresponds to a step of β-reduction in the calculus, for example:

$$\frac{\dfrac{\dfrac{D_1}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda x.t : A \supset B}\ \supset \mathrm{i} \qquad \dfrac{D_2}{\Gamma \vdash u : A}}{\Gamma \vdash (\lambda x.t)\ u : B}\ \supset \mathrm{e} \quad \rightarrow_{\mathrm{cut-elim}} \quad \begin{array}{c} D_2 \\ \Gamma \vdash u : A \\ D_1 \\ \Gamma \vdash t\{x/u\} : B \end{array}$$

Here the cut is eliminated by suppressing the hypothesis A in all sequents in the proof D_1 and then replacing with the proof D_2 each axiom rule used with A.

2.3 The sequent calculus LJ

Sequent calculi are deductive systems manipulating sequents. Sequents are usually constituted of at least two zones separated by a metalogical connective called turnstile ($\vdash$). These zones can be either lists, sets, multisets or singletons of formulas. The choice of stores behaviors may seem frivolous at first sight, but it is very important in the design of an inference system. Often several choices are available, but choosing

the fittest one allows for much simpler and more canonical proofs, by enforcing some strategies, or regrouping small inferences steps into bigger and more suitable ones.

Here is the example of the sequent calculus LJ for intuitionistic logic:

Definition 2.9. The *sequents* in LJ are of the form $\Gamma \vdash B$ where Γ is a multiset of formulas and B is a formula[1].

As before, we will only be interested in the implicational fragment of LJ for which the inference rules are:

$$\frac{\Gamma \vdash u : A \qquad \Gamma, x : B \vdash t : C}{\Gamma, y : A \supset B \vdash t\{x = y\ u\} : C}\ \supset_l \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B}\ \supset_r$$

$$\frac{\Gamma \vdash u : A \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash t\{x/u\} : B}\ \text{cut} \qquad \frac{}{\Gamma, x : A \vdash x : A}\ \text{Ax}$$

Where $t\{x/u\}$ is the term t in which all occurrences of x have been *substituted* by u.

Once again, cut-elimination steps corresponds to β-reduction steps, but in a more straightforward way than for natural deduction (cuts are directly materialized by the presence of the cut rule):

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x : A \vdash x : A}\ \text{Ax}}{\Gamma \vdash \lambda x.x : A \supset A}\ \supset_r \qquad \dfrac{\dfrac{}{u : B \vdash u : B}\ \text{Ax} \qquad \dfrac{}{\Gamma, z : A \vdash z : A}\ \text{Ax}}{\Gamma, y : A \supset A \vdash y\ a = z\{z/y\ a\} : A}\ \supset_l}{\Gamma = \{u : B\} \vdash (\lambda x.x)\ u = (y\ u)\{y/\lambda x.x\} : B}\ \text{Cut}$$

$$\rightarrow_{\texttt{cut-elim}} \quad \frac{}{u : B \vdash u : B}\ \text{Ax}$$

Thus, when designing a proof system it is possible to infer the reduction rules of the associated calculus by carefully studying cut-elimination. An example of this construction is present in [BNGG15] for LJF, and the resulting system will be described in 4.3.

In the rest of this report cut-elimination won't be our major concern. In fact we will mostly work with cut-free sequent calculi (from which the rule Cut has been removed). These systems are interesting because they only type terms which are in a sort of *normal form*. For example the previous one, LJ, without Cut would exactly correspond to the classical β-normal terms of the λ-calculus .

But the rough sequent calculus presentation of typing systems has at least one major drawback: a unique term can have several typing derivations, that is, a formula typing a term can have several proofs. (For example the sequent $f : A \supset B, y : A \vdash \lambda x.f\ y : C \supset B$ is provable either by applying $\supset_l$ followed by $\supset_r$ or by the other way round).

We will now introduce the notion of focusing which addresses this issue.

[1] It is interesting to remark that LJ is obtained from LK, its classical counterpart, by restricting the right-hand-sides of sequents to contain at most one formula.

3 Focusing

Sequent calculus has encountered a lot of success in giving a proof theory to classical, intuitionistic and linear logics. However its main feature, tiny low-level building blocks for proofs, is a major drawback for proof generation. A simple algorithm based on it would give rise to a lot of non-determinism. Moreover, this chaos allows for numerous proofs of the same formula implying a lack of canonicity. For example, there are many choices in the order of the inference rules which are completely equivalents. Focusing techniques appeared has a mean to guide the proving process and to reduce mayhem in the application of inference rules in LJ and LK, its classical counterpart.

One of these early techniques, *uniform proofs* [MNPS91], consists in an alternation of two phases: *goal-directed search* and *backchaining*. It was developed for the purpose of enriching logic programming languages such as Prolog. The result of this operation is the λProlog language [MN12][2].

Then Andreoli extended the same two-phases technique to linear logic giving birth to *focusing* [And92]. Latter appeared several systems with different kinds of focusing such as Herbelin's LJT [Her95b] and Dyckhoff and Lengrand's LJQ [DL06].

But our main system of interest will be the LJF system elaborated by Liang and Miller [LM09] which is a more general focused version of LJ more general than LJT and LJQ. We will then see how these fragments can be obtained from LJF by restricting atoms polarities.

3.1 Essential mechanisms

Invertibility The main principle of focusing is to organize a proof in an alternation of two phases. The *asynchronous phase* during which are applied invertible rules only and the *synchronous phase*.

> **Definition 3.1.** We call *invertible* the rules whose conclusion and premises are equiprovable. Two such rules can always be permuted in the tree of inferences.

Thus the application order of a series of invertible rules has no impact on the provability of a formula. We will then choose to always apply them following the left-most order. In the two-phases framework these sequences will constitute the *asynchronous* phase. As we shall see, connectives whose rules are invertible will be called *negatives* or *asynchronous* whereas the others will be called *positives* or *synchronous*.

Connectives and polarity As stated previously, we will describe the entire LJF system, which is complete with respects to first-order intuitionistic logic, even if we will only use the implicational fragment.

The usual conjunction connective of intuitionistic logic comes in two versions: $\wedge^+$ and $\wedge^-$ which are equivalent in terms of provability. The polarity of a formula is therefore entirely determined by its top-level connective. The disjunction connective only has a positive occurrence, $\vee$, because of the intuitionistic setting, but the $\supset$ connective can be seen to be a form of $\vee^-$ that needs to be controlled.

The formulas (denoted $\mathsf{A}, \mathsf{B} \dots$), positive formulas ($\mathsf{P}, \mathsf{Q} \dots$) and negative formulas ($\mathsf{N}, \mathsf{M} \dots$) are defined as follows:

[2] λProlog which was used during this internship as a prototyping language.

Definition 3.2. Let $\mathcal{A}^+$ be a set of symbols $(a^+, b^+, c^+, ...)$ whose elements are called *positive atoms* and $\mathcal{A}^-$ its negative counterpart. Formulas are:

- $A, B ::= P \mid N$
- $P, Q ::= a^+ \in \mathcal{A}^+ \mid t \mid f \mid A \wedge^+ B \mid A \vee B \mid \exists x.A$
- $N, M ::= a^- \in \mathcal{A}^- \mid A \wedge^- B \mid A \supset B \mid \forall x.A$

The provability of a formula is not changed by the choice of polarities attributed to its atom. That is, erasing the polarities of atoms and connective of a provable formula in LJF always leads to a provable formula in LJ.

3.2 LJF, an intuitionistic focused sequent calculus

Schematic variables

- Γ is a multiset of polarized formulas.
- Θ is a list of polarized formulas.
- C denotes either a negative formula or positive atom.
- E denotes either a positive formula or negative atom.
- $\mathcal{R}$ denotes $\Delta_1 \Downarrow \Delta_2$ where $\Delta_1 \cup \Delta_2$ should contain exactly one element.
- As usual, P and N respectively denote positive and negative formulas and $A, B, ..$ arbitrary polarized ones.

Three kinds of sequents

- $\Gamma \Uparrow \Theta \vdash \Delta_1 \Uparrow \Delta_2$ are *unfocused* sequents, where $\Delta_1 \cup \Delta_2$ should contain exactly one element.
- $\Gamma \Downarrow A \vdash E$ are *left-focused* sequents.
- $\Gamma \vdash A \Downarrow \cdot$ are *right-focused* sequents.

The system (Fig. 1) The rules are divided in four groups:

- Asynchronous rules used during the negative phases;
- Synchronous rules used during the focusing phases;
- Structural rules which mediate between the phases: D-ecide rules start focusing and R-elease rules stop it.

In the asynchronous phase it has been chosen to consume the list Θ in the left-most order until it is empty and then Δ_1. The phases are maximal, that is to say that it is not possible to focus on the left if the right formula is not positive (or a negative atom) and the foci can't be released until it turns positive. Therefore Γ will always contain only negative formulas or positive atoms.

Asynchronous Introduction Rules

$$\frac{\Gamma \Uparrow A \vdash B \Uparrow \cdot}{\Gamma \Uparrow \cdot \vdash A \supset B \Uparrow \cdot} \supset_r \qquad \frac{\Gamma \Uparrow \cdot \vdash A \Uparrow \cdot \qquad \Gamma \Uparrow \cdot \vdash B \Uparrow \cdot}{\Gamma \Uparrow \cdot \vdash A \wedge^- B \Uparrow \cdot} \wedge^-_r \qquad \frac{}{\Gamma \Uparrow \cdot \vdash t^- \Uparrow \cdot} t^-_r$$

$$\frac{\Gamma \Uparrow \cdot \vdash [y/x]A \Uparrow \cdot}{\Gamma \Uparrow \cdot \vdash \forall x.A \Uparrow \cdot} \forall_r \qquad \frac{\Gamma \Uparrow [y/x]A, \Theta \vdash \mathcal{R}}{\Gamma \Uparrow \exists x.A, \Theta \vdash \mathcal{R}} \exists_l \qquad \frac{}{\Gamma \Uparrow f^+, \Theta \vdash \mathcal{R}} f^+_l$$

$$\frac{\Gamma \Uparrow A, B, \Theta \vdash \mathcal{R}}{\Gamma \Uparrow A \wedge^+ B, \Theta \vdash \mathcal{R}} \wedge^+_l \qquad \frac{\Gamma \Uparrow \Theta \vdash \mathcal{R}}{\Gamma \Uparrow t^+, \Theta \vdash \mathcal{R}} t^+_l \qquad \frac{\Gamma \Uparrow B_1, \Theta \vdash \mathcal{R} \qquad \Gamma \Uparrow B_2, \Theta \vdash \mathcal{R}}{\Gamma \Uparrow B_1 \vee B_2, \Theta \vdash \mathcal{R}} \vee_l$$

Synchronous Introduction Rules

$$\frac{\Gamma \vdash A \Downarrow \cdot \qquad \Gamma \Downarrow B \vdash E}{\Gamma \Downarrow A \supset B \vdash E} \supset_l \qquad \frac{\Gamma \vdash A \Downarrow \cdot \qquad \Gamma \vdash B \Downarrow \cdot}{\Gamma \vdash A \wedge^+ B \Downarrow \cdot} \wedge^+_r \qquad \frac{\Gamma \Downarrow A_i \vdash E}{\Gamma \Downarrow A_1 \wedge^- A_2 \vdash E} \wedge^-_l$$

$$\frac{\Gamma \vdash A_i \Downarrow \cdot}{\Gamma \vdash A_1 \vee A_2 \Downarrow \cdot} \vee_r \qquad \frac{\Gamma \Downarrow [t/x]A \vdash E}{\Gamma \Downarrow \forall x.A \vdash E} \forall_l \qquad \frac{\Gamma \vdash [t/x]A \Downarrow \cdot}{\Gamma \vdash \exists x.A \Downarrow \cdot} \exists_r \qquad \frac{}{\Gamma \vdash t^+ \Downarrow \cdot} t^+_r$$

Identity rules

$$\frac{}{\Gamma \Downarrow a^- \vdash a^-} I_l \qquad \frac{}{\Gamma, a^+ \vdash a^+ \Downarrow \cdot} I_r \qquad \frac{\Gamma \Uparrow \cdot \vdash B \Uparrow \cdot \qquad \Gamma \Uparrow B \vdash \cdot \Uparrow E}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow E} \text{Cut}$$

Structural rules (Decide, Release, Store)

$$\frac{\Gamma, N \Downarrow N \vdash E}{\Gamma, N \Uparrow \cdot \vdash \cdot \Uparrow E} D_l \qquad \frac{\Gamma \vdash P \Downarrow \cdot}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow P} D_r \qquad \frac{\Gamma \Uparrow P \vdash \cdot \Uparrow E}{\Gamma \Downarrow P \vdash E} R_l \qquad \frac{\Gamma \Uparrow \cdot \vdash N \Uparrow \cdot}{\Gamma \vdash N \Downarrow \cdot} R_r$$

$$\frac{C, \Gamma \Uparrow \Theta \vdash \mathcal{R}}{\Gamma \Uparrow C, \Theta \vdash \mathcal{R}} S_l \qquad \frac{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow E}{\Gamma \Uparrow \cdot \vdash E \Uparrow \cdot} S_r$$

Figure 1: The focused intuitionistic sequent calculus LJF.

3.3 Switching strategies using polarity

One of the main success of LJF is the ability to change the proof search strategy with different choices of atom's polarities. We will illustrate this with the standard forward / backward chaining example.

Let us consider the following start sequent:

$$\Gamma \Uparrow \cdot \vdash \cdot \Uparrow c \qquad \text{where } \Gamma = \{a, a \supset b, b \supset c\}$$

Forward chaining On the one hand, choosing positive polarities for a and b forces to adopt the forward chaining style (where $a \supset b$ is processed before $b \supset c$). Indeed starting by focusing over $b \supset c$ and then applying $\supset_l$ would require the use of the rule I_l with atom c which would fail because of c being positive. The correct derivation is the following:

$$\dfrac{\dfrac{}{a \in \Gamma \vdash a \Downarrow \cdot}\, I_r \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, b \vdash b \Downarrow \cdot}\, I_r \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, b, c \vdash c \Downarrow \cdot}\, I_r}{\Gamma, b, c \Uparrow \cdot \vdash \cdot \Uparrow c}\, D_r}{\Gamma, b \Downarrow c \vdash c}\, R_l + S_l}{\Gamma, b \Downarrow b \supset c \vdash c}\, I_r}{\Gamma, b \Uparrow \cdot \vdash \cdot \Uparrow c}\, D_l}{b \vdash c \Downarrow \cdot}\, R_l + S_l}{\Gamma \Downarrow a \supset b \vdash c}\, \supset_l}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow c}\, D_l$$

Backward chaining On the other hand, and for a similar reason to the previous case, with negative atoms the proof becomes:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \Downarrow a \vdash a}\, I_l}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow a}\, D_l}{\Gamma \vdash a \Downarrow \cdot}\, R_r + S_r \qquad \dfrac{}{\Gamma \Downarrow b \vdash b}\, I_l}{\Gamma \Downarrow a \supset b \vdash b}\, \supset_l}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow b}\, D_l}{\Gamma \vdash b \Downarrow \cdot}\, R_r + S_r \qquad \dfrac{}{\Gamma \Downarrow c \vdash c}\, I_l}{\Gamma \Downarrow b \supset c \vdash c}\, \supset_l}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow c}\, D_l$$

In fact these two strategies are the ones implemented in the focused sequent calculi LJQ (Dyckhoff & Lengrand [DL06]) and LJT (Herbelin [Her95b]) which turned out to be fragments of LJF. It is interesting to note that the λ-calculus associated with these systems will respectively adopt the call-by-name and call-by-value reduction strategies.

4 Adding terms

Now that we have efficient proof systems, we want to study their computational content. We will now add terms to these systems and study the behavior of these calculi.

For this section we will again restrict the calculus to a much weaker one where only the $\supset$ connective is allowed. This part of the internship is based on a recent work by

Brock-Nannestad and Guenot and Gustafsson [BNGG15]. We will first describe the terms added to the LJT and LJQ fragments, and then the unified system for LJF.

In each system, we will try to model a nested term of the form . We are interested in such a term because some of its representations implies sharing (see Fig. 2), and our goal is to investigate how focusing can express or not this aspect of the term.

If a term syntax allows for naming of subterms, such as the ones we are going to use, then it is possible to write the same term with different levels of sharing. But all these representations, with or without sharing should be equal after replacement of all named vars by their associated terms.

Between two "equivalent" representations of the same term the gain of compression can be very substantial. Our canonical example will be the term $\mathsf{f}(\mathsf{f}(\mathsf{a},\mathsf{a}),\mathsf{f}(\mathsf{a},\mathsf{a}))$, of type ι in the store $\Gamma = \{\mathsf{f} : \iota \supset \iota \supset \iota, \mathsf{a} : \iota\}$. As illustrated by Fig. 2, the size of its representation can either grow exponentially or linearly which is a considerable difference.

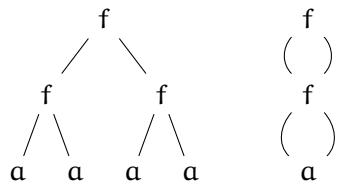


Figure 2: Two equivalent representations of the same term: a tree and a minimal *Directed acyclic graphs* (DAG[4]).

4.1 In LJT

By restricting LJF to atoms with negative polarity we get Herbelin's LJT system shown in Fig. 3.

$$\frac{}{\Gamma \Downarrow \mathsf{a} \vdash \varepsilon : \mathsf{a}}\,\mathrm{I} \qquad \frac{\Gamma, \mathsf{x} : \mathsf{A} \Downarrow \mathsf{A} \vdash \mathsf{k} : \mathsf{B}}{\Gamma, \mathsf{x} : \mathsf{A} \vdash \mathsf{x}\ \mathsf{k} : \mathsf{B}}\,\mathrm{D}$$

$$\frac{\Gamma \vdash \mathsf{u} : \mathsf{A} \qquad \Gamma \Downarrow \mathsf{B} \vdash \mathsf{k} : \mathsf{C}}{\Gamma \Downarrow \mathsf{A} \supset \mathsf{B} \vdash \mathsf{u} :: \mathsf{k} : \mathsf{C}}\,\supset_{\mathrm{l}} \qquad \frac{\Gamma, \mathsf{x} : \mathsf{A} \vdash \mathsf{t} : \mathsf{B}}{\Gamma \vdash \lambda \mathsf{x}.\mathsf{t} : \mathsf{A} \supset \mathsf{B}}\,\supset_{\mathrm{r}}$$

$$\mathsf{t}, \mathsf{u} ::= \mathsf{x}\ \mathsf{k} \mid \lambda \mathsf{x}.\mathsf{t}$$
$$\mathsf{k}, \mathsf{m} ::= \varepsilon \mid \mathsf{u} :: \mathsf{k}$$

Figure 3: The focused sequent calculus LJT.

Terms are divided in two syntactic categories, the new one being the representation of a list of arguments. In fact, applications $((\mathsf{f}\ \mathsf{a})\ \mathsf{b})\ \mathsf{c}$ are here denoted as $\mathsf{f}\ [\mathsf{a},\mathsf{b},\mathsf{c}]$. This form introduced by Herbelin [Her95a] is more suited to annotate sequent calculus.

In this calculus, the previously mentioned term is denoted

$$\mathsf{f}\ (\mathsf{f}\ (\mathsf{a}\ \varepsilon) :: (\mathsf{a}\ \varepsilon) :: \varepsilon) :: (\mathsf{f}\ (\mathsf{a}\ \varepsilon) :: (\mathsf{a}\ \varepsilon) :: \varepsilon) :: \varepsilon$$

[4]A directed graph with no directed cycles, formed by a collection of vertices and directed edges. DAGs naturally fit well the representation of terms with sharing.

which is not satisfying at all from the point of view of sharing, this encoding clearly corresponds to the first case of Fig. 2, the worst one. However, due to the use of the backward chaining strategy, it is not possible to obtain a more compressed encoding of our term in this framework.

4.2 In LJQ

By restricting LJF to atoms with positive polarity we get Lengrand's LJQ system [DL06] shown in Fig. 4:

$$\frac{}{A \in \Gamma \vdash x : A \Downarrow \cdot}\ \mathrm{I} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B \Downarrow \cdot}\ \supset_r$$

$$\frac{\Gamma \vdash p : A \Downarrow \cdot}{\Gamma \vdash \uparrow p : A}\ \mathrm{D} \qquad \frac{\Gamma \vdash p : A \Downarrow \cdot \qquad \Gamma, z : B \vdash t : C}{x : A \supset B \in \Gamma \vdash t[z = x\ p] : C}\ \supset_l$$

$$t, u ::= \uparrow p \mid t[z = x\ p]$$
$$p, q ::= x \mid \lambda x.t$$

Figure 4: The focused sequent calculus LJQ.

The philosophy of this calculus is very different, here every application pass through numerous explicit substitutions. In this setting, we can write our test term in the first fashion of Fig. 2: (without the heavy shifts (↑))

$$z_3[z_2 = z_1(z_6[z_6 = z_5\ a][z_5 = f\ a])][z_1 = f(z_4[z_4 = z_3\ a][z_3 = f\ a])]$$

but also in the second one:

$$z_4[z_4 = z_3\ z_2][z_3 = f\ z_2][z_2 = z_1\ a][z_1 = f\ a]$$

This calculus allows more proofs of the same formula, some are enormous, others are more efficient. Here sharing is feasible unlike in LJT. As LJQ is the restriction of LJF to positive atoms, such a shared term should be reproducible in LJF and we will do it in the next section.

It is important to remark that here sharing appears directly in *normal terms*. The explicit substitution $t[x/u]$ is different from the cut-full term $(\lambda x.t)\ u$.

It is interesting to have such "static" features in our calculus because it makes possible the description of structures having these features without having a dynamic behavior. We can imagine for example studying data structures with built-in sharing and parallelism properties.

4.3 Back to LJF

As we have seen, polarity choices have a major impact on the representation of terms and especially in matter of sharing. Extreme choices have extreme consequences, all negative atoms create no sharing and all positive atoms add a lot of unexpected naming of subterms.

These two behaviors can go together in a unique system which is LJF annotated with terms presented in the next figure.

$$\frac{\Gamma, x : N \Downarrow N \vdash k : E}{\Gamma, N \Uparrow \cdot \vdash \cdot \Uparrow x\ k : E}\ D_l \qquad \frac{\Gamma \Uparrow x : P \vdash \cdot \Uparrow t : E}{\Gamma \Downarrow P \vdash \kappa x.t : E}\ R_l \qquad \frac{x : C, \Gamma \Uparrow \Theta \vdash \mathcal{R}}{\Gamma \Uparrow x : C, \Theta \vdash \mathcal{R}}\ S_l$$

$$\frac{\Gamma \vdash p : P \Downarrow \cdot}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow \uparrow p : P}\ D_r \qquad \frac{\Gamma \Uparrow \cdot \vdash t : N \Uparrow \cdot}{\Gamma \vdash \downarrow t : N \Downarrow \cdot}\ R_r \qquad \frac{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow t : E}{\Gamma \Uparrow \cdot \vdash t : E \Uparrow \cdot}\ S_r$$

$$\frac{}{\Gamma \Downarrow a^- \vdash \varepsilon : a^-}\ I_l \qquad \frac{}{\Gamma, x : a^+ \vdash x : a^+ \Downarrow \cdot}\ I_r$$

$$\frac{\Gamma \Uparrow x : A \vdash t : B \Uparrow \cdot}{\Gamma \Uparrow \cdot \vdash \lambda x.t : A \supset B \Uparrow \cdot}\ \supset_r \qquad \frac{\Gamma \vdash p : A \Downarrow \cdot \qquad \Gamma \Downarrow B \vdash k : E}{\Gamma \Downarrow A \supset B \vdash p :: k : E}\ \supset_l$$

$$\begin{aligned} \text{Terms}: \quad & t, u ::= \lambda x.t \mid x\ k \mid \uparrow p \\ \text{Values}: \quad & p, q ::= x \mid \downarrow t \\ \text{Continuations}: \quad & k ::= \varepsilon \mid p :: k \mid \kappa x.t \end{aligned}$$

Figure 5: LJF with terms.

In this calculus there are two kinds of binders, λ and κ. The first one is the usual binder of λ-calculus while the other one can be understood as what we will call a *continuation* binder. It is used at the end of function application to store the resulting term, making it available in the rest of the term.

Reductions rules (extracted from the cut-elimination procedure) are numerous for this calculus and we will not recall all of them here. However it is important to understand how the continuation binder works. The two main rules of function application are the followings (in a relax and handy syntax):

$$(\lambda x.t)\ (q :: k) \to (t\{x/q\})\ k \tag{1}$$

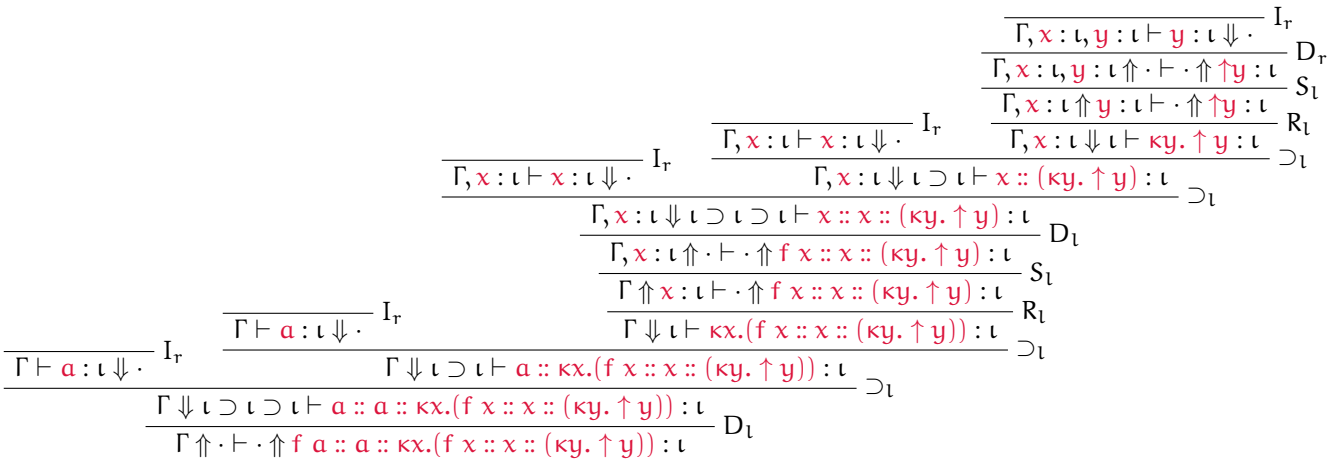
$$\uparrow q\ (\kappa x.t) \to t\{x/q\} \tag{2}$$

Rule (1) applies the first argument q of the list and is called several times until the list is empty. Then rule (2) propagates the result q of the computation to the rest of the term t. Here is an example of such a reduction:

$$(\lambda x.\uparrow x)\ (a :: (\kappa x.t)) \to \uparrow a\ (\kappa x.t) \to t\{x/a\}$$

As we are studying the cut-free segment of LJF, it does not really make sense to discuss term reduction, but it is important to understand how such continuations work, as this kind of term representations will be used in the remainder of this report.

In this system, our companion-term of Fig. 2 with positive ι atoms could be derived as follows:



As we can see the multi-ary applications are done in unique focusing phases and a continuation binder is created when the phase is exited (with a R_ι), allowing use of the constructed sub-term in the rest of the term.

This form is interesting because it is very similar to the *A-normal form*:

```
let x = f a a in
    let y = f x x in ↑y
```

As we will see, our work often resonates with compiling processes. Here, the apparition of A-normal forms is of interest because it as been showed in [FSDF93] that they could replace the *Continuation Passing Style* transformation in compilers: the same optimization remain possible, but the process of transforming code in A-normal form is more straightforward than the CPS transformation. This article has gone famous and A-normal forms are now an important object of study in the compilation community even if some more recent papers do advocate for the use of CPS [Ken07].

Furthermore it is relevant to remark that this framework works perfectly with higher order structure such as:

```
let x = f a (λz. let y = f z z in y) in ↑x
```

5 Multi-focusing and parallelism

As we have seen with the example of sharing, even if each of the previous systems are able to prove the same set of formulas, some offer more proofs of the same formula, which translates into more possibilities of sharing in terms. Now, we also want to be able to describe terms with parallel computations. It seems that this feature relate to the relatively new notion of *multi-focusing* proposed in [MS07].

5.1 Presentation

As we saw earlier there are a lot of proofs of the same formula in sequent calculus systems, and lot of them differ only from simple permutations of inferences rules. Other proof systems like natural deduction or proof nets capture more parallelism and don't have this flaw.

By using multi-focusing techniques Chaudhuri, Miller and Saurin addressed that issue to recover a form of canonicity within the cut-free sequent calculus itself [CMS08].

Their system allows focusing on multiple formulas in the same phase. They showed that selecting a "maximal focus" results in canonical proofs, in the same sense as in proof nets.

We are here interested in the impact of multi-focusing on terms. Indeed, as multi-focusing was introduced to be equivalent to heavily parallel frameworks such as proof-nets or expansion trees, it is natural to ask if this approach leads to terms involving a form of parallelism.

The system (restricted to implication only) that we are going to describe is a new formulation of the one in [CMS08] which decompose the process of focusing.

$$\frac{\Gamma \Downarrow \cdot ; \cdot \vdash t : E}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow t : E}\ \mathsf{Co} \qquad \frac{\Gamma, f : N \Downarrow \Theta ; N \vdash (\cdot\ \mathcal{A}) :: \mathcal{L} : E}{\Gamma, f : N \Downarrow \Theta ; \cdot \vdash (f\ \mathcal{A}) :: \mathcal{L} : E}\ \mathsf{D_l} \qquad \frac{x : C, \Gamma \Uparrow \Theta \vdash t : \mathcal{R}}{\Gamma \Uparrow C, \Theta \vdash \kappa x.t : \mathcal{R}}\ \mathsf{S_l}$$

$$\frac{\Gamma \Downarrow \Theta, P ; \cdot \vdash \mathcal{L} : E}{\Gamma \Downarrow \Theta ; P \vdash (\cdot\ \varepsilon) :: \mathcal{L} : E}\ \mathsf{Cl} \qquad \frac{\Gamma \vdash t : P \Downarrow \cdot}{\Gamma \Uparrow \cdot ; \cdot \vdash t : P \Uparrow}\ \mathsf{D_r} \qquad \frac{\Gamma \Uparrow \mathcal{P} \vdash \cdot \Uparrow t : E}{\Gamma \Downarrow \mathcal{P} ; \cdot \vdash t : E}\ \mathsf{R_l} \qquad \frac{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow t : E}{\Gamma \Uparrow \cdot \vdash t : E \Uparrow \cdot}\ \mathsf{S_r}$$

Figure 6: Structural rules of the multi-focusing system MLJF

For this purpose the left-focused zone is now divided in two parts separated by a semicolon and the asynchronous phase is decomposed in small pieces:

- It starts with the *commence* rule Co
- Then a first focus is selected with $\mathsf{D_l}$ and put on the right of the semicolon
- Next a series of $\supset_l$ rules is applied until the focus turns positive
- Then *classify* is called, moving the positive formula to the left of the semicolon
- Therewith we can select a new focus or end the synchronous phase with a release $\mathsf{R_l}$ and a series of stores $\mathsf{S_l}$.

$$\frac{\Gamma \vdash t : A \Downarrow \cdot \qquad \Gamma \Downarrow \mathcal{P} ; B, \Theta \vdash (\cdot\ \mathcal{A}) :: \mathcal{L} : E}{\Gamma \Downarrow \mathcal{P} ; A \supset B, \Theta \vdash (\cdot\ t :: \mathcal{A}) :: \mathcal{L} : E}\ \supset_l \qquad \frac{\Gamma \Uparrow x : A \vdash t : B \Uparrow \cdot}{\Gamma \Uparrow \cdot \vdash \lambda x.t : A \supset B \Uparrow \cdot}\ \supset_r \qquad \frac{}{\Gamma, x : a^+ \vdash x : a^+ \Downarrow \cdot}\ \mathsf{I_r}$$

Figure 7: Other rules of MLJF

The terms of this calculus are described by the following grammar:

$$\begin{aligned} \text{Terms}: \quad & t, u ::= x \in \mathcal{X} \mid \lambda x.t \\ \text{Arguments}: \quad & \mathcal{A} ::= \varepsilon \mid t :: \mathcal{A} \\ \text{Continuations}: \quad & \mathcal{L} ::= \kappa x.t \mid (f\ \mathcal{A}) :: \mathcal{L} \mid (\cdot\ \mathcal{A}) :: \mathcal{L} \end{aligned}$$

Again an extensive use of lists is made, as it fits well the construction of multi-focused proofs.

5.2 An example

In this calculus the term

```
let x1 = f a a and x2 = f b b in
    let x3 = f x1 x2 in
        x3
```

is encoded as:

$$(f\ a :: a :: \varepsilon) :: (f\ b :: b :: \varepsilon) :: \kappa x_1.\kappa x_2((f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3)$$

and in the store $\Gamma = \{f : \iota \supset \iota \supset \iota, a : \iota, b : \iota\}$, it has type ι. The proof associated to this term illustrates well the sequential introduction of foci during multi-focusing and how continuations are created only at the end of the multi-focused phase, with the two successive stores:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash a : \iota \Downarrow \cdot}\,I_r \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, x_1, x_2, x_3 : \iota \vdash x_3 : \iota \Downarrow \cdot}\,I_r}{\Gamma, x_1, x_2 \Uparrow \iota; \cdot \vdash \cdot \Uparrow \kappa x_3.x_3 : \iota}\,S_l + D_r}{\Gamma, x_1, x_2 \Downarrow \cdot; \iota \vdash (\cdot\ \varepsilon) :: \kappa x_3.x_3 : \iota}\,Cl + R_l}{\Gamma, x_1, x_2 : \iota \Uparrow \cdot \vdash \cdot \Uparrow (f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3 : \iota}\,Co + D_l + 2\times \supset_l}{\Gamma, x_1 : \iota \Uparrow \iota \vdash \cdot \Uparrow \kappa x_2.((f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3) : \iota}\,S_l}{\Gamma \Uparrow \iota, \iota \vdash \cdot \Uparrow \kappa x_1.\kappa x_2.((f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3) : \iota}\,S_l}{\Gamma \Downarrow \iota, \iota; \cdot \vdash \kappa x_1.\kappa x_2.((f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3) : \iota}\,R_l}{\Gamma \Downarrow \iota; \iota \vdash (\cdot\ \varepsilon) :: \kappa x_1.\kappa x_2.((f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3) : \iota}\,Cl}{\Gamma \Downarrow \iota; \cdot \vdash (f\ b :: b :: \varepsilon) :: \kappa x_1.\kappa x_2.((f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3) : \iota}\,D_l + 2\times \supset_l \qquad \dfrac{\dfrac{}{\Gamma \vdash a : \iota \Downarrow \cdot}\,I_r}{\Gamma \Downarrow \cdot; \iota \vdash (\cdot\ \varepsilon) :: (f\ b :: b :: \varepsilon) :: \kappa x_1.\kappa x_2.((f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3) : \iota}\,Cl}{\Gamma \Downarrow \cdot; \iota \supset \iota \vdash (\cdot\ a :: \varepsilon) :: (f\ b :: b :: \varepsilon) :: \kappa x_1.\kappa x_2.((f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3) : \iota}\,\supset_l}{\Gamma \Downarrow \cdot; \iota \supset \iota \supset \iota \vdash (\cdot\ a :: a :: \varepsilon) :: (f\ b :: b :: \varepsilon) :: \kappa x_1.\kappa x_2.((f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3) : \iota}\,D_l}{\Gamma \Downarrow \cdot; \cdot \vdash (f\ a :: a :: \varepsilon) :: (f\ b :: b :: \varepsilon) :: \kappa x_1.\kappa x_2.((f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3) : \iota}\,Co}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow (f\ a :: a :: \varepsilon) :: (f\ b :: b :: \varepsilon) :: \kappa x_1.\kappa x_2.((f\ x_1 :: x_2 :: \varepsilon) :: \kappa x_3.x_3) : \iota}$$

6 Examples worthy of interest

The following section mostly relates to work in progress at the time of writing. Its purpose is to underline the richness of the questioning related to this internship's subject. As some links with compiling optimizations appeared to be relevant and will be discussed here, readers not familiar with them can skim through [AND98] and [Muc97] for classical data-and-control-flow analysis algorithms.

6.1 Non-vacuous check

Now that we have terms enhanced with sharing ability, we may want to check that each named term is used at least once. It is a well known problem in compiler optimization: the code author or previous optimizations may have introduced unused variables and such useless code is eliminated via an iterative algorithm.

Such a check can also be performed on the proof associated to the term by rigorously checking that each variable added to the context is used at least once. Indeed a proof search algorithm could easily be enhanced to check that property by keeping a record of used variables.

This aspect also appeared to be directly related to linear logic in which it is possible to maintain a fine control of resources. Thus it would be interesting to give a term representation to LLF, a focused sequent calculus for linear logic, and to see if this non-vacuous property can be guaranteed by the proof-theory itself.

6.2 Maximal multi-focusing

In [CMS08] the authors introduce a global property named *maximal multi-focusing*. In a maximally multi-focused proof no focused phases can be moved down and merged with a lower focused phase. The hope is that terms associated to maximally multi-focused proofs indeed are as "parallel" as possible. Furthermore, as every term naming has been pushed as down as possible this could simplify a lot the search for vacuous assignments.

Furthermore, one could want to check if a term is "maximally shared", hence, if the proof associated to it is maximally multi-focused. To that intend an algorithm to turn a multi-focused proof to a maximal one would be useful. For now no efficient algorithm exists since only the existence of such proofs is formalized in the literature. The relation with terms, enhanced with parallelism, may be an inspiration to create an optimal algorithm as this problem is very similar to the common subexpression elimination procedure of compilers.

6.3 Equality check

Term equality is a complex problem. One should first ask themselves what kind of equality they want to achieve. Syntactic equality is often the simplest one to check, but is it a good criterion? In fact, in a term with sharing, two nested let-expressions may be permuted if none depends on the other. Similarly, in a term with parallelism bindings under the same let-expression can be treated in any order. Furthermore, one might imagine to unfold terms with let-expression by performing the substitutions before checking equality.

During the internship, several experiments have been carried out in λProlog , implementing simple algorithms to check term equality with permutations and without unfolding. An example of a λProlog program testing such equality is present in annexe A. This algorithm is neither the most intelligent nor the quickest, but it shows that λProlog is a very good language for prototyping algorithms about logic and term representations.

But the major progress may once again come from maximally multi-focused proofs as they, like proof-nets, get rid of a lot of syntactic bureaucracy. This directly tackles the obstacle of proof identity criteria and may play a role in understanding problems such as the concurrent equality of CLF [WCPW03].

6.4 Prospects in ProofCert

The ProofCert machinery makes use of LJF and LKF (LJF's classical counterpart) to develop what is called *Foundational Proof Certificates* [CM15]. The main idea is that the kernel being LJF or LKF, users will be able to align the inference rules of their own systems with phases of the kernel system.

The certificates may have many forms and even λ-terms can be used in this framework. Indeed, as we have seen, a term encodes a proof of its type. Such a checker has

already been described for simply-typed terms in βη-long normal form in a not-yet-published paper by Chihani and Miller [CM15].

But some more unexpected usages of terms could be made. The ProofCert project has nearly reached its first goal: providing a framework to check proofs for correctness with any number of implementations of proof checkers. The next big thing would be to structure libraries of theorems and proofs that would allow them to be retrieved, further analyzed and rechecked by skeptical users. Such libraries would greatly benefit from efficient sharing features to manage the uncountable number of dependencies of complex theorems. One way to tackle this issue could be to use terms as libraries, and of course terms with sharing and concurrent abilities such as the ones described in this report.

7 Conclusion

Term representations and proof theory have a long common history, and this enriching relationship does not seems to have exhausted all its resources. Focused proof systems may lead to new typing system as some features – such as term sharing and parallelism – of the associated calculi emerge quite naturally from them. Indeed, as we have seen, LJF allows for the encoding of terms with sharing abilities, and its multi-focused version also adds support for parallel structures in a very straightforward way.

The inquiry led during this internship exposes clearly the tight links uniting term representations and sequent calculi. The chaotic essence of sequent calculi inference rules actually gives a great deal of liberty to model term calculi by restricting them with focusing. It is most likely that there is still much to learn about terms from logic and vice versa.

The calculi proposed during this internship do not enjoy formal proofs and this will be the first of numerous future works. Validating the calculi may be done by studying the cut-elimination processes of associated logic systems, extracting from them the reduction rules of the calculi and finally checking that the semantics is the one we are waiting for.

References

[And92] Jean-Marc Andreoli. Logic Programming with Focusing Proofs in Linear Logic. *Journal of Logic and Computation*, 2(3):297–347, June 1992.

[AND98] W. APPEL ANDREW. *Modern Compiler Implementation in ML*. Cambridge University Press, 1998.

[Bar93] Henk Barendregt. *Lambda Calculi with Types*, volume 2 of *Handbook of Logic in Computer Science*. 1993.

[BG] Henk Barendregt and Silvia Ghilezan. *Lambda Terms for Natural Deduction, Sequent Calculus and Cut Elimination*.

[BNGG15] Taus Brock-Nannestad, Nicolas Guenot, and Daniel Gustafsson. Computation in focused intuitionistic logic. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming*, pages 43–54. ACM, 2015.

[CHM13] Kaustuv Chaudhuri, Stefan Hetzl, and Dale Miller. *A Multi-Focused Proof System Isomorphic to Expansion Proofs*. 2013.

[Chu40] Alonzo Church. A Formulation of the Simple Theory of Types. *The Journal of Symbolic Logic*, 5(2):56–68, June 1940.

[CM15] Zakaria Chihani and Dale Miller. A semantic framework for proof evidence. *Journal of Automated Reasoning*, 2015. Draft.

[CMS08] Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin. Canonical Sequent Proofs via Multi-Focusing. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and Luke Ong, editors, *Fifth Ifip International Conference On Theoretical Computer Science – Tcs 2008*, number 273 in IFIP International Federation for Information Processing, pages 383–396. Springer US, 2008.

[DL06] Roy Dyckhoff and Stéphane Lengrand. LJQ: A Strongly Focused Calculus for Intuitionistic Logic. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Logical Approaches to Computational Barriers*, number 3988 in Lecture Notes in Computer Science, pages 173–185. Springer Berlin Heidelberg, 2006.

[FSDF93] Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The Essence of Compiling with Continuations. In *Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation*, PLDI '93, pages 237–247, New York, NY, USA, 1993. ACM.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[Her95a] Hugo Herbelin. *Sequents qu'on calcule : de l'interpretation du calcul des sequents comme calcul de lambda-termes et comme calcul de strategies gagnantes*. PhD thesis, Paris 7, January 1995.

[Her95b] Hugo Herbelin. A -calculus structure isomorphic to Gentzen-style sequent calculus structure. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic*, number 933 in Lecture Notes in Computer Science, pages 61–75. Springer Berlin Heidelberg, 1995.

[Ken07] Andrew Kennedy. Compiling with Continuations, Continued. In *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming*, ICFP '07, pages 177–190, New York, NY, USA, 2007. ACM.

[LM09] Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009.

[MN12] Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, 2012.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51(1-2):125–157, March 1991.

[MS07] Dale Miller and Alexis Saurin. From proofs to focused proofs: a modular proof of focalization in linear logic. In *CSL 2007: Computer Science Logic, volume 4646 of LNCS*, pages 405–419. Springer-Verlag, 2007.

[Muc97] Steven S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufmann, 1997.

[WCPW03] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A Concurrent Logical Framework: The Propositional Fragment. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs*, number 3085 in Lecture Notes in Computer Science, pages 355–377. Springer Berlin Heidelberg, April 2003.

A Equality test in λProlog

The signature

```
sig permLets.
%Libasic contains very primitive addons to LProlog such as pairs and list utilities
accum_sig libasic.

kind var, cst, tm, cont, ty type.


% The constants of our calculus:
type a, b tm.
type f, g  tm.

% The vars, applications and abstractions:
type coer     var -> tm.
type app      tm -> list tm -> tm.
type abs      (var -> tm) -> tm.

% The parallel lets:
type body     tm -> cont.
type loc      (var -> cont) -> cont.
type parlet   list tm -> cont -> tm.

% The lotfcontaux method:
type priv var -> o.
type assoc var -> nat -> o.
type lofcontaux  nat -> cont -> list nat -> o.

% The equality algorithm:
type eqAux list tm -> list nat -> list tm -> list nat -> o.
type eq2 tm -> tm -> o.
```

The module

```
module permLets.
accumulate libasic.

% lofcontaux numbers the variables declared in a parallel let expression:
lofcontaux _ (body b) [].
lofcontaux _ (body a) [].
lofcontaux _ (body f) [].
```

```
lofcontaux _ (body g) [].
lofcontaux _ (body (coer X)) [N] :- not (priv X), assoc X N.

lofcontaux N (body (app A B)) L :-
           enum B T, lofcontaux N (body T) LB,
           lofcontaux N (body A) LA,
           concat LA LB L.

lofcontaux N (body (abs A)) L :-
           pi x\ priv x => lofcontaux N (body (A x)) L.

lofcontaux N (loc Loc) L :-
         pi x\ assoc x N =>  lofcontaux (nat_s N) (Loc x) L.

% eqAux Checks equality of each subterms named in the let:
eqAux _ [] _ [].
eqAux LA1 (N1::L1) LA2 (N2::L2) :-
      extract LA1 N1 U, extract LA2 N2 V,
      eq2 U V, eqAux LA1 L1 LA2 L2.

% Finally eq2 checks the equality between two terms with parallel lets:
eq2 a a.
eq2 b b.
eq2 f f.
eq2 g g.

eq2 (abs A1) (abs A2) :-
    % Here a higher-order feature of LProlog is used to take care of bound variables.
    pi x\ eq2 (A1 x) (A2 x).

eq2 (app U LA1) (app V LA2) :-
    eq2 U V,
    list_eq LA1 LA2 eq2. % Checks the equality of two lists

eq2 (parlet LA1 B1) (parlet LA2 B2) :-
   list_perm LA1 LA2, % Checks if LA2 is a permutation of LA1
   lofcontaux nat_z B1 L1,
   lofcontaux nat_z B2 L2,
   eqAux LA1 L1 LA2 L2.
```