Quati: From Linear Logic Specifications to Inference Rules (Extended Abstract)

Vivek Nigam & Leonardo Lima
Universidade Federal da Paraíba, Brazil
`vivek.nigam@gmail.com` & `leonardo.alfs@gmail.com`

Giselle Reis
Technische Universität Wien, Vienna, Austria
`giselle@logic.at`

Abstract

In our previous work, we have shown that a great number of sequent calculus proof systems, such as a multiconclusion system for intuitionisitic logic, can be specified as theories in subexponential linear logic, a refinement of linear logic. These theories are natural, allowing, for example, the development of techniques for checking automatically non-trivial properties of the encoded systems, such as whether the encoded proof system admits cut-elimination or which rule permutations are always allowed. However, it takes some effort to check whether a linear logic theory corresponds indeed to a given proof system. This paper fills this gap by developing a technique that automatically transforms a linear logic specification into the sequent calculus inference rules it specifies thus allowing the user to check for errors in the specification. We applied this technique to our previous work for checking and enumerating rule permutations from linear logic specifications, obtaining figures as they would normally appear in a proof theory text book. We are currently implementing our technique in a tool called Quati to be launched soon.

1 Introduction

Quati is a tool to be launched soon that takes a specification in subexponential linear logic [**?**], henceforth referred to as *SELL*, and infers automatically rule permutations that are always allowed. The theory and technique that are the basis of this tool is explained in our previous work [**?**]. This paper describes one of Quati's features that was not explained before, namely, how to construct sequent calculus rules as they would appear in a standard Proof-Theory book [**?**] from a specification in *SELL*.

Subexponential linear logic is a refinement of linear logic which allows any number of exponential-like connectives (i.e., ! and ?), called subexponentials, written $!^\ell$ and $?^\ell$ with a label ℓ. These labels are organized in a pre-order, $\prec$, which specifies the provability relation among subexponentials. Moreover, some subexponentials allow weakening and contraction of the formulas to which they are applied, these are classified as unbounded. The remaining subexponentials do not allow weakening and contraction of their formulas, these are classified as bounded. In our recent work [**?**], we demonstrated that a number of proof systems with rather complicated structural rules can be specified as *SELL* theories, e.g., the multiconclusion system **mLJ** for intuitionisitic logic and a proof system for the modal logic S4. In fact our encoding is quite strong, having an adequacy on the level of derivations [**?**]. This means that there is a one-to-one correspondence between the (focused) derivations in *SELL* obtained from the theory and the derivations in the encoded proof system.

This notion of adequacy is formalized by using a more advanced proof theory concept called focusing, which we refrain to explain into details here for readability purposes. Further details can be found in [**?**]. In a nutshell focusing is a complete discipline for proof search, introduced first in the context of linear logic programming by Andreoli [**?**], which reduces the proof search non-determinism. Focused proofs can be seen as a normal-form for proof search.

For our running example, we will use the implication right rule of the multi-conclusion sequent calculus system for intuitionistic logic called **mLJ** [**?**]. We have shown in [**?**] that the linear logic formula F to the left corresponds exactly to the rule depicted on the right. In this setting, we say that intuitionistic logic is the object-level logic, and *SELL* is the meta-level logic.

$$F = \exists A.\exists B.[\lceil A \supset B\rceil^{\perp} \otimes !^{l}(?^{l}\lfloor A\rfloor \parr ?^{r}\lceil B\rceil)] \qquad\qquad \dfrac{\Gamma, A \longrightarrow B}{\Gamma \longrightarrow A \supset B, \Delta}\ [\supset_R]$$

This is illustrated by the following focused derivation that introduces the formula F:

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathcal{L} \vdots_{\infty} \lfloor\Gamma\rfloor \vdots_{l} \lceil\Delta, A \supset B\rceil \vdots_{r} \cdot \Downarrow \lceil A \supset B\rceil^{\perp}}\ [I] \quad \dfrac{\dfrac{\vdash \mathcal{L} \vdots_{\infty} \lfloor\Gamma, A\rfloor \vdots_{l} \lceil B\rceil \vdots_{r} \cdot \Uparrow}{\vdash \mathcal{L} \vdots_{\infty} \lfloor\Gamma\rfloor \vdots_{l} \cdot \vdots_{r} \cdot \Uparrow ?^{l}\lfloor A\rfloor \parr ?^{r}\lceil B\rceil)}\ [\parr, ?^{r}, ?^{l}]}{\vdash \mathcal{L} \vdots_{\infty} \lfloor\Gamma\rfloor \vdots_{l} \lceil\Delta, A \supset B\rceil \vdots_{r} \cdot \Downarrow !^{l}(?^{l}\lfloor A\rfloor \parr ?^{r}\lceil B\rceil)}\ [!^{l}]}{\vdash \mathcal{L} \vdots_{\infty} \lfloor\Gamma\rfloor \vdots_{l} \lceil\Delta, A \supset B\rceil \vdots_{r} \cdot \Downarrow \lceil A \supset B\rceil^{\perp} \otimes !^{l}(?^{l}\lfloor A\rfloor \parr ?^{r}\lceil B\rceil)}\ [\otimes]}{\vdash \mathcal{L} \vdots_{\infty} \lfloor\Gamma\rfloor \vdots_{l} \lceil\Delta, A \supset B\rceil \vdots_{r} \cdot \Uparrow}\ [D_{\infty}, 2 \times \exists]$$

Here we need to explain a bit of our notation. We use two linear logic atomic predicates $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ that take object-level formulas as arguments. The former predicate specifies a formula on the left-hand side of the object logic's sequent, while the latter a formula on the right-hand side. Thus, the collection of atomic formulas $\lfloor A_1\rfloor, \ldots \lfloor A_n\rfloor, \lceil B_1\rceil, \ldots, \lceil B_m\rceil$ in a linear logic sequent specifies the object-level sequent $A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$. If $\Gamma = A_1, \ldots, A_n$ is a multiset of formulas, then we write $\lfloor\Gamma\rfloor$ for $\lfloor A_1\rfloor, \ldots, \lfloor A_n\rfloor$, similarly for $\lceil\Gamma\rceil$.

The symbols $\Uparrow$ and $\Downarrow$ are part of the sequents of the focused system for *SELL*, enforcing the focusing discipline. For the derivation above, we use three subexponential labels ∞, l and r. This is captured in the syntax by the sequent with four contexts, one for each label and a context for formulas not marked with a subexponential: $\vdash \mathcal{L} \vdots_{\infty} \lfloor A_1\rfloor, \ldots \lfloor A_n\rfloor \vdots_{l} \lceil B_1\rceil, \ldots, \lceil B_m\rceil \vdots_{r} \cdot$ which stands for the linear logic sequent $?^{\infty}\mathcal{L}, ?^{l}\lfloor A_1\rfloor, \ldots, ?^{l}\lfloor A_n\rfloor, ?^{r}\lceil B_1\rceil, \ldots, ?^{r}\lceil B_m\rceil$, where $\mathcal{L}$ is the theory encoding the object-level logic.

The interesting fact is that the derivation above is the only way of introducing the formula F by using the focusing discipline. Notice that this derivation corresponds indeed to **mLJ**'s implication right rule: The derivation's conclusion is $\vdash \mathcal{L} \vdots_{\infty} \lfloor\Gamma\rfloor \vdots_{l} \lceil\Delta, A \supset B\rceil \vdots_{r} \cdot \Uparrow$ specifying the conclusion of **mLJ**'s implication right rule $\Gamma \longrightarrow \Delta, A \supset B$ and the derivation has one open premise $\vdash \mathcal{L} \vdots_{\infty} \lfloor\Gamma, A\rfloor \vdots_{l} \lceil B\rceil \vdots_{r} \cdot \Uparrow$, which corresponds to the premise of **mLJ**'s implication right rule $\Gamma, A \longrightarrow B$.

Checking whether one given linear logic formula G corresponds to some inference rule, in the same way as the formula F corresponds to **mLJ**'s implication right rule, requires some effort. In particular, one needs to construct the focused derivation introducing the given formula G and check whether this derivation indeed corresponds to the desired inference rule. This is particularly challenging as a given formula may specify more than one rule (see [**?**]).

As we show in our recent work [**?**], constructing such focused derivations is much easier by using propositional theories, called Answer-Sets [**?**]. These theories were used for checking which rule permutations are valid. However, one downside of this method is that it yields derivations which are hardly understandable for someone that is not familiar with linear logic, such as the focused derivation shown above. This is a serious limitation as it requires a deep understanding of focusing in order to understand the proof figures printed.

This paper solves this problem by showing how to automatically derive and draw the object-level inference rules specified by a linear logic formula as they would appear in a proof theory textbook. We are currently implementing the techniques mentioned in this paper in a tool called Quati to be released soon. We show this procedure in a rather informal fashion, using the example above as the running example. We are currently writing down the correctness proof of our specification.

Table 1: List of atomic formulas used together with their denotations and their logical axiomatization $\mathcal{T}$. Following usual logic programming conventions, all non-predicate term symbols are assumed to be universally quantified, and we use commas, ",", for conjunctions and "←" for the reverse implication.

Alphabet	Denotation	Logic Specification
$in(F, \Gamma)$	$F \in \Gamma$	No theory.
$unitctx(F, \Gamma)$	$\Gamma = \{F\}$	(r1) $in(F, \Gamma) \leftarrow unitctx(F, \Gamma)$. (r2) $\bot \leftarrow in(F_1, \Gamma), unitctx(F, \Gamma), F_1 \neq F$.
$emp(\Gamma)$	$\Gamma = \emptyset$	(r3) $\bot \leftarrow in(F, \Gamma), emp(\Gamma)$.
$union(\Gamma^1, \Gamma^2, \Gamma)$	$\Gamma = \Gamma^1 \cup \Gamma^2$	(r4) $in(F, \Gamma) \leftarrow in(F, \Gamma^1), union(\Gamma^1, \Gamma^2, \Gamma)$. (r5) $in(F, \Gamma) \leftarrow in(F, \Gamma^2), union(\Gamma^1, \Gamma^2, \Gamma)$. (r6) $emp(\Gamma) \leftarrow emp(\Gamma^1), emp(\Gamma^2), union(\Gamma^1, \Gamma^2, \Gamma)$. (r7) $in(F, \Gamma^1) \leftarrow$ *not* $in(F, \Gamma^2), in(F, \Gamma), union(\Gamma^1, \Gamma^2, \Gamma)$. (r8) $in(F, \Gamma^2) \leftarrow$ *not* $in(F, \Gamma^1), in(F, \Gamma), union(\Gamma^1, \Gamma^2, \Gamma)$.

2 Construction of Derivation Skeletons

Derivation Skeletons As explained in our previous work [?], we showed how to specify derivations in a declarative fashion by using a pair $\langle \Xi, \mathcal{B} \rangle$ called *derivation skeletons*, where Ξ is a generic derivation and $\mathcal{B}$ is a set of constraints. Its formal definition was introduced in [?] and we only informally describe them here.

The set of constraints that we use are depicted in Table 1. The meaning of these constraints is specified by the rules (r1), (r2), . . . , (r8), expressed as logic program clauses, also depicted in Table 1. These rules and the predicates in Table 1 specify in a declarative fashion the content of a context variable, Γ, in a derivation. The encoding is all based on atomic formulas of the form $in(F, \Gamma)$, which specify that the formula F is in the context Γ.

The atomic formula $unitctx(F, \Gamma)$ specifies that the context Γ has a single formula F. The first rule (r1) specifies that $in(F, \Gamma)$, while the second rule (r2) is a constraint rule specifying that there is no other formula F' different from F in the context Γ.

In some situations, for instance, when specifying the linear logic initial rule [?], we need to specify that some contexts are empty, which is done by using the atomic formula $emp(\Gamma)$. Rule (r3) is a constraint that specifies that no formula can be in an empty context.

The most elaborate specification are the rules (r4) – (r8), which specify the atomic formula $union(\Gamma^1, \Gamma^2, \Gamma)$, i.e. $\Gamma = \Gamma^1 \cup \Gamma^2$. The rules (r4) and (r5) specify that $\Gamma^1 \subseteq \Gamma$ and $\Gamma^2 \subseteq \Gamma$, that is, a formula occurrence that is in Γ^i is also in Γ. The rule (r6) specifies that if both Γ^1 and Γ^2 are empty then so is Γ. The rules (r7) and (r8) specify that these contexts are bounded, that is, the union $\Gamma = \Gamma^1 \cup \Gamma^2$ is a multiset union. An occurrence of a formula in Γ either comes from Γ^1 or from Γ^2.

Consider the following illustrative example of how a derivation skeleton specifies declaratively an inference rule:

Example: Consider the $\otimes_R$ rule shown to the left. Its *inference skeleton* is the pair $\langle \Xi_\otimes, \mathcal{B}_\otimes \rangle$, where $\Xi_\otimes$ is the derivation shown to the right:

$$\frac{\Gamma \vdash \Delta, A \qquad \Gamma' \vdash \Delta', B}{\Gamma, \Gamma' \vdash \Delta, \Delta', A \otimes B}\ [\otimes_R] \qquad \frac{\Gamma_1^1 \vdash \Gamma_2^1 \quad \Gamma_1^2 \vdash \Gamma_2^2}{\Gamma_1^0 \vdash \Gamma_2^0}$$

Notice that the generic derivation is composed by context variables Γ_j^i. In particular, each sequent and each side of the sequent is marked with a context variable, that is, its contents are not specified, thus generic. It is only used for specifying the shape of the derivation and not the formulas that appear on it.

The configuration of formulas in the derivation are specified by set of constraints. The set of constraints associated to the generic derivation above is:

$$\mathcal{B} = \left\{ \begin{array}{c} unitctx(A \otimes B, \Gamma^1_{aux}), unitctx(A, \Gamma^2_{aux}), unitctx(B, \Gamma^3_{aux}), \\ union(\Gamma^1_{aux}, \Gamma^4_{aux}, \Gamma^0_2), union(\Gamma^2_{aux}, \Gamma^5_{aux}, \Gamma^1_2), union(\Gamma^3_{aux}, \Gamma^6_{aux}, \Gamma^2_2), \\ union(\Gamma^5_{aux}, \Gamma^6_{aux}, \Gamma^4_{aux}), union(\Gamma^1_1, \Gamma^1_2, \Gamma^0_1) \end{array} \right\}$$

Here the context variables of the form Γ^i_{aux}, where $i \in \mathbb{N}$, are auxiliary context variables not appearing in the generic derivation above, but used to specify them. It is easy to check that the Logic Program (LP) $\mathcal{B}_\otimes \cup \mathcal{T}$ has a single model (called answer-set by the logic programming community), containing the formulas $in(A \otimes B, \Gamma^0_2)$, $in(A, \Gamma^1_2)$ and $in(B, \Gamma^2_2)$. There is a number of efficient solvers available. For Quati we use DLV [**?**].

In our previous work [**?**], we discuss the advantages of representing a derivation using logic specifications. In particular, we can reason over these specifications, thus reasoning about different logics in a uniform way. For instance, we developed the machinery necessary for checking whether a rule permutes over another rule in the object logic using its *SELL* specification. Therefore, we can check the permutation of rules in many systems using the same method. In the following section, we enter into the details of extracting derivation skeletons from a linear logic specification. This is new with respect to [**?**] and complements this work with our previous work [**?**] on encoding proof systems in *SELL*.

2.1 Extracting a Derivation Skeleton from a Linear Logic Formula

In the Introduction, we briefly described that linear logic can be used as a meta-logic to specify object-logic inference rules with a strong level of adequacy [**?**]. Here we will show how we can obtain the corresponding derivation skeleton from a linear logic formula.

We show some illustrative cases of our procedure, which is defined recursively on the size of formulas. Assume that there are n subexponentials. We initialize the algorithm by first constructing a sequent:

$$\vdash \Gamma^0_1, \ldots, \Gamma^0_n$$

Each context variable corresponds to one of the subexponential contexts. For example, for the encoding of **mLJ** shown in the introduction, there are three contexts: l, r, ∞, resulting in the following initial sequent:

$$\vdash \Gamma^0_\infty, \Gamma^0_l, \Gamma^0_r$$

We initialize this sequent as Seq_k, where $k = 0$. Now, given a linear logic formula F and the sequent Seq_k, our procedure runs recursively on the height of the focused derivation introducing the linear logic formula F and returns a set of derivation skeletons all with the same generic derivation, $\langle \Xi, \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$. Each $\mathcal{B}_i$ specifies a possible derivation.

Case $\otimes$**:** if F is of the form $A \otimes B$, we construct the derivation skeleton:

$$\frac{\vdash \Gamma^{k+1}_1, \ldots, \Gamma^{k+1}_n \qquad \vdash \Gamma^{k+2}_1, \ldots, \Gamma^{k+2}_n}{\vdash \Gamma^k_1, \ldots, \Gamma^k_n}$$

where Seq_k is the sequent $\vdash \Gamma^k_1, \ldots, \Gamma^k_n$ and the context variables Γ^{k+1}_j and Γ^{k+2}_j are fresh for all j that correspond to a bounded subexponential j and $\Gamma^{k+1}_i = \Gamma^k_i = \Gamma^{k+2}_i$ for all unbounded subexponentials i.

Let $\langle \Xi_{k+1}, \mathcal{B}^1_{k+1}, \ldots, \mathcal{B}^m_{k+1} \rangle$ and $\langle \Xi_{k+2}, \mathcal{B}^1_{k+2}, \ldots, \mathcal{B}^l_{k+2} \rangle$ be the sets of derivation skeletons obtained by this algorithm when using the first and second premises with respectively A and B. Then the result of the algorithm on F and Seq_k is the derivation skeleton with generic derivation

$$\frac{\Xi_{k+1} \qquad \Xi_{k+2}}{\vdash \Gamma^k_1, \ldots, \Gamma^k_n}$$

and set of set of constraints $\{\mathcal{B}^i_{k+1} \cup \mathcal{B}^j_{k+2} \cup \mathcal{B}_\otimes \mid 1 \leq i \leq m, 1 \leq j \leq l\}$, where $\mathcal{B}_\otimes$ is

$$\mathcal{B}_\otimes = \{union(\Gamma^{k+1}_j, \Gamma^{k+2}_j, \Gamma^k_j) \mid \ j \text{ bounded}\}$$

The set $\mathcal{B}_\otimes$ simply specifies that the contents of Γ^k_j are split among the premises if j is a bounded subexponential.

Case $!^\ell$: When $F = !^\ell A$, the derivation skeleton has a single premise:

$$\frac{\vdash \Gamma^{k+1}_1, \ldots, \Gamma^{k+1}_n}{\vdash \Gamma^k_1, \ldots, \Gamma^k_n}$$

where Seq_k is the sequent $\vdash \Gamma^k_1, \ldots, \Gamma^k_n$ and $\Gamma^{k+1}_i = \Gamma^k_i$ for all unbounded subexponentials.

Let $\langle \Xi_{k+1}, \mathcal{B}^1_{k+1}, \ldots, \mathcal{B}^m_{k+1}\rangle$ be the derivation skeletons obtained by this algorithm when using the premise and A. The result of the algorithm on F and Seq_k is the derivation skeleton

$$\frac{\Xi_{k+1}}{\vdash \Gamma^k_1, \ldots, \Gamma^k_n}$$

and the set of constraints $\{\mathcal{B}^i_{k+1} \cup \mathcal{B}_{!^\ell} \mid 1 \leq i \leq m\}$, where $\mathcal{B}_{!^\ell}$ is the set

$$\mathcal{B}_{!^\ell} = \{union(\Gamma^{k+1}_j, \Gamma^j_{aux}, \Gamma^k_j), emp(\Gamma^j_{aux}) \mid \ j \text{ bounded}\} \cup \{emp(\Gamma^{k+1}_i) \mid \ \ell \not\preceq i\}$$

where all auxiliary contexts Γ^j_{aux} are fresh. The set to the left specifies that the contents of Γ^k_i are the same as the contents of Γ^{k+1}_i for all contexts i that are bounded and the set to the right specifies that all contexts Γ^{k+1}_i for subexponentials i such that $\ell \not\preceq i$ have to be necessarily empty. The emptiness of these contexts is exactly the side-condition of the rule for $!^\ell$.

Case literal: There are two possibilities, one where $F = A^\perp$ is a literal introduced by an initial rule or $F = A$ is an atomic formula resulting in an open premise. We show the former case, as the latter case is similar to the cases below.

This is the base case of the algorithm. It returns the derivation:

$$\frac{}{Seq_k}$$

and the set of set of constraints $\{\mathcal{B}^j_B \mid j \text{ bounded}\} \cup \{\mathcal{B}^j_U \mid j \text{ unbounded}\}$, where

$$\begin{aligned}\mathcal{B}^j_B &= \{unitctx(A, \Gamma^j_k)\} \cup \{emp(\Gamma^i_k) \mid i \neq j \text{ and } i \text{ bounded}\}\\ \mathcal{B}^j_U &= \{in(A, \Gamma^j_k)\} \cup \{emp(\Gamma^i_k) \mid \ i \text{ bounded}\}\end{aligned}$$

The first type of set specifies the case when the matching atomic formula, A, appears alone in a bounded context, while the second type of set specifies the case when the matching atomic formula appears in an unbounded context, in which case it does not have to appear alone.

The following example illustrates the execution of the algorithm described above.

Example: Consider the linear logic formula shown to the left that corresponds, as described in the introduction, to the **mLJ** inference rule shown to the right.

$$F = \exists A.\exists B.[\lceil A \supset B\rceil^\perp \otimes !^l(?^l\lfloor A\rfloor \parr ?^r\lceil B\rceil)] \qquad\qquad \frac{\Gamma, A \longrightarrow B}{\Gamma \longrightarrow A \supset B, \Delta}\ [\supset_R]$$

Applying the algorithm described above, we obtain the following Derivation Skeleton, which is in fact the derivation obtained by our implementation, thus the difference of notation; for example $rght(\cdot)$ is used for $\lceil\cdot\rceil$ and $lft(\cdot)$ used for $\lfloor\cdot\rfloor$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma^8_\Gamma;\Gamma^3_r;\Gamma^3_l;\Gamma^1_{infty};\Downarrow \neg rght(imp(A)(B))} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\Gamma^9_\Gamma;\Gamma^6_r;\Gamma^5_l;\Gamma^1_{infty};\Uparrow}{\Gamma^9_\Gamma;\Gamma^4_r;\Gamma^5_l;\Gamma^1_{infty};\Uparrow ?^r rght(B)}}{\Gamma^9_\Gamma;\Gamma^4_r;\Gamma^3_l;\Gamma^1_{infty};\Uparrow ?^l lft(A) :: ?^r rght(B)}}{\Gamma^9_\Gamma;\Gamma^4_r;\Gamma^3_l;\Gamma^1_{infty};\Uparrow ?^l lft(A) \parr ?^r rght(B)}}{\Gamma^9_\Gamma;\Gamma^3_r;\Gamma^3_l;\Gamma^1_{infty};\Downarrow !^l ?^l lft(A) \parr ?^r rght(B)}}{\Gamma^7_\Gamma;\Gamma^3_r;\Gamma^3_l;\Gamma^1_{infty};\Downarrow \neg rght(imp(A)(B)) \otimes !^l ?^l lft(A) \parr ?^r rght(B)}}{\Gamma^7_\Gamma;\Gamma^3_r;\Gamma^3_l;\Gamma^1_{infty};\Downarrow \exists A \neg rght(imp(A)(B)) \otimes !^l ?^l lft(A) \parr ?^r rght(B)}}{\Gamma^7_\Gamma;\Gamma^3_r;\Gamma^3_l;\Gamma^1_{infty};\Downarrow \exists B \exists A \neg rght(imp(A)(B)) \otimes !^l ?^l lft(A) \parr ?^r rght(B)}}{\Gamma^7_\Gamma;\Gamma^3_r;\Gamma^3_l;\Gamma^1_{infty};\Uparrow}$$

There is just a single set of constraints attached to this generic derivation:

$$\left\{\begin{array}{c} union(\Gamma^8_\Gamma, \Gamma^9_\Gamma, \Gamma^7_\Gamma), in(rght(imp(A)(B)), \Gamma^3_r), emp(\Gamma^8_\Gamma), \\ emp(\Gamma^9_\Gamma), emp(\Gamma^4_r), unitctx(lft(A), \Gamma^4_l), unitctx(rght(B), \Gamma^5_r), \\ union(\Gamma^3_l, \Gamma^4_l, \Gamma^5_l), union(\Gamma^4_r, \Gamma^5_r, \Gamma^6_r). \end{array}\right\}$$

Notice that the constraints specify that the context Γ^6_r in the open premise may only contain the formula $\lceil B \rceil$ as expected and moreover the formula $\lceil A \supset B \rceil$ is in the context Γ^3_r. This set of constraints is given to the prover DLV together with the theory depicted in Table 1 and the output of the prover are the minimal models of the theory. For this particular example, there is a single model depicted below:

$$\left\{\begin{array}{c} union(\Gamma^8_\Gamma, \Gamma^9_\Gamma, \Gamma^7_\Gamma), union(\Gamma^4_r, \Gamma^5_r, \Gamma^6_r), union(\Gamma^3_l, \Gamma^4_l, \Gamma^5_l), \\ in(rght(imp(A)(B)), \Gamma^3_r), in(lft(A), \Gamma^4_l), in(lft(A), \Gamma^5_l), in(rght(B), \Gamma^5_r), in(rght(B), \Gamma^6_r), \\ unitctx(lft(A), \Gamma^4_l), unitctx(rght(B), \Gamma^5_r), emp(\Gamma^7_\Gamma), emp(\Gamma^8_\Gamma), emp(\Gamma^9_\Gamma), emp(\Gamma^4_r) \end{array}\right\}$$

It is easy to check that the algorithm described above does construct derivation skeletons that correspond indeed to the focused derivation introducing a formula. However, there is a downside to this representation as it is non-standard. One needs to understand the meaning of constraints and of focused derivations. In the following section, we demonstrate how to transform such a derivation skeleton into an inference rule closer to how it would be shown in a proof theory textbook.

3 From Derivation Skeletons to Inference Rules

This section details how we transform a derivation skeleton $\langle \Xi, \mathcal{B} \rangle$ into an inference rule. To be more precise, we transform a model M of the theory $\mathcal{B} \cup \mathcal{T}$ into an inference rule. Thus the same derivation skeleton may have several (or no) inference rules associated to it, namely one inference rule to each model of its set of constraints. Given a model M and the generic derivation Ξ, we apply the following rewrite rules in two phases:

Phase 1:	$unitctx(F, \Gamma) : \Gamma \to F$	$emp(\Gamma)$:	$\Gamma \to \cdot$	$union(\Gamma', \Gamma'', \Gamma)$:	$\Gamma \to \Gamma', \Gamma''$
Phase 2:		$in(F, \Gamma)$:	$\Gamma \to \Gamma, F$		

In the first phase, we proceed as follows: We set $\Xi_i := \Xi$ where $i := 0$. We repeat the following procedure: If $unitctx(F, \Gamma)$, $emp(\Gamma)$, $union(\Gamma', \Gamma'', \Gamma)$ are in M, such that there is an occurrence of Γ in Ξ_i, then apply the corresponding rule, obtaining Ξ_{i+1} and set $i := i + 1$. Let Ξ' be the resulting generic derivation. For the second phase we initialize $M_i := M$ and $\Xi_i := \Xi'$, where $i := 0$. We repeat the following procedure while there is a formula of the form $in(F, \Gamma)$ in M_i: Let $in(F, \Gamma) \in M_i$, then we apply the corresponding rule to all occurrences of Γ in Ξ_i obtaining Ξ_{i+1} and set $M_{i+1} = M \setminus \{in(F, \Gamma)\}$ and $i := i + 1$.

Example: For the derivation skeleton shown in the example at the end of Section 2.1 and using the model shown in that example, we obtain the following derivation (again obtained from our implementation):

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_r^0, imp(A)(B) \Downarrow \neg lft(imp(A)(B))} \quad \dfrac{\dfrac{\dfrac{\Gamma_r^0, imp(A)(B) \Uparrow}{\Gamma_r^0, imp(A)(B) \Uparrow ?^r rght(A)}}{\Gamma_r^0, imp(A)(B) \Downarrow ?^r rght(A)} \quad \dfrac{\dfrac{imp(A)(B) \Uparrow}{\Gamma_r^0, imp(A)(B) \Uparrow ?^l lft(B)}}{\Gamma_r^0, imp(A)(B) \Downarrow ?^l lft(B)}}{\Gamma_r^0, imp(A)(B) \Downarrow ?^r rght(A) \otimes ?^l lft(B)}}{\Gamma_r^0, imp(A)(B) \Downarrow \neg lft(imp(A)(B)) \otimes ?^r rght(A) \otimes ?^l lft(B)}}{\Gamma_r^0, imp(A)(B) \Downarrow \exists A \neg lft(imp(A)(B)) \otimes ?^r rght(A) \otimes ?^l lft(B)}}{\Gamma_r^0, imp(A)(B) \Downarrow \exists B \exists A \neg lft(imp(A)(B)) \otimes ?^r rght(A) \otimes ?^l lft(B)}}{\Gamma_r^0, imp(A)(B) \Uparrow}
$$

Two-sided Sequents As we are using the one-sided presentation of linear logic with subexponentials, the derivations used in our derivation skeleton is one sided. This contrasts with the usual presentation of such inference rules in textbooks that have two sided sequents. For this, however, we need more information from the user. In particular, the user should specify which type of formulas, of the form $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$ or both, and how many, a single or many formulas, a subexponential context is supposed to have. This is done in the specification by declarations like the following:

```
subexpctx l many lft.    subexpctx r many rght.
```

The declaration on the left specifies that the contexts for the subexponential l have many formulas, but all of them are of them form $\lfloor \cdot \rfloor$, which means that there are formulas to be placed to the left of the sequent, similarly for the declaration on the right.

Given such declarations, we associate to each context of a subexponential Γ_s^i the context Γ_s^i, with the same name, to represent the object logic formulas in Γ_s^i that should be placed to the left-hand-side of sequents, and Δ_s^i for the object logic formulas in Γ_s^i that should be placed to the right-hand-side. With these new contexts, the derivation above is rewritten to the derivation below where sequents are two-sided. The meta-level formulas of the form $\lceil F \rceil$ (respectively, $\lfloor F \rfloor$) are replaced by F and placed to the right-hand-side (respectively, left-hand-side) of the sequent. We also elide contexts that have no declaration of which formulas they have; for instance, the contexts for the subexponential $infty$ are elided.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_l^3 \vdash \Delta_r^3, imp(A)(B)} \quad \dfrac{\dfrac{\dfrac{\dfrac{\Gamma_l^3, A \vdash B}{\Gamma_l^3, A \vdash \cdot}}{\Gamma_l^3 \vdash \cdot}}{\Gamma_l^3 \vdash \cdot}}{\Gamma_l^3 \vdash \Delta_r^3, imp(A)(B)}}{\Gamma_l^3 \vdash \Delta_r^3, imp(A)(B)}}{\Gamma_l^3 \vdash \Delta_r^3, imp(A)(B)}}{\Gamma_l^3 \vdash \Delta_r^3, imp(A)(B)}}{\Gamma_l^3 \vdash \Delta_r^3, imp(A)(B)}
$$

Notice that, since the declaration above specifies that the subexponential r contains many formulas, but they are all placed to the right-hand-side of sequents, the derivation above only has the Δ_r^i contexts, while the contexts of the form Γ_r^i are not shown.

The last step is then to simply collapse the derivation obtained and construct the inference rule by using the conclusion of the derivation and its open premises. For the derivation above, we obtain the following inference rule:

$$\frac{\Gamma_l^3, A \vdash B}{\Gamma_l^3 \vdash \Delta_r^3, imp(A)(B)}\ imp_R$$

Notice that such a rule is very close to how it would appear in a standard proof-theory textbook.

4 Conclusions and Future Work

This paper describes informally the procedure for extracting inference rules, as they would appear in a proof theory textbook, from a specification of a proof system in subexponential linear logic. This procedure is currently being implemented as part of a tool called Quati that computes rule permutations.

There is a number of future work directions to follow from this work. The first one is to formalize our ideas, part of which can be found already in our previous work [**?**]. Besides the implementation of our method, another direction that we are currently investigating is the extraction of *SELL* specifications from a set of inference rules, i.e. the opposite direction of what was shown in this work. This step will facilitate the use of our tools, such as the one described in [**?**], called TATU [**?**], without requiring a greater understanding of subexponential linear logic to encode proof systems in it.

Finally, we are also investigating how to use Quati in other applications, such as the correctness of algorithms to maintain recursive distributed database [**?**].

References

[1] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.

[2] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In *ICLP*, pages 579–597, 1990.

[3] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[4] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7:499–562, July 2006.

[5] S. Maehara. Eine darstellung der intuitionistischen logik in der klassischen. *Nagoya Mathematical Journal*, pages 45–64, 1954.

[6] Vivek Nigam, , Elaine Pimentel, and Giselle Reis. An extended framework for specifying and reasoning about proof systems. Accepted to Journal of Logic and Computation. Available on Nigam's homepage.

[7] Vivek Nigam, Limin Jia, Boon Thau Loo, and Andre Scedrov. Maintaining distributed logic programs incrementally. *Computer Languages, Systems & Structures*, 38(2):158–180, 2012.

[8] Vivek Nigam and Dale Miller. Algorithmic specifications in linear logic with subexponentials. In *ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 129–140, 2009.

[9] Vivek Nigam and Dale Miller. A framework for proof systems. *J. Autom. Reasoning*, 45(2):157–188, 2010.

[10] Vivek Nigam, Elaine Pimentel, and Giselle Reis. The Tatu system. `http://www.logic.at/staff/giselle/tatu/`, 2013.

[11] Vivek Nigam, Giselle Reis, and Leonardo Lima. Checking proof transformations with ASP. In *ICLP (Technical Communications)*, 2013.

[12] Anne S. Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 1996.