Certified Proof Search for Intuitionistic Linear Logic

Guillaume Allais[1] **and Conor McBride**[1]

1 **University of Strathclyde**
Glasgow, Scotland
{guillaume.allais, conor.mcbride}@strath.ac.uk

Abstract

In this article we show the difficulties a type-theorist may face when attempting to formalise a decidability result described informally. We then demonstrate how generalising the problem and switching to a more structured presentation can alleviate her suffering.

The calculus we target is a fragment of Intuitionistic Linear Logic (ILL onwards) and the tool we use to construct the search procedure is Agda (but any reasonable type theory equipped with inductive families would do). The example is simple but already powerful enough to derive a solver for equations over a commutative monoid from a restriction of it.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Agda, Proof Search, Linear Logic, Certified programming

1 Introduction

Type theory [15] equipped with inductive families [9] is expressive enough that one can implement *certified* proof search algorithms which are not merely oracles outputting a one bit answer but full-blown automated provers producing derivations which are statically known to be correct [5, 18]. It is only natural to delve into the literature to try and find decidability proofs which, through the Curry-Howard correspondence, could make good candidates for mechanisation (see e.g. Pierre Crégut's work on Presburger arithmetic [6]). Reality is however not as welcoming as one would hope: most of these proofs have not been formulated with mechanisation in mind and would require a huge effort to be ported *as is* in your favourite theorem prover.

In this article, we argue that it would indeed be a grave mistake to implement them *as is* and that type-theorists should aim to develop better-structured algorithms. We show, working on a fragment of ILL [11], the sort of pitfalls to avoid and the generic ideas leading to better-behaved formulations.

In section 2 we describe the fragment of ILL we are studying; section 3 defines a more general calculus internalising the notion of leftovers thus making the informal description of the proof search mechanism formal; and section 4 introduces resource-aware contexts therefore giving us a powerful language to target with our proof search algorithm implemented in section 7. The soundness and completeness results proved respectively in section 6 and section 5 are what let us recover a proof of the decidability of the ILL fragment considered from the one of the more general system. Finally, section 8 presents an application of this proof search procedure to automatically discharge equations over a commutative monoid. This solver is then further specialised to proving that two lists are bag equivalent thus integrating really well with Danielsson's previous work [7].

2 The Calculus, Informally

Our whole development is parametrised by a type of atomic proposition Pr on which we do not put any constraint except that equality of its inhabitants should be decidable. We name $\_\stackrel{?}{=}\_$ the function

of type $(p\ q : Pr) \to \mathsf{Dec}\ (p \equiv q)$ witnessing this property.

The calculus we are considering is a fragment of Intuitionistic Linear Logic composed of *atomic* types (lifting Pr), *tensor* and *with* products. This is summed up by the following grammar for types:

$$\mathsf{ty} ::= \kappa\ Pr \mid \mathsf{ty} \otimes \mathsf{ty} \mid \mathsf{ty}\ \&\ \mathsf{ty}$$

The calculus' sequents ($\Gamma \vdash \sigma$) are composed of a multiset of types (Γ) describing the resources available in the context and a type (σ) corresponding to the proposition one is trying to prove. Each type constructor comes with both introduction and elimination rules (also known as, respectively, right and left rules because of the side of the sequent they affect) described in Figure 1. Multisets are intrinsically extensional hence the lack of a permutation rule one may be used to seeing in various list-based presentations.

$$\frac{}{\{\!|\ \sigma\ |\!\} \vdash \sigma}\,ax \qquad \frac{\Gamma \vdash \sigma \qquad \Delta \vdash \tau}{\Gamma \uplus \Delta \vdash \sigma \otimes \tau}\otimes^r \qquad \frac{\Gamma \uplus \{\!|\ \sigma, \tau\ |\!\} \vdash \upsilon}{\Gamma \uplus \{\!|\ \sigma \otimes \tau\ |\!\} \vdash \upsilon}\otimes^l$$

$$\frac{\Gamma \vdash \sigma \qquad \Gamma \vdash \tau}{\Gamma \vdash \sigma\ \&\ \tau}\,\&^r \qquad \frac{\Gamma \uplus \{\!|\ \sigma\ |\!\} \vdash \upsilon}{\Gamma \uplus \{\!|\ \sigma\ \&\ \tau\ |\!\} \vdash \upsilon}\,\&^l_1 \qquad \frac{\Gamma \uplus \{\!|\ \tau\ |\!\} \vdash \upsilon}{\Gamma \uplus \{\!|\ \sigma\ \&\ \tau\ |\!\} \vdash \upsilon}\,\&^l_2$$

Figure 1 Introduction and Elimination rules for ILL

However these rules are far from algorithmic: the logician needs to *guess* when to apply an elimination rule or which partition of the current context to pick when introducing a tensor. This makes this calculus really ill-designed for her to perform a proof search in a sensible manner. So, rather than sticking to the original presentation and trying to work around the inconvenience of dealing with rules which are not algorithmic and intrinsically extensional notions such as the one of multisets, it is possible to generalise the calculus in order to have a more palatable formal treatment.

The principal insight in this development is that proof search in Linear Logic is not just about fully using the context provided to us as an input in order to discharge a goal. The bulk of the work is rather to use parts of some of the assumptions in a context to discharge a first subgoal; collect the leftovers and invest them into trying to discharge another subproblem. Only in the end should the leftovers be down to nothing. This observation leads to the definition of two new notions: first, the calculus is generalised to one internalising the notion of leftovers; second, the contexts are made resource-aware meaning that they keep the same structure whilst tracking whether (parts of) an assumption has been used already. Proof search becomes consumption annotation.

3 Generalising the Problem

In this section, we will start by studying a simple example showcasing the role the idea of leftovers plays during proof search before diving into the implementation details of such concepts.

3.1 Example

Let us study how one would describe the process of running a proof search algorithm for our fragment of ILL. The intermediate data structures, despite looking similar to usual ILL sequents, are not quite valid proof trees as we purposefully ignore left rules. We write

$$\Delta \Rightarrow \frac{\pi}{\Gamma \vdash \sigma}$$

to mean that the current proof search state is Δ and we managed to build a pseudo-derivation π of type $\Gamma \vdash \sigma$. π and Γ may be replaced by question marks when we haven't yet reached a point where we have a found proof and thus instatiated them.

In order to materialise the idea that some resources in Δ are available whereas others have already been consumed, we are going to mark with a box $\boxed{\ }$ (the parts of) the assumptions which are currently available. During the proof search, the state Δ will keep its structure but we will update destructively its resource annotations. For instance, consuming σ out of $\Delta = \boxed{(\sigma \ \& \ \tau)} \otimes \upsilon$ will turn Δ into $(\sigma \ \& \ \boxed{\tau}) \otimes \upsilon$.

Let us now go ahead and observe how one looks for a proof of the following formula (where σ and τ are assumed to be atomic): $(\sigma \otimes \tau) \ \& \ \sigma \vdash \tau \otimes \sigma$. The proof search problem we are facing is therefore:

$$\boxed{(\sigma \otimes \tau) \ \& \ \sigma} \Rightarrow \frac{?}{? \vdash \tau \otimes \sigma}$$

The goal's head symbol is a $\otimes$; as we have no interest in guessing whether to apply left rules—if at all necessary—, or how to partition the current context, we are simply going to start by looking for a proof of its left subcomponent using the full context. Given that τ is an atomic formula, the only way for us to discharge this goal is to use an assumption available in the context. Fortunately, there is a τ in the context; we are therefore able to produce a derivation where τ has now been consumed. In terms of our proof search, this is expressed by using an axiom rule and destructively updating the context:

$$\boxed{(\sigma \otimes \tau) \ \& \ \sigma} \Rightarrow \frac{?}{? \vdash \tau} \qquad \rightsquigarrow \qquad (\boxed{\sigma} \otimes \tau) \ \& \ \boxed{\sigma} \Rightarrow \frac{}{\tau \vdash \tau}\, ax$$

Now that we are done with the left subgoal, we can deal with the right one using the leftovers $(\boxed{\sigma} \otimes \tau) \ \& \ \boxed{\sigma}$. We are once more facing an atomic formula which we can only discharge by using an assumption. This time there are two candidates in the context except that one of them is inaccessible: solving the previous goal has had the side-effect of picking one side of the & thus rejecting the other entirely. In other words: a left rule has been applied implicitly! The only meaningful step in the proof search is therefore:

$$(\boxed{\sigma} \otimes \tau) \ \& \ \boxed{\sigma} \Rightarrow \frac{?}{? \vdash \sigma} \qquad \rightsquigarrow \qquad (\sigma \otimes \tau) \ \& \ \boxed{\sigma} \Rightarrow \frac{}{\sigma \vdash \sigma}\, ax$$

We can then come back to our $\otimes$-headed goal and combine these two derivations by using a right introduction rule for $\otimes$. $(\sigma \otimes \tau) \ \& \ \boxed{\sigma}$ being a fully used context ($\boxed{\sigma}$ is inaccessible), we can conclude that our search has ended successfully:

$$(\sigma \otimes \tau) \ \& \ \boxed{\sigma} \Rightarrow \frac{\dfrac{}{\tau \vdash \tau}\, ax \qquad \dfrac{}{\sigma \vdash \sigma}\, ax}{(\sigma \otimes \tau) \ \& \ \sigma \vdash \tau \otimes \sigma}\, \otimes^r$$

The fact that the whole context is used by the end of the search tells us that this should translate into a valid ILL proof tree. And it is indeed the case: by following the structure of the pseudo-proof

we just generated above and adding the required left rules[1], we get the following derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\tau \vdash \tau}\,ax \qquad \dfrac{}{\sigma \vdash \sigma}\,ax}{\sigma, \tau \vdash \tau \otimes \sigma}\otimes^r}{\sigma \otimes \tau \vdash \tau \otimes \sigma}\otimes^l}{(\sigma \otimes \tau) \,\&\, \sigma \vdash \tau \otimes \sigma}\,\&_1^l$$

3.2 A Calculus with Leftovers

This observation of a proof search algorithm in action leads us to the definition of a three place relation describing the new calculus where the notion of leftovers from a subproof is internalised. When we write down the sequent $\Gamma \vdash \sigma \boxtimes \Delta$, we mean that from the input Γ, we can prove σ with leftovers Δ. Let us see what a linear calculus would look like in this setting.

If we assume that we already have in our possession a similar relation $\Gamma \ni k \boxtimes \Delta$ describing the act of consuming a resource $\kappa\ k$[2] from a context Γ with leftovers Δ, then the axiom rule translates to:

$$\dfrac{\Gamma \ni k \boxtimes \Delta}{\Gamma \vdash \kappa\ k \boxtimes \Delta}\,ax$$

The introduction rule for tensor in the system with leftovers does not involve partitioning a multiset (a list in our implementation) anymore: one starts by discharging the first subgoal, collects the leftovers from this computation, and then feeds them to the procedure now working on the second subgoal.

$$\dfrac{\Gamma \vdash \sigma \boxtimes \Delta \qquad \Delta \vdash \tau \boxtimes E}{\Gamma \vdash \sigma \otimes \tau \boxtimes E}$$

This is a left-skewed presentation but could just as well be a right-skewed one. We also discuss (in subsection 9.2) the opportunity for parallelisation of the proof search a symmetric version could offer as well as the additional costs it would entail.

The with type constructor on the other hand expects both subgoals to be proven using the same resources. We formalise this as the fact that both sides are proved using the input context and that both leftovers are then synchronised (for a sensible, yet to be defined, definition of synchronisation). Obviously, not all leftovers will be synchronisable: checking whether they are may reject proof candidates which are not compatible.

$$\dfrac{\Gamma \vdash \sigma \boxtimes \Delta_1 \qquad \Gamma \vdash \tau \boxtimes \Delta_2 \qquad \Delta \equiv \Delta_1 \odot \Delta_2}{\Gamma \vdash \sigma \,\&\, \tau \boxtimes \Delta}$$

We can now rewrite (see Figure 2) the proof described earlier in a fashion which distinguishes between the state of the context before one starts proving a goal and after it has been discharged entirely.

It should not come as a surprise that this calculus does not have any elimination rule for the various type constructors: elimination rules do not consume anything, they merely shuffle around (parts of) assumptions in the context and are, as a consequence, not interesting proof steps. These are therefore

[1] We will explain in section 6 how deciding where these left rules should go can be done automatically.

[2] In this presentation, we limit the axiom rule to atomic formulas only but it is not an issue: it is a well-known fact that an axiom rule for any formula is admissible by a simple induction on the formula's structure.

$$\dfrac{\dfrac{}{\boxed{(\sigma \otimes \tau)\ \&\ \sigma} \vdash \tau \boxtimes (\boxed{\sigma} \otimes \tau)\ \&\ \boxed{\sigma}}\,ax \qquad \dfrac{}{(\boxed{\sigma} \otimes \tau)\ \&\ \boxed{\sigma} \vdash \sigma \boxtimes (\sigma \otimes \tau)\ \&\ \boxed{\sigma}}\,ax}{\boxed{(\sigma \otimes \tau)\ \&\ \sigma} \vdash \tau \otimes \sigma \boxtimes (\sigma \otimes \tau)\ \&\ \boxed{\sigma}}\,\otimes^r$$

Figure 2 A proof with input / output contexts and usage annotations

$$\frac{}{\kappa\ k : \mathsf{Cover}\ \kappa\ k} \qquad \frac{S : \mathsf{Cover}\ \sigma \qquad T : \mathsf{Cover}\ \tau}{S \otimes T : \mathsf{Cover}\ \sigma \otimes \tau} \qquad \frac{S : \mathsf{Cover}\ \sigma}{S \otimes [\ \tau\ ] : \mathsf{Cover}\ \sigma \otimes \tau}$$

$$\frac{T : \mathsf{Cover}\ \tau}{[\ \sigma\ ] \otimes T : \mathsf{Cover}\ \sigma \otimes \tau}$$

$$\frac{}{\sigma\ \&\ \tau : \mathsf{Cover}\ \sigma\ \&\ \tau} \qquad \frac{\mathrm{S} : \mathsf{Cover}\ \sigma}{S\ \&[\ \tau\ ] : \mathsf{Cover}\ \sigma\ \&\ \tau} \qquad \frac{\mathrm{T} : \mathsf{Cover}\ \tau}{[\ \sigma\ ]\ \& T : \mathsf{Cover}\ \sigma\ \&\ \tau}$$

Figure 3 The Cover datatype

implicit in the process. This remark resonates a lot with Andreoli's definition of focusing [2] whose goal was to prune the search space by declaring that the logician does not care about the order in which some commuting rules are applied.

Ultimately, these rules being implicit is not an issue as witnessed by the fact that the soundness result we give in section 6 is constructive: we can mechanically decide where to optimally insert the appropriate left rules for the ILL derivation to be correct.

4 Keeping the Structure

We now have a calculus with input and output contexts; but there is no material artefact describing the relationship between these two. Sure, we could prove a lemma stating that the leftovers are precisely the subset of the input context which has not been used to discharge the goal but the proof would be quite involved because, among other things, of the merge operation hidden in the tensor rule.

But this is only difficult because we have forgotten the structure of the problem and are still dealing with rather extensional notions. Indeed, all of these intermediate contexts are just *the* one handed over to us when starting the proof search procedure except that they come with an usage annotation describing whether the various assumptions are still available or have already been consumed. This is the intuition we used in our example in subsection 3.1 when marking available resources with a box $\boxed{\ }$ and keeping used ones rather than simply dropping them from the context and that is made fully explicit in Figure 2.

4.1 Resource-Aware Contexts

Let us make this all more formal. We start by defining Covers: given a type σ, a cover S is a formal object describing precisely which (non-empty) set of parts of σ has been consumed already. The set of covers of a type σ is represented by an inductive family $\mathsf{Cover}\ \sigma$ listing all the different ways in which σ may be partially used. The introduction rules, which are listed in Figure 3, can be justified in the following manner: The cover for an atomic proposition can only be one thing: the atom itself;

In the case of a tensor, both subparts can be partially used (cf. $S \otimes T$) or it may be the case that only one side has been dug into so far (cf. $S \otimes[\ \tau\ ]$ and $[\ \sigma\ ] \otimes T$);

Similarly, a cover for a with-headed assumption can be a choice of a side (cf. $S\ \&[\ \tau\ ]$ and $[\ \sigma\ ]\&\ T$). Or, more surprisingly, it can be a full cover (cf. $\sigma\ \&\ \tau$) which is saying that *both* sides will be entirely used in different subtrees. This sort of full cover is only ever created when synchronising two output contexts by using a with introduction rule as in the following example:

$$\dfrac{\dfrac{}{\boxed{\sigma\ \&\ \tau} \vdash \tau \boxtimes \boxed{\sigma}\ \&\ \boxed{\tau}}\,ax \qquad \dfrac{}{\boxed{\sigma\ \&\ \tau} \vdash \sigma \boxtimes \sigma\ \&\ \boxed{\tau}}\,ax}{\boxed{\sigma\ \&\ \tau} \vdash \tau\ \&\ \sigma \boxtimes \sigma\ \&\ \tau}\,\&^r$$

The Usage of a type σ is directly based on the idea of a cover; it describes two different situations: either the assumption has not been touched yet or it has been (partially) used. Hence Usage is the following datatype with two infix constructors[3]:

$$\frac{}{[\ \sigma\ ] : \mathsf{Usage}\ \sigma} \qquad \frac{S : \mathsf{Cover}\ \sigma}{]\ S\ [: \mathsf{Usage}\ \sigma}$$

Finally, we can extend the definition of Usage to contexts by a simple pointwise lifting. We call this lifting Usages to retain the connection between the two whilst avoiding any ambiguities.

$$\frac{}{\varepsilon : \mathsf{Usages}\ \varepsilon} \qquad \frac{\Gamma : \mathsf{Usages}\ \gamma \qquad S : \mathsf{Usage}\ \sigma}{\Gamma \bullet S : \mathsf{Usages}\ \gamma \bullet \sigma}$$

4.1.0.1 Erasures

From an Usage(s), one can always define an erasure function building a context listing the hypotheses marked as used. We write $\lceil\ \_\ \rfloor$ for such functions and define them by induction on the structure of the Usage(s).

4.1.0.2 Injection

We call inj the function taking a context γ as argument and describing the Usages γ corresponding to a completely mint context.

4.2 Being Synchronised, Formally

Now that Usages have been introduced, we can give a formal treatment of the notion of synchronisation we evoked when giving the with introduction rule for the calculus with leftovers. Synchronisation is meant to say that the two Usages are equal modulo some inconsequential variations. These inconsequential variations partly correspond to the fact that left rules may be inserted at different places in different subtrees.

Synchronisation is a three place relation $\Delta \equiv \Delta_1 \odot \Delta_2$ defined as the pointwise lifting of an analogous one working on Covers. Let us study the latter one which is defined in an inductive manner.

It is reflexive which means that its diagonal $S \equiv S \odot S$ is always inhabited. For the sake of simplicity, we do not add a constructor for reflexivity: this rule is admissible by induction on S based on the fact that synchronisation for covers comes with all the structural rules one would expect:

[3] The way the brackets are used is meant to convey the idea that $[\ \sigma\ ]$ is in mint condition whilst $]\ S\ [$ is dented. The box describing an hypothesis in mint conditions is naturally mimicking the $\boxed{\ }$ we have written earlier on.

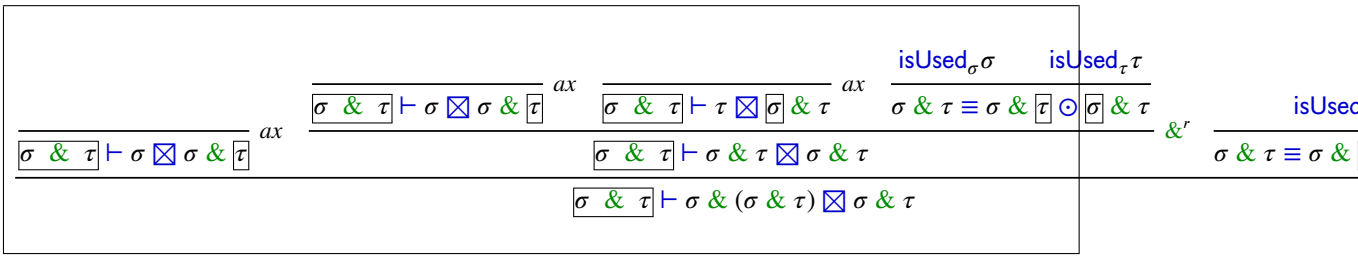


Figure 4 A derivation with a synchronisation combining a left cover of a with together with a full one.

if two covers' root constructors are equal and their subcovers are synchronised then it is only fair to say that both of them are synchronised.

It is also symmetric in its two last arguments which means that for any Δ, Δ_1, and Δ_2, if $\Delta \equiv \Delta_1 \odot \Delta_2$ holds then so does $\Delta \equiv \Delta_2 \odot \Delta_1$.

Synchronisation is not quite equality: subderivations may very-well use different parts of a with-headed assumption without it being problematic. Indeed: if both of these parts are *entirely* consumed then it simply means that we will have to introduce a different left rule at some point in each one of the subderivations. This is the only point in the process where we may introduce the cover $\sigma \,\&\, \tau$. It can take place in different situations:

The two subderivations may be *fully* using completely different parts of the assumption[4]:

$$\frac{\mathsf{isUsed}_\sigma S \qquad \mathsf{isUsed}_\tau T}{\sigma \,\&\, \tau \equiv S \,\&[\ \tau\ ] \odot [\ \sigma\ ]\& \, T}$$

But it may also be the case that only one of them is using only one side of the & whilst the other one is a full cover (see Figure 4 for an example of such a case):

$$\frac{\mathsf{isUsed}_\sigma S}{\sigma \,\&\, \tau \equiv S \,\&[\ \tau\ ] \odot \sigma \,\&\, \tau} \qquad \frac{\mathsf{isUsed}_\tau T}{\sigma \,\&\, \tau \equiv [\ \sigma\ ]\& \, T \odot \sigma \,\&\, \tau}$$

4.3 Resource-Aware Primitives

Now that Usages are properly defined, we can give a precise type to our three place relations evoked before:

$$\frac{\Gamma : \mathsf{Usages}\ \gamma \qquad k : \mathbb{N} \qquad \Delta : \mathsf{Usages}\ \gamma}{\Gamma \ni k \boxtimes \Delta : \mathsf{Set}} \qquad \frac{\Gamma : \mathsf{Usages}\ \gamma \qquad \sigma : \mathsf{ty} \qquad \Delta : \mathsf{Usages}\ \gamma}{\Gamma \vdash \sigma \boxtimes \Delta : \mathsf{Set}}$$

The definition of the calculus has already been given before and will not be changed. However we can at once define what it means for a resource to be consumed in an axiom rule. _∋_⊠_ for Usages is basically a proof-carrying de Bruijn index [8]. The proof is stored in the zro constructor and simply leverages the definition of an analogous _∋_⊠_ for Usage.

$$\frac{pr : S \ni k \boxtimes S'}{\mathsf{zro}\ pr : \Gamma \bullet S \ni k \boxtimes \Gamma \bullet S'} \qquad \frac{pr : \Gamma \ni k \boxtimes \Delta}{\mathsf{suc}\ pr : \Gamma \bullet S \ni k \boxtimes \Delta \bullet S}$$

[4] The definition of the predicate isUsed is basically mimicking the one of Cover except that the tensor constructors leaving one side untouched are disallowed.

The definition of _∋_⊠_ for Usage is based on two inductive types respectively describing what it means for a resource to be consumed out of a mint assumption or out of an existing cover.

4.3.1 Consumption from a Mint Assumption

We write $[\ \sigma\ ] \ni k \boxtimes S$ to mean that by starting with a completely mint assumption of type σ, we consume k and end up with the cover S describing the leftovers.

In the case of an atomic formula there is only one solution: to use it and end up with a total cover:

$$\frac{}{[\ \kappa\ k\ ] \ni k \boxtimes \kappa\ k}$$

In the case of with and tensor, one can decide to dig either in the left or the right hand side of the assumption to find the right resource. This gives rise to four similarly built rules; we will only give one example: going left on a tensor:

$$\frac{[\ \sigma\ ] \ni k \boxtimes S}{[\ \sigma \otimes \tau\ ] \ni k \boxtimes S \otimes [\ \tau\ ]}$$

4.3.2 Consumption from an Existing Cover

When we have an existing cover, the situation is slightly more complicated. First, we can dig into an already partially used sub-assumption using what we could call structural rules. All of these are pretty similar so we will only present the one harvesting the content on the left of a with type constructor:

$$\frac{\mathrm{S} \ni k \boxtimes S'}{\mathrm{S}\ \&[\ \tau\ ] \ni k \boxtimes S'\ \&[\ \tau\ ]}$$

Second, we could invoke the rules defined in the previous paragraphs to extract a resource from a sub-assumption that had been spared so far. This can only affect tensor-headed assumption as covers for with-headed ones imply that we have already picked a side and may not use anything from the other one. Here is a such rule:

$$\frac{[\ \tau\ ] \ni k \boxtimes T}{\mathrm{S} \otimes [\ \tau\ ] \ni k \boxtimes S \otimes T}$$

We now have a fully formal definition of the more general system we hinted at when observing the execution of the search procedure in subsection 3.1. We call this alternative formulation of the fragment of ILL we have decided to study **ILLWL** which stands for **I**ntuitionistic **L**inear **L**ogic **W**ith **L**eftovers. It will only be useful if it is equivalent to ILL. The following two sections are dedicated to proving that the formulation is both sound (all the derivations in the generalised calculus give rise to corresponding ones in ILL) and complete (if a statement can be proven in ILL then a corresponding one is derivable in the generalised calculus).

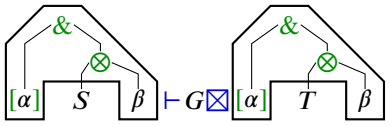
5 Completeness

The purpose of this section is to prove the completeness of our generalised calculus: to every derivation in ILL we can associate a corresponding one in the consumption-based calculus.

One of the major differences between the two calculi is that in the one with leftovers, the context decorated with consumption annotations is the same throughout the whole derivation whereas we constantly chop up the multiset of resources in ILL. To go from ILL to ILLWL, we need to introduce a notion of weakening which give us the ability to talk about working in a larger context.

5.1 A Notion of Weakening for ILLWL

One of the particularities of Linear Logic is precisely that there is no notion of weakening allowing to discard resources without using them. In the calculus with leftovers however, it is perfectly sensible to talk about resources which are not impacted by the proof process: they are merely passed around and returned untouched at the end of the computation. Given, for instance, a derivation $S \vdash G \boxtimes \mathrm{T}$ [5] in our calculus with leftovers, it makes sense to apply the same extension of the context to both the input and output context:



These considerations lead us to examine the notion of Usage(s) extensions describing systematically how one may enrich a context and to prove their innocuousness when it comes to derivability.

5.1.1 Usage extensions

We call h-Usage extension of type σ (written $\langle\ h\ \rangle\mathsf{Usage}\ \sigma$) the description of a structure containing exactly one hole denoted $\langle\ \ \rangle$ into which, using $\_\gg\mathsf{U}\_$, one may plug an Usage h in order to get an Usage σ. We give side by side the constructors for the inductive type $\langle\ \_\ \rangle\mathsf{Usage}\ \_$ and the corresponding case for $\_\gg\mathsf{U}\_$. The most basic constructor says that we may have nothing but a hole:

$$\frac{}{\langle\ \rangle : \langle\ h\ \rangle\mathsf{Usage}\ h} \qquad H \gg\mathsf{U} \langle\ \rangle = H$$

Alternatively, one may either have a hole on the left or right hand side of a tensor product (where $\_\otimes\mathsf{U}\_$ is the intuitive lifting of tensor to Usage unpacking both sides and outputting the appropriate annotation):

$$\frac{L : \langle\ h\ \rangle\mathsf{Usage}\ \sigma \qquad R : \mathsf{Usage}\ \tau}{\langle\ L\ \rangle \otimes R : \langle\ h\ \rangle\mathsf{Usage}\ \sigma \otimes \tau} \qquad H \gg\mathsf{U} \langle\ L\ \rangle \otimes R = (H \gg\mathsf{U} L) \otimes\mathsf{U} R$$

$$\frac{L : \mathsf{Usage}\ \sigma \qquad R : \langle\ h\ \rangle\mathsf{Usage}\ \tau}{L \otimes\langle\ R\ \rangle : \langle\ h\ \rangle\mathsf{Usage}\ \sigma \otimes \tau} \qquad H \gg\mathsf{U} L \otimes\langle\ R\ \rangle = L \otimes\mathsf{U} (H \gg\mathsf{U} R)$$

Or one may have a hole on either side of a with constructor as long as the other side is kept mint ($\_\&\mathsf{U}[\_]$ and $[\_]\&\mathsf{U}\_$ are, once more, operators lifting the Cover constructors to Usage):

$$\frac{L : \langle\ h\ \rangle\mathsf{Usage}\ \sigma}{\langle\ L\ \rangle\ \&[\ \tau\ ] : \langle\ h\ \rangle\mathsf{Usage}\ \sigma\ \&\ \tau} \qquad H \gg\mathsf{U} \langle\ L\ \rangle\ \&[\ \tau\ ] = (H \gg\mathsf{U} L)\ \&\mathsf{U}[\ \tau\ ]$$

$$\frac{R : \langle\ h\ \rangle\mathsf{Usage}\ \tau}{[\ \sigma\ ]\&\langle\ R\ \rangle : \langle\ h\ \rangle\mathsf{Usage}\ \sigma \otimes \tau} \qquad H \gg\mathsf{U} [\ \sigma\ ]\&\langle\ R\ \rangle = [\ \sigma\ ]\&\mathsf{U} (H \gg\mathsf{U} R)$$

[5] We write S for $\varepsilon \bullet] S [$ in order to lighten the presentation

5.1.2 Usages extensions

Usages extensions are akin to Altenkirch et al.'s Order Preserving Embeddings [1] except that they allow the modification of the individual elements which are embedded in the larger context using a Usage extension. We list below the three OPE constructors together with the corresponding cases of _≫Us_ describing how to transport a Usages along an extension. One can embed the empty context into any other context:

$$\frac{\Delta : \mathsf{Usages}\ \delta}{\varepsilon\ \Delta : \langle\ \varepsilon\ \rangle\mathsf{Usages}\ \delta} \qquad \varepsilon \gg\mathsf{Us}\ \varepsilon\ \Delta = \Delta$$

Or one may extend the head Usage using the tools defined in the previous subsection:

$$\frac{hs : \langle\ \gamma\ \rangle\mathsf{Usages}\ \delta \qquad h : \langle\ \sigma\ \rangle\mathsf{Usage}\ \tau}{hs \bullet h : \langle\ \gamma \bullet \sigma\ \rangle\mathsf{Usages}\ \delta \bullet \tau} \qquad \Gamma \bullet S \gg\mathsf{Us}\ hs \bullet h = (\Gamma \gg\mathsf{Us}\ hs) \bullet (S \gg\mathsf{U}\ h)$$

Finally, one may simply throw in an entirely new Usage:

$$\frac{hs : \langle\ \gamma\ \rangle\mathsf{Usages}\ \delta \qquad S : \mathsf{Usage}\ \sigma}{hs \bullet' S : \langle\ \gamma\ \rangle\mathsf{Usages}\ \delta \bullet \sigma} \qquad \Gamma \gg\mathsf{Us}\ hs \bullet' S = (\Gamma \gg\mathsf{Us}\ hs) \bullet S$$

Now that this machinery is defined, we can easily state and prove the following simple weakening lemmma:

▶ **Lemma 1** (Weakening for ILLWL). *Given Γ and Δ two* Usages *γ and a goal σ such that $\Gamma \vdash \sigma \boxtimes \Delta$ holds true, for any hs of type $\langle\ \gamma\ \rangle$*Usages *δ, it holds that: $\Gamma \gg\mathsf{Us}\ hs \vdash \sigma \boxtimes \Delta \gg\mathsf{Us}\ hs$.*

Proof. The proof is by induction on the derivation $\Gamma \vdash \sigma \boxtimes \Delta$ and relies on intermediate lemmas corresponding to the definition of weakening for _∋_⊠_ and _≡_⊙_. ◀

5.2 Proof of completeness

The first thing to do is to prove that the generalised axiom rule given in ILL is admissible in ILLWL.

▶ **Lemma 2** (Admissibility of the Axiom Rule). *Given a type σ, one can find S, a full* Usage *σ, such that* injs *$(\varepsilon \bullet \sigma) \vdash \sigma \boxtimes \varepsilon \bullet S$.*

Proof. By induction on σ, using weakening to be able to combine the induction hypotheses. ◀

The admissibility of the axiom rule allows us to prove completeness by a structural induction on the derivation:

▶ **Theorem 3** (Completeness). *Given a context γ and a type σ such that $\gamma \vdash \sigma$, we can prove that there exists Γ a* full Usages *γ such that* inj *$\gamma \vdash \sigma \boxtimes \Gamma$.*

Proof. The proof is by induction on the derivation $\gamma \vdash \sigma$.

Axiom The previous lemma is precisely dealing with this case.

With Introduction is combining the induction hypotheses by using the fact that two full Usages are always synchronisable and their synchronisation is a full Usages.

Tensor Introduction relies on the fact that the (proof relevant) way in which the two premises' contexts are merged gives us enough information to generate the appropriate Usages extensions along

which to weaken the induction hypotheses. The two weakened derivations are then proven to be compatible (the weakened output context of the first one is equal to the weakened input of the second one) and combined using a tensor introduction rule whose output context is indeed fully used.

Left rules The left rules are dealt with by defining ad-hoc functions mimicking the action of splitting a variable in the context (for tensor) or picking a side (for with) at the Usages level and proving that these actions do not affect derivability in ILLWL negatively. ◀

This is overall a reasonably simple proof but it had to be expected: ILL is a more explicit system listing precisely when every single left rule is applied whereas ILLWL is more on the elliptic side. Let us now deal with soudness:

6 Soundness

The soundness result tells us that from a derivation in the more general calculus, one can create a valid derivation in ILL. To be able to formulate such a statement, we need a way of listing the assumptions which have been used in a proof $\Gamma \vdash \sigma \boxtimes \Delta$; informally, we should be able to describe a Usages E such that $\lceil\, \mathrm{E} \,\rfloor \vdash \sigma$. To that effect, we introduce the notion of difference between two usages.

6.1 Usages Difference

A Usages difference E between Γ and Δ (two elements of type Usages γ) is a Usages γ such that $\Delta \equiv \Gamma - E$ holds where the three place relation $\_\equiv\_-\_$ is defined as the pointwise lifting of a relation on Usages described in Figure 5. This inductive datatype, itself based on a definition of cover differences, distinguishes three cases: if the input and the output are equal then the difference is a mint assumption, if the input was a mint assumption then the difference is precisely the output Usage and, finally, we may also be simply lifting the notion of Cover difference when both the input and the output are dented.

$$\frac{}{S \equiv S - [\ \sigma\ ]} \qquad \frac{}{S \equiv [\ \sigma\ ] - S} \qquad \frac{S \equiv S_1 - S_2}{]\ S\ [\equiv]\ S_1\ [-]\ S_2\ [}$$

Figure 5 Usage differences

Cover differences ($\_\equiv\_-\_$) are defined by an inductive type described (minus the expected structural laws which we let the reader infer) in Figure 6.

$$\frac{S \equiv S_1 - S_2}{S \otimes T \equiv S_1 \otimes T - S_2 \otimes[\ \tau\ ]} \qquad \frac{S \equiv S_1 - S_2}{S \otimes T \equiv S_1 \otimes[\ \tau\ ] - S_2 \otimes T}$$

$$\frac{T \equiv T_1 - T_2}{S \otimes T \equiv S \otimes T_1 - [\ \sigma\ ] \otimes T_2} \qquad \frac{T \equiv T_1 - T_2}{S \otimes T \equiv [\ \sigma\ ] \otimes T_1 - S \otimes T_2}$$

$$\frac{}{S \otimes T \equiv [\ \sigma\ ] \otimes T - S \otimes[\ \tau\ ]} \qquad \frac{}{S \otimes T \equiv S \otimes[\ \tau\ ] - [\ \sigma\ ] \otimes T}$$

Figure 6 Cover differences

6.2 Soundness Proof

The proof of soundness is split into auxiliary lemmas which are used to combine the induction hypothesis. These lemmas, where the bulk of the work is done, are maybe the places where the precise role played by the constraints enforced in the generalised calculus come to light. We state them here and skip the relatively tedious proofs. The interested reader can find them in the `Search/Calculus.agda` file.

▶ **Lemma 4** (Introduction of with). *Assuming that we are given two subproofs $\Delta_1 \equiv \Gamma - E_1$ and $\lceil\ E_1\ \rfloor \vdash \sigma$ on one hand and $\Delta_2 \equiv \Gamma - E_2$ and $\lceil\ E_2\ \rfloor \vdash \tau$ on the other, and that we know that the two Usages γ respectively called Δ_1 and Δ_2 are such that $\Delta \equiv \Delta_1 \odot \Delta_2$ then we can generate E, an Usages γ, such that $\Delta \equiv \Gamma - E$, $\lceil\ E\ \rfloor \vdash \sigma$, and $\lceil\ E\ \rfloor \vdash \tau$.*

Proof. The proof is by induction over the structure of the derivation stating that Δ_1 and Δ_2 are synchronised. ◀

We can prove a similar theorem corresponding to the introduction of a tensor constructor. We write $E \equiv E_1 \bowtie E_2$ to mean that the context E is obtained by interleaving E_1 and E_2. This notion is defined inductively and, naturally, is proof-relevant. It corresponds in our list-based formalisation of ILL to the multiset union mentioned in the tensor introduction rule in Figure 1.

▶ **Lemma 5** (Introduction of tensor). *Given F_1 and F_2 two Usages γ such that: $\Delta \equiv \Gamma - F_1$ and $\lceil\ F_1\ \rfloor \vdash \sigma$ on one hand and $E \equiv \Delta - F_2$ and $\lceil\ F_2\ \rfloor \vdash \tau$ on the other, then we can generate F an Usages γ together with two contexts E_1 and E_2 such that: $E \equiv \Gamma - F$, $\lceil\ F\ \rfloor \equiv E_1 \bowtie E_2$, $E_1 \vdash \sigma$ and $E_2 \vdash \tau$*

▶ **Theorem 6** (Soundness of the Generalisation). *For all context γ, all Γ, Δ of type Usages γ and all goal σ such that $\Gamma \vdash \sigma \boxtimes \Delta$ holds, there exists an E such that $\Delta \equiv \Gamma - E$ and $\lceil\ E\ \rfloor \vdash \sigma$.*

Proof. The proof is by induction on the derivation; using auxiliary lemmas to combine the induction hypothesis. ◀

▶ **Corollary 7** (Soundness of the Proof Search). *If the proof search shows that inj $\gamma \vdash \sigma \boxtimes \Delta$ holds for some Δ and Δ is a full usage then $\gamma \vdash \sigma$.*

The soundness result relating the new calculus to the original one makes explicit the fact that valid ILL derivations correspond to the ones in the generalised calculus which have no leftovers. Together with the completeness result it implies that if we can write a decision procedure for ILLWL then we will automatically have one for ILL.

7 Proof Search

We have defined a lot of elegant datatypes so far but the original goal was to implement a proof search algorithm for the fragment of ILL we have decided to study. The good news is that all the systems we have described have algorithmic rules: read bottom-up, they are a set of constructor-directed recipes to search for a proof. Depending on the set of rules however, they may or may not be deterministic and they clearly are not total because not all sequents are provable. This simply means that we will be working in various monads. The axiom rule forces us to introduce non-determinism (which we will model using the list monad); there are indeed as many ways of proving an atomic proposition as there are assumptions of that type in the context. The rule for tensor looks like two stateful operations being run sequentially: one starts by discharging the first subgoal, waits for it to *return* a modified context and then threads these leftovers to tackle the second one. And, last but not least, the rule for

with looks very much like a map-reduce diagram: we start by generating two subcomputations which can be run in parallel and later on merge their results by checking whether the output contexts can be said to be synchronised (and this partiality will be dealt with using the maybe monad).

Now, the presence of these effects is a major reason why it is important to have the elegant intermediate structures we can generate inhabitants of. Even if we are only interested in the satisfiability of a given problem, having material artefacts at our disposal allows us to state and prove properties of these functions easily rather than having to suffer from boolean blindness: "A Boolean is a bit uninformative" [16]. And we know that we may be able to optimise them away [21, 10] in the case where we are indeed only interested in the satisfiability of the problem and they turn out to be useless.

7.1 Consuming an Atomic Proposition

The proof search procedures are rather simple to implement (they straightforwardly follow the specifications we have spelled out earlier) and their definitions are succinct. Let us study them.

▶ **Lemma 8.** *Given a type σ and an atomic proposition k, we can manufacture a list of pairs consisting of a* Cover *σ we will call S and a proof that $[\ \sigma\ ] \ni k \boxtimes S$.*

Proof. We write _∈?[_] for the function describing the different ways in which one can consume an atomic proposition from a mint assumption. This function, working in the list monad, is defined by structural induction on its second (explicit) argument: the mint assumption's type.

Atomic Case If the mint assumption is just an atomic proposition then it may be used if and only if it is the same proposition. Luckily this is decidable; in the case where propositions are indeed equal, we return the corresponding consumption whilst we simply output the empty list otherwise.

Tensor & With Case Both the tensor and with case amount to picking a side. Both are equally valid so we just concatenate the lists of potential proofs after having mapped the appropriate lemmas inserting the constructors recording the choices made over the results obtained by induction hypothesis. ◀

The case where the assumption is not mint is just marginally more complicated as there are more cases to consider:

▶ **Lemma 9.** *Given a cover S and an atomic proposition k, we can list the ways in which one may extract and consume k.*

Proof. We write _∈?]_[for the function describing the different ways in which one can consume an assumption from an already existing cover. This function, working in the list monad, is defined by structural induction on its second (explicit) argument: the cover.

Atomic Case The atomic proposition has already been used, there is therefore no potential proof:

Tensor Cases The tensor cases all amount to collecting all the ways in which one may use the sub-assumptions. Whenever a sub-assumption is already partially used (in other words: a Cover) we use the induction hypothesis delivered by the function _∈?]_[itself; if it is mint then we can fall back to using the previous lemma. In each case, we then map lemmas applying the appropriate rules recording the choices made.

With Cases Covers for with are a bit special: either they are stating that an assumption has been fully used (meaning that there is no way we can extract the atomic proposition k out of it) or a side has already been picked and we can only explore one sub-assumption. As for the other cases, we need to map auxiliary lemmas. ◀

Now that we know how to list the ways in which one can extract and consume an atomic proposition from a mint assumption or an already existing cover, it is trivial to define the corresponding process for an Usage.

▶ **Corollary 10.** *Given an S of type Usage σ and an atomic proposition k, one can produce a list of pairs consisting of a Usage σ we will call T and a proof that $S \ni k \boxtimes T$.*

Proof. It amounts to calling the appropriate function to do the job and apply a lemma to transport the result. ◀

This leads us to the theorem describing how to implement proof search for the _∋_⊠_ relation used in the axiom rule.

▶ **Theorem 11.** *Given a Γ of type Usages γ and an atomic proposition k, one can produce a list of pairs consisting of a Usages γ we will call Δ and a proof that $\Gamma \ni k \boxtimes \Delta$.*

Proof. We simply call the function _∈?_ described in the previous corollary to each one of the assumptions in the context and collect all of the possible solutions: ◀

7.2 Producing Derivations

Assuming the following lemma stating that we can test for being synchronisable, we have all the pieces necessary to write a proof search procedure listing all the ways in which a context may entail a goal.

▶ **Lemma 12.** *Given Δ_1 and Δ_2 two Usages γ, it is possible to test whether they are synchronisable and, if so, return a Usages γ which we will call Δ together with a proof that $\Delta \equiv \Delta_1 \odot \Delta_2$. We call _⊙?_ this function.*

▶ **Theorem 13** (Proof Search)**.** *Given an S of type Usage σ and a type τ, it is possible to produce a list of pairs consisting of a Usage σ we will call T and a proof that $S \vdash \tau \boxtimes T$.*

Proof. We write _⊢?_ for this function. It is defined by structural induction on its second (explicit) argument: the goal's type. We work, the whole time, in the list monad.

Atomic Case Trying to prove an atomic proposition amounts to lifting the various possibilities provided to us by _∈?_ thanks to the axiom rule ax.

Tensor Case After collecting the leftovers for each potential proof of the first subgoal, we try to produce a proof of the second one. If both of these phases were successful, we can then combine them with the appropriate tree constructor ⊗.

With Case Here we produce two independent sets of potential proofs and then check which subset of their cartesian product gives rise to valid proofs. To do so, we call _⊙?_ on the returned Usages to make sure that they are synchronisable and, based on the result, either combine them using whenSome or fail by returning the empty list. ◀

7.3 From Proof Search to a Decision Procedure

The only thing missing in order for us to have a decision procedure is a proof that all possible *interesting* cases are considered by the proof search algorithm. The "interesting" keyword is here very important. In the _∋_⊠_ case, it is indeeed crucial that we try all potential candidates as future steps may reject subproofs.

▶ **Lemma 14** (No Overlooked Assumption)**.** *Given Γ, Δ two Usages γ and k an atom such that there is a proof pr that $\Gamma \ni k \boxtimes \Delta$ holds, $k \in? \Gamma$ contains the pair (Δ , pr).*

In the _≡_⊙_ case, however, it is not as important: the formalisation is made shorter by having a constructor for symmetry rather than twice as many introduction rules. This does not mean that we are interested in the proofs where one spends time applying symmetry over and over again. As a

consequence, we have to acknowledge the fact that the proof discovered by the search procedure may be different from any given proof of the same type. And this constraint is propagated all the way up to the main theorem

▶ **Theorem 15** (No Overlooked Derivation). *Given Γ, Δ two* Usages *γ and σ a type, if $\Gamma \vdash \sigma \boxtimes \Delta$ holds then there exists a derivation pr of $\Gamma \vdash \sigma \boxtimes \Delta$ such that the pair $(\Delta\ ,\ pr)$ belongs to the list $\Gamma \vdash? \sigma$.*

From this result, we can conclude that we have in practice defined a decision procedure for ILLWL and therefore ILL as per the soundness and completeness results proven in section 6 and section 5 respectively.

8 Applications: building Tactics

A first, experimental, version of the procedure described in the previous sections was purposefully limited to handling atomic propositions and tensor product. One could argue that this fragment is akin to Hutton's razor [12]: small enough to allow for quick experiments whilst covering enough ground to articulate founding principles. Now, the theory of ILL with just atomic propositions and tensor products is exactly the one of bag equivalence: a goal will be provable if and only if the multiset of its atomic propositions is precisely the context's one.

Naturally, one may want to write a solver for Bag Equivalence based on the one for ILL. But it is actually possible to solve an even more general problem: equations on a commutative monoid. Agda's standard library comes with a solver for equations on a semiring but it's not always the case that one has such a rich structure to take advantage of.

8.1 Equations on a Commutative Monoid

This whole section is parametrised by Mon a commutative monoid (as defined in the file `Algebra.agda` of Agda's standard library) whose carrier Carrier Mon is assumed to be such that equality of its elements is decidable ($\_\stackrel{?}{=}\_$ will be the name of the corresponding function). Alternatively, we may write $M.name$ to mean the *name* defined by the commutative monoid Mon (e.g. $M.Carrier$ will refer to the previously mentionned set Carrier Mon).

We start by defining a grammar for expressions with a finite number of variable whose carrier is a commutative monoid: a term may either be a variable (of which there are a finite amount n), an element of the carrier set or the combination of two terms.

```
data Expr (n : ℕ) : Set where
  `v    : (k : Fin n)          → Expr n
  `c    : (el : Carrier Mon)   → Expr n
  _`•_  : (t u : Expr n)       → Expr n
```

Assuming the existence of a valuation assigning a value of the carrier set to each one of the variables, a simple semantics can be given to these expressions:

```
Valuation : ℕ → Set
Valuation n = Vec M.Carrier n

⟦_⟧ᴱ : {n : ℕ} (t : Expr n) (ρ : Valuation n) → M.Carrier
⟦ `v k  ⟧ᴱ ρ = lookup k ρ
⟦ `c el ⟧ᴱ ρ = el
```

```
⟦ t ⁖• u ⟧ᴱ ρ = ⟦ t ⟧ᴱ ρ M.• ⟦ u ⟧ᴱ ρ
```

Now, we can normalise these terms down to vastly simpler structures: every Expr n is equivalent to a pair of an element of the carrier set (in which we have accumulated the constant values stored in the tree) together with the list of variables present in the term. We start by defining this Model together with its semantics:

```
Model : (n : ℕ) → Set
Model n = M.Carrier × List (Fin n)

⟦_⟧ᴹˢ : {n : ℕ} (ks : List (Fin n)) (ρ : Valuation n) → M.Carrier
⟦ ks ⟧ᴹˢ ρ = foldr M._•_ M.ε (map (flip lookup ρ) ks)

⟦_⟧ᴹ : {n : ℕ} (t : Model n) (ρ : Valuation n) → M.Carrier
⟦ el , ks ⟧ᴹ ρ = el M.• ⟦ ks ⟧ᴹˢ ρ
```

We then provide a normalisation function turning a Expr n into such a pair. The variable and constant cases are trivial whilst the _⁖•_ is handled by an auxiliary definition combining the induction hypotheses:

```
_••_ : {n : ℕ} → Model n → Model n → Model n
(e , ks) •• (f , ls) = e M.• f , ks ++ ls

norm : {n : ℕ} (t : Expr n) → Model n
norm (⁖v k)   = M.ε   , k :: []
norm (⁖c el)  = el    , []
norm (t ⁖• u) = norm t •• norm u
```

This normalization step is proved semantics preserving with respect to the commutative's monoid notion of equality by the following lemma:

▶ **Lemma 16** (Normalisation Soundness). *Given t an* Expr n*, for any ρ a* Valuation n*, we have:* ⟦ t ⟧ᴱ ρ M.≈ ⟦ norm t ⟧ᴹ ρ.

This means that if we know how to check whether two elements of the model are equal then we know how to do the same for two expressions: we simply normalise both of them, test the normal forms for equality and transport the result back thanks to the soundness result. But equality for elements of the model is not complex to test: they are equal if their first components are and their second ones are the same multisets. This is where our solver for ILL steps in: if we limit the context to atoms only and the goal to being one big tensor of atomic formulas then we prove precisely multiset equality. Let us start by defining this subset of ILL we are interested in. We introduce two predicates on types isAtoms saying that contexts are made out of atomic formulas and isProduct restricting goal types to big products of atomic propositions:

$$\frac{}{\kappa\ k : \mathsf{isProduct}\ \kappa\ k} \qquad \frac{S : \mathsf{isProduct}\ \sigma \qquad T : \mathsf{isProduct}\ \tau}{S \otimes T : \mathsf{isProduct}\ \sigma \otimes \tau}$$

For each one of these predicates, we define the corresponding erasure function (fromAtoms and fromProduct respectively) listing the hypotheses mentioned in the derivation. We can then formulate the following soundness theorem:

▶ **Lemma 17.** *Given three contexts γ, δ and e composed only of atoms (we call Γ, Δ and E the respective proofs that* isAtoms *holds for them) and a proof that γ is obtained by merging δ and e together, we can demonstrate that for all ρ a* Valuation *n:*

⟦ fromAtoms Γ ⟧$^{\mathsf{Ms}}$ ρ M.≈ ⟦ fromAtoms Δ ⟧$^{\mathsf{Ms}}$ ρ M.• ⟦ fromAtoms E ⟧$^{\mathsf{Ms}}$ ρ

Proof. The proof is by induction on the structure of the proof that γ is obtained by merging δ and e together. ◀

This auxiliary lemma is what allows us to prove the main soundness theorem which will allow to derive a solver for commutative monoids from the one we already have:

▶ **Theorem 18.** *From a context γ and a goal σ such that Γ and S are respectively proofs that* isAtoms *γ and* isProduct *σ hold true, and from a given proof that $\gamma \vdash \sigma$ we can derive that for any (ρ : Valuation n),* ⟦ fromAtoms Γ ⟧$^{\mathsf{Ms}}$ ρ M.≈ ⟦ fromProduct S ⟧$^{\mathsf{Ms}}$ ρ.

Proof. The proof is by induction on the derivation of type $\gamma \vdash \sigma$. The hypothesis that assumptions are atomic discards all cases where a left rule might have been applied whilst the one saying that the goal is a big product helps us discard the with introduction case.

The two cases left are therefore the variable one (trivial) and the tensor introduction one which is dealt with by combining the induction hypotheses generated by the subderivations using the previous lemma. ◀

The existence of injection function taken a list of atomic proposition as an input, delivering an appropriately atomic context or product goal is the last piece of boilerplate we need. Fortunately, it is very easy to deliver:

▶ Proposition 19 (Injection functions). From a list of atomic propositions xs, one can produce a context injs xs such that there is a proof Γ of isAtoms (injs xs) and fromAtoms Γ is equal to xs.

Similarly, from a non-empty list $x :: xs$, one can produce a type inj x xs such that there is a proof S of isProduct (inj x xs) and fromProduct S is equal to $x :: xs$.

Proof. In the first case, we simply map the atomic constructor over the list of propositions. In the second one, we create a big right-nested tensor product. ◀

We can now combine all of these elements to prove:

▶ **Corollary 20.** *Given t and u two* Expr *n and ρ a* Valuation *n, one can leverage the ILL solver to (perhaps) produce a derivation proving that* ⟦ t ⟧$^{\mathsf{E}}$ ρ M.≈ ⟦ u ⟧$^{\mathsf{E}}$ ρ

Proof. We know from Theorem 16 that we can reduce that problem to the equivalent ⟦ norm t ⟧$^{\mathsf{E}}$ ρ M.≈ ⟦ norm u ⟧$^{\mathsf{E}}$ ρ so we start by normalizing both sides to $(e\ ,\ ks)$ on one hand and $(f\ ,\ ls)$ on the other. These two normal forms are then equal if the two constants e and f are themselves equal (which, by assumption, we know how to decide) and the two lists of variables ks and ls are equal up to permutation which is the case if we are able to produce an ILL derivation injs $ks \vdash$ inj ls as per the combination of the soundness result and the injection functions' properties. ◀

Now, the standard library already contains a proof that (ℕ, 0, _+_) is a commutative monoid so we can use this fact (named ℕ+ here) to have a look at an example. In the following code snippet, LHS, RHS and CTX are respectively reified versions of the left and right hand sides of the equation, as well as the Valuation 2 mapping variables in the Expr language to their names in Agda.

```
2+x+y+1 : (x y : ℕ) → 2 + (x + y + 1) ≡ y + 3 + x
2+x+y+1 x y = proveMonEq LHS RHS CTX
```

```
  where open ℕ+
        'x   = 'v (# 0)
        'y   = 'v (# 1)
        LHS  = 'c 2 '• (('x '• 'y) '• 'c 1)
        RHS  = ('y '• 'c 3) '• 'x
        CTX  = x :: y :: []
```

The normalization step reduced proving this equation to proving that the pair $(3, \{\!|x, y|\!\})$ is equal to the pair $(3, \{\!|y, x|\!\})$. Equality of the first components is trivial whilst the multiset equality one is proven true by our solver.

8.2 Proving Bag Equivalence

We claimed that proving equations for a commutative monoid was more general than mere bag equivalence. It is now time to make such a statement formal: using Danielsson's rather consequent library for reasoning about Bag Equivalence [7], we can build a tactics for proving bag equivalences of expressions involving finite lists (and list variables) by simply leveraging the solver defined in the previous subsection. Assuming that we have a base Set named *Pr* equipped with a decidable equality $\_\stackrel{?}{=}\_$, here is how to proceed:

▶ **Lemma 21.** List *Pr* *equipped with the binary operation* _++_ *is a commutative monoid for the equivalence relation* _≈−bag_.

We therefore have a solver for this theory. Now, it would be a grave mistake to translate constants using the 'c constructor of the solver: results would be accumulated using concatenation and compared for *syntactic equality* rather than up to permutation. This means that, for instance, 1 :: 2 :: *xs* and 2 :: 1 :: *xs* would be declared distinct because their normal forms would be, respectively, the pair 1 :: 2 :: [], *xs* :: [] on one hand and 2 :: 1 :: [], *xs* :: [] on the other one. Quite embarrassing indeed.

Instead we ought to treat the expressions as massive joins of lists of singletons (seen as variables) and list variables. And this works perfectly well as demonstrated by the following example:

```
example : (xs ys : List ℕ) →
  1 :: 2 :: xs ++ 1 :: ys ≈−bag  ys ++ 2 :: xs ++ 1 :: 1 :: []
example xs ys = proveMonEq LHS RHS CTX
  where open BE
    '1   = 'v (# 0)
    '2   = 'v (# 1)
    'xs  = 'v (# 2)
    'ys  = 'v (# 3)
    LHS  = (('1 '• '2) '• 'xs) '• '1 '• 'ys
    RHS  = 'ys '• ('2 '• 'xs) '• '1 '• '1
    CTX  = sgl 1 :: sgl 2 :: xs :: ys :: []
```

Once more, LHS, RHS and CTX are the respective reifications of the left and right hand sides of the equation as well as the one of the context. All these reification are done by hand. Having a nice interface for these solvers would involve a little bit of engineering work such as writing a (partial) function turning elements of the Term type describing quoted Agda term into the corresponding Expr. All of these issues have been thoroughly dealt with by Van Der Walt and Swierstra [19, 20].

9 Conclusion, Related and Future Work

We have seen how, starting from provability in Intuitionistic Linear Logic, a problem with an extensional formulation, we can move towards a type-theoric approach to solving it. This was done firstly by generalising the problem to a calculus with leftovers better matching the proof search process and secondly by introducing resource-aware contexts which are datatypes retaining the important hidden *structure* of the problem. These constructions led to the definition of Intuitionistic Linear Logic With Leftovers, a more general calculus enjoying a notion of weakening but, at the same time, sound and complete with respect to ILL. Provability of formulas in ILL being decidable is then a simple corollary of it being decidable for ILLWL. Finally, a side effect of this formalization effort is the definition of helpful tactics targetting commutative monoids and, in particular, bag equivalence of lists.

This development has evident connections with Andreoli's vastly influential work on focusing in Linear Logic [2] which demonstrates that by using a more structured calculus (the focused one), the logician can improve her proof search procedure by making sure that she ignores irrelevant variations between proof trees. The fact that our approach is based on never applying a left rule explicitly and letting the soundness result insert them in an optimal fashion is in the same vein: we are, effectively, limiting the search space to proof trees with a very specific shape without losing any expressivity.

In the domain of certified proof search, Kokke and Swierstra have designed a prolog-style procedure in Agda [13] which, using a fuel-based model, will explore a bounded part of the set of trees describing the potential proofs generated by backward-chaining using a fixed set of deduction rules as methods.

As already heavily hinted at by the previous section, there is a number of realms which benefit from proof search in Linear Logic. Bag equivalence [7] is clearly one of them but recent works also draw connections between Intuitionistic Linear Logic and narrative representation, proof search then becomes narrative generation [4, 14, 3] and a proof is seen as a trace corresponding to one possible storyline given the plot-devices available in the context. Our approach is certified to produce all possible derivations (modulo commuting the application of the left rules) and therefore all the corresponding storylines.

9.1 Tackling a Larger Fragment

The fragment we are studying is non-trivial: as showcased, having only tensor and atomic formulas would already be equivalent to testing bag equivalence between the context and the goal; limiting ourselves to with and atomic formulas would amount to checking that there is a non-empty intersection between the context and the goal. However mixing tensors and withs creates a more intricate theory hence this whole development. It would nonetheless be exciting to tackle a larger fragment in a similar, well-structured manner.

A very important connector in ILL is the lollipop. Although dealing with it on the right hand side is extremely simple (one just extends the context with the newly acquired assumption and check that it has been entirely consumed in the subderivation corresponding to the body of the lambda abstraction), its elimination rule is more complex: if $\sigma \multimap \tau$ belongs to the context, then one needs to be able to make this specific assumption temporarily unavailable when proving its premise. Indeed, it would otherwise be possible to use its own body to discharge the premise thus leading to a strange fixpoint making e.g. $\sigma \multimap (\sigma \otimes \sigma) \vdash \sigma$ provable. We have explored various options but a well-structured solution has yet to be found.

9.2 Search Parallelisation

The reader familiar with linear logic will not have been surprised by the fact that some rules are well-suited for a parallel exploration of the provability of its sub-constituents. The algorithm we have presented however remains sequential when it comes to a goal whose head symbol is a tensor. But that is not a fatality: it is possible to design a tensor introduction rule following the map-reduce approach seen earlier. It will let us try to produce both subproofs in parallel before performing an *a posteriori* check to make sure that the output contexts of the two subcomputations are disjoint.

$$\frac{\Gamma \vdash \sigma \boxtimes \Delta_1 \qquad \Gamma \vdash \tau \boxtimes \Delta_2 \qquad \Delta \equiv \Delta_1 \uplus \Delta_2}{\Gamma \vdash \sigma \otimes \tau \boxtimes \Delta}$$

This approach would allow for a complete parallelisation of the work at the cost of more subproofs being thrown away at the merge stage because they do not fit together.

9.3 Connection to Typechecking

A problem orthogonal to proof search but that could benefit from the techniques and datastructures presented here is the one of typechecking. In the coeffect calculus introduced by Petricek, Orchard and Mycroft [17], extra information is attached to the variables present in the context. Their approach allows for writing derivations in Bounded Linear Logic or building a program with an attached dataflow analysis. However their deduction rules, when read bottom-up, are suffering from some of the issues we highlighted in this paper's introduction (having to guess how to partition a context for instance). This may be tractable for Hindley-Milner-like type systems enjoying type inference but we are interested in more powerful type theories.

We believe that moving from their presentation to one with input and output contexts as well as keeping more structured contexts would give rise to a range of calculi whose judgements are algorithmic in nature thus making them more amenable to (bidirectional) typechecking. Our notion of variable annotation also allows for slightly more subtle invariants being tracked: the annotation's structure may depend on the structure of the variable's type.

Special Thanks

This paper was typeset thanks to Stevan Andjelkovic's work to make compilation from literate agda to LaTeX possible.

Ben Kavanagh was instrumental in pushing us to introduce a visual representation of consumption annotations thus making the lump of nested predicate definitions more accessible to the first time reader.

References

1 Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Categorical reconstruction of a reduction free normalization proof. In *Category Theory and Computer Science*, pages 182–199. Springer, 1995.

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

3 Anne-Gwenn Bosser, Marc Cavazza, Ronan Champagnat, et al. Linear logic for non-linear storytelling. In *ECAI*, pages 713–718, 2010.

4 Anne-Gwenn Bosser, Pierre Courtieu, Julien Forest, and Marc Cavazza. Structural analysis of narratives with the coq proof assistant. In *Interactive Theorem Proving*, pages 55–70. Springer, 2011.

5 Samuel Boutin. Using reflection to build efficient and certified decision procedures. In *Theoretical Aspects of Computer Software*, pages 515–529. Springer, 1997.

6 Pierre Crégut. Une procédure de décision réflexive pour un fragment de l'arithmétique de presburger. In *Informal proceedings of the 15th journées francophones des langages applicatifs*, 2004.

7 Nils Anders Danielsson. Bag equivalence via a proof-relevant membership relation. In *Interactive Theorem Proving*, pages 149–165. Springer, 2012.

8 Nicolaas Govert De Bruijn. Lambda Calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. In *Indagationes Mathematicae (Proceedings)*, volume 75, pages 381–392. Elsevier, 1972.

9 Peter Dybjer. Inductive families. *Formal aspects of computing*, 6(4):440–465, 1994.

10 Andrew Gill, John Launchbury, and Simon L Peyton Jones. A short cut to deforestation. In *Proceedings of the conference on Functional programming languages and computer architecture*, pages 223–232. ACM, 1993.

11 Jean-Yves Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

12 Graham Hutton. Fold and Unfold for Program Semantics. In *Proceedings of the 3rd ACM SIGPLAN International Conference on Functional Programming*, Baltimore, Maryland, September 1998.

13 Pepijn Kokke and Wouter Swierstra. Auto in Agda: programming proof search. Submitted to ICFP 2014., 2014.

14 Chris Martens, Anne-Gwenn Bosser, Joao F Ferreira, and Marc Cavazza. Linear logic programming for narrative generation. In *Logic Programming and Nonmonotonic Reasoning*, pages 427–432. Springer, 2013.

15 Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1984.

16 Conor McBride. Epigram: Practical programming with dependent types. In *Advanced Functional Programming*, pages 130–170. Springer, 2005.

17 Tomas Petricek, Dominic Orchard, and Alan Mycroft. Coeffects: A calculus of context-dependent computation. In *ICFP 2014*, 2014.

18 Robert Pollack. On extensibility of proof checkers. In *Types for Proofs and Programs*, pages 140–161. Springer, 1995.

19 Paul van der Walt. Reflection in Agda. *Master's thesis, Universiteit Utrecht*, 2012.

20 Paul Van Der Walt and Wouter Swierstra. Engineering proof by reflection in Agda. In *Implementation and Application of Functional Languages*, pages 157–173. Springer, 2013.

21 Philip Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical computer science*, 73(2):231–248, 1990.