Quati: An Automated Tool for Proving Permutation Lemmas

Vivek Nigam[1], Giselle Reis[2], and Leonardo Lima[1]

[1] Universidade Federal da Paraíba, Brazil
[2] Technische Universität Wien, Austria

Abstract. The proof of many foundational results in structural proof theory, such as the admissibility of the cut rule and the completeness of the focusing discipline, rely on permutation lemmas. It is often a tedious and error prone task to prove such lemmas as they involve many cases. This paper describes the tool Quati which is an automated tool capable of proving a wide range of inference rule permutations for a great number of proof systems. Given a proof system specification in the form of a theory in linear logic with subexponentials, Quati outputs in LaTeX the permutation transformations for which it was able to prove correctness and also the possible derivations for which it was not able to do so. As illustrated in this paper, Quati's output is very similar to proof derivation figures one would normally find in a proof theory book.

1 Introduction

Permutation lemmas play an important role in proof theory. Many foundational results about proof systems rely on the fact that some rules permute over others. For instance, permutation lemmas are used in Gentzen-style cut-elimination proofs [4], the completeness proof of focusing disciplines [1,7], and the proof of Herbrand's theorem [5].

Proving permutation lemmas, however, is often a tedious and error-prone task as there are normally many cases to consider. As an example, consider the case of permuting $\vee_l$ over $\rightarrow_l$ in the intuitionistic calculus LJ. In order to show whether these two rules permute, one needs to check *every possible case* in which $\rightarrow_l$ occurs above $\vee_l$ in a derivation. When using a multiplicative calculus, there are four possibilities for such derivation, two allow a permutation of the rules while the other two do not. Here's one of each:

$$\dfrac{\varphi_1 \quad \dfrac{\varphi_2 \quad \varphi_3}{\Gamma' \vdash A \quad \Gamma'', Q, B \vdash F}\,\rightarrow_l}{\Gamma, P \vdash F \quad \Gamma', \Gamma'', A \rightarrow B, Q \vdash F}\,\vee_l \qquad \rightsquigarrow \qquad \dfrac{\varphi_2 \quad \dfrac{\varphi_1 \quad \varphi_3}{\Gamma, P \vdash F \quad \Gamma'', B, Q \vdash F}\,\vee_l}{\Gamma' \vdash A \quad \Gamma, \Gamma'', P \vee Q, B \vdash F}\,\rightarrow_l$$

$$\dfrac{\varphi_1 \quad \dfrac{\varphi_2 \quad \varphi_3}{\Gamma', Q \vdash A \quad \Gamma'', B \vdash F}\,\rightarrow_l}{\Gamma, P \vdash F \quad \Gamma', \Gamma'', A \rightarrow B, Q \vdash F}\,\vee_l \qquad \rightsquigarrow \quad ?$$

The combinatorial nature of proving permutation lemmas can be observed in this example. While there are "only" four cases to consider for this pair of rules, for proving

S. Demri, D. Kapur, and C. Weidenbach (Eds.): IJCAR 2014, LNAI 8562, pp. 255–261, 2014.

the completeness of the focusing discipline, one needs to study which permutations are allowed and therefore all pairs of rules need to be considered [7]. Moreover, the fact that the cases are rarely documented makes it hard for others to check the correctness of the transformations. For instance, the cut-elimination result for bi-intuitionistic logic given by Rauszer [14] was later found to be incorrect [2] exactly because one of the permutation lemmas was not true. Therefore, an automated tool to check for these lemmas would be of great help. This paper introduces such a tool called Quati.[1]

While here we will restrict ourselves to simply illustrate Quati's functionalities and implementation design, we observe that its underlying theory is described in the papers [8,13,9]. We briefly review this body of work.

In [13], we show how to reduce the problem of proving permutation lemmas to solving an answer-set program [3]. That is, given a proof system $\mathcal{P}$ satisfying some properties, we reduce the problem of checking whether a rule r_1 in $\mathcal{P}$ always permutes over r_2 in $\mathcal{P}$ to solving an answer-set program. Each solution of this program corresponds to one possible permutation case. This result sets the foundations for Quati.

However, the exact language in which proof systems are specified was not dealt in [13]. It was subject of the paper [8] which shows that a great number of proof systems for different logics (*e.g.*, linear, intuitionistic, classical, modal logics) can be specified as theories in linear logic with subexponentials (SELL) [11]. These specifications are shown to have a strong adequacy, namely, *on the level of derivations* [12], meaning that there is a one to one correspondence of derivations in the specified logic (object logic) to derivations in linear logic with subexponentials. Moreover, [8] also shows how to check whether proof systems specified in SELL admit cut-elimination. This lead to the tool TATU[2]. Therefore, SELL is a suitable framework for specifying proof systems.

Finally, in the workshop paper [9], we show how to integrate the material in [13] and [8]. Given a proof system specified in SELL, we reduce the problem of checking whether a rule permutes over another to an answer-set program. In the same paper, we also discuss how to extract proof derivation figures similar to those shown in a standard proof theory book [15] from the solutions of the generated answer-set programs.

Quati is the result of this series of papers. This paper is organized as follows: Section 2 describes Quati's syntax and its features, while Section 3 describes its implementation. In Section 4 we end by pointing out future work.

2 Quati at Work

Throughout this section, we will use the specification for the intuitionistic logic's multi-conclusion calculus MLJ [6] as our running example. First we specify Quati's syntax and then its features.

2.1 Syntax

Quati's underlying logic, linear logic with subexponentials (SELL) [11], is a powerful framework for the specification of proof systems. Subexponentials, written $!^{\ell}$, $?^{\ell}$, arise

[1] Quati is a mammal from the raccoon family native to South America. Its name comes from the Tupi-guarani, a language spoken by native indians in Brazil, and means "long nose".

[2] `https://www.logic.at/staff/giselle/tatu/`

$Side ::= \texttt{lft} \mid \texttt{rght}$ $\quad CtxType ::= \texttt{many} \mid \texttt{single}$ $\quad SubType ::= \texttt{unb} \mid \texttt{lin}$

$SubSig ::= SubDecl\ \ SubSpec\ \ SubRel$

$SubDecl ::= \texttt{subexp}\langle String\rangle\langle SubType\rangle.$

$SubSpec ::= \texttt{subexpctx}\langle String\rangle\langle CtxType\rangle\langle Side\rangle.$

$SubRel ::= \texttt{subexprel}\langle String\rangle\texttt{<}\langle String\rangle.$

$Bipoles ::= (\texttt{not}\langle Atoms\rangle)\texttt{*}\langle BodyPos\rangle.$

$BodyPos ::= \texttt{one} \mid BodyNeg \mid \texttt{[}\langle String\rangle\texttt{]bang}BodyNeg \mid$
$\quad BodyPos\texttt{*}BodyPos \mid BodyPos\texttt{+}BodyPos$

$BodyNeg ::= \texttt{top} \mid \texttt{bot} \mid \langle MarkAtoms\rangle \mid \langle BodyNeg\rangle|\langle BodyNeg\rangle \mid$
$\quad \langle BodyNeg\rangle\texttt{\&}\langle BodyNeg\rangle$

$Atoms ::= \langle Side\rangle\langle Form\rangle \qquad MarkAtoms ::= \texttt{[}\langle String\rangle\texttt{]?}\langle Atoms\rangle$

Fig. 1. Here $Form$ is a term of type `form`

$\texttt{*}:\otimes$	$\texttt{+}:\oplus$	$\texttt{\&}:\&$	$\texttt{|}:\parr$	$\texttt{[i]bang}:!^i$	$\texttt{one}:1$	$\texttt{top}:\top$	$\texttt{bot}:\bot$	$\texttt{[i]?}:?^i$

Fig. 2. Syntax for the linear logic connectives

from the observation that the linear logic exponentials are not canonical (see [8] for an extensive discussion). It is known that these operators greatly increase the expressiveness of the system when compared to linear logic. For instance, subexponentials can be used to represent contexts of proof systems [8], to mark the epistemic state of agents [10], or to specify locations in sequential computations [11]. The main feature of subexponentials is that they are organized in a pre-order, $\preceq$, which specifies the provability relation among them. In [8], we have shown that a great number of proof systems for linear, classic, intuitionistic and modal logics can be specified in SELL with a strong level of adequacy. Another important reason for using SELL as specification language is that one can also use other available tools, such as the tool TATU which is capable of checking whether a proof system specified in SELL admits cut-elimination.

A Quati program is a SELL theory with some more annotations. Its syntax is given in Figure 1 and explained in detail by using our running example MLJ. A Quati program consists of two files: (1) a *type signature* file, with suffix `.sig` and (2) a specification file with suffix `.pl` consisting of two parts: (a) a *subexponential signature* and (b) the rules' specifications or *bipoles*.

Type signature This file contains type and kind declarations of the object logic's elements. The kind `form` is built-in and represent the type of formulas of the object logic. In general, only the connectives' types need to be declared in this file:

```
%%%%%%%%%%%%%%%%%%%% Signature %%%%%%%%%%%%%%%%%%%%
type imp form -> form -> form.
```

Subexponential signature The following subexponential signature is used for specifying the proof system MLJ:

```
%%%%%%%%%%%%%%%% Subexponential Signature %%%%%%%%%%%%%%%%
subexp l unb.                          subexp r unb.
```

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

Intuitively, one subexponential corresponds to one context of the object logic sequent.[3] MLJ has only two contexts, one to the left and another to the right side of the sequent, thus we use two subexponentials `l` and `r`. Moreover, as both contexts (to the left and right) behave classically in MLJ, we specify `l` and `r` to be unbounded, denoted by `unb`. In contrast, the specification of LJ would specify the subexponential `r` to be linear, as the right side of LJ's sequents behaves linearly.

The commands `subexpctx l many lft.` and `subexpctx r many rght.` are not formally needed for specifying proof systems, but as discussed in [9], they are needed in order to improve the visualization of the proof rules. In particular, the former specifies that the context corresponding to the subexponential `l` contains only formulas of the left side of the sequent, denoted by `lft`, and may contain many formulas, denoted by `many`. In contrast, as the context to the right side of LJ sequents has only one formula, the subexponential `r` for that system would be annotated with `single`.

The pre-order among the subexponentials is specified on the last line using the keyword `subexprel`.

Bipoles The second part of the `.pl` file is composed by $bipoles$. The concrete syntax for SELL connectives is depicted in Figure 2. The class of $bipole$ formulas often appear in proof theory literature due to its good focusing behaviour [1]. The following bipoles specify, respectively, the left and right implication introduction rules [8]. The capital letters are assumed to be existentially quantified.

```
%%%%%%%%%%%%%%%%%%%%%%%%%% Bipoles %%%%%%%%%%%%%%%%%%%%%%%%%%
% Implication
(not (lft (imp A B))) * (([r]? (rght A)) * ([l]? (lft B))).
(not (rght (imp A B))) * [l]bang (([l]? (lft A)) | ([r]? (rght B))).
```

The head of these bipoles, formulas `(not (lft (imp A B)))` and `(not (rght (imp A B)))`, specify that an implication formula to, respectively, the left and right-hand-side is introduced. The body specifies the premises of these rules. For instance, the first bipole specifies that its corresponding inference rules has two premises because of the branching caused by the tensor `*` appearing in the body of its rule, while the second has only one premise as no branching is required. The interesting bit is the $!^l$ (`[l]bang`) in the second bipole specifying that the context of the subexponential `r` should be weakened as `l > r`. In fact, by using advanced proof theoretic machinery, namely focusing [1], we can make this intuition precise in the sense. We refer to [8] for more details on encodings.

2.2 Features

Quati has two main features: (1) It can construct the corresponding inference rule(s) associated to a SELL formula; and (2) it can prove permutation lemmas. We illustrate these features with the specification of MLJ implication introduction rules shown above.

[3] There are some specifications where a subexponential is used to capture the structural properties of the proof system and therefore does not necessarily correspond to a context in the object logic. See [8] for more on this.

Rule Construction Proving the adequacy theorems for a given SELL specification is also error-prone. As detailed in [8], to prove (strong) adequacy we need to show that all the possible focused derivations that introduce a formula in the specification correspond to an inference rule of the proof system being specified. Quati automates the proof of such adequacy theorems by constructing from a bipole the corresponding inference rule. To do so, Quati uses the machinery described in [13,9] reducing this problem to the problem of solving answer-set programs.

For the MLJ specification given above, one can use the command `#rule` in the command line and select a SELL bipole in the loaded specification. Then Quati generates a LaTeX document containing all possible inference rules that correspond to that bipole. If we select the bipole used to specify MLJ's implication right rule, Quati outputs the LaTeX code for the following figure:

$$\frac{\dot{l}\ \Gamma_l^0, a \vdash \dot{r}\ b}{\dot{l}\ \Gamma_l^0 \vdash \dot{r}\ \Delta_r^0, imp(a)(b)}\ imp_R$$

Notice that this rule looks very similar to MLJ's implication right introduction rule shown in any proof theory textbook. The context Δ_r^0 is erased in the premise. The $\dot{l}$ and $\dot{r}$ are used to delimit the contexts for the subexponentials `l` and `r`, respectively. Quati uses the subexponential specification to infer that the context for `l` (resp. for `r`) should only be on the left-hand-side (resp. right-hand-side) of the sequent.

Under the hood, Quati is constructing the focused derivation [1] that introduces such a SELL bipole as described in [8]. This can be observed by using the command `#bipole`. For the same SELL bipole used above, Quati returns the LaTeX code for the following figure, corresponding to its focused derivation:

$$\dfrac{\dfrac{\Gamma^4_{gamma}; \Gamma^4_r; \Gamma^3_l; \Gamma^1_{infty}; \Downarrow \neg rght(imp(a)(b))}{\Gamma^5_{gamma}; \Gamma^4_r; \Gamma^3_l; \Gamma^1_{infty}; \Downarrow !^l ?^l lft(a) \parr ?^r rght(b)}\ \otimes \quad \dfrac{\dfrac{\dfrac{\dfrac{\Gamma^5_{gamma}; \Gamma^7_r; \Gamma^5_l; \Gamma^1_{infty}; \Uparrow}{\Gamma^5_{gamma}; \Gamma^5_r; \Gamma^5_l; \Gamma^1_{infty}; \Uparrow ?^r rght(b)}}{\Gamma^5_{gamma}; \Gamma^5_r; \Gamma^3_l; \Gamma^1_{infty}; \Uparrow ?^l lft(a) ::^r rght(b)}}{\Gamma^5_{gamma}; \Gamma^5_r; \Gamma^3_l; \Gamma^1_{infty}; \Uparrow ?^l lft(a) \parr ?^r rght(b)}}{\Gamma^5_{gamma}; \Gamma^4_r; \Gamma^3_l; \Gamma^1_{infty}; \Downarrow !^l ?^l lft(a) \parr ?^r rght(b)}}{\Gamma^3_{gamma}; \Gamma^4_r; \Gamma^3_l; \Gamma^1_{infty}; \Downarrow \neg rght(imp(a)(b)) \otimes !^l ?^l lft(a) \parr ?^r rght(b)}}{\Gamma^3_{gamma}; \Gamma^4_r; \Gamma^3_l; \Gamma^1_{infty}; \Uparrow}$$

Rule Permutation. As described in the Introduction, Quati can be used to prove permutation lemmas. The command `#permute` checks whether the permutation of two selected rules is always allowed or not. Quati outputs, again in LaTeX, the cases for which it was able to find the permutation and the cases for which it was not able to find a permutation. For example, when Quati checks whether MLJ's implication left introduction rule permutes over MLJ's implication right introduction, it correctly finds two possible permutation cases and it cannot find one of the cases for which is indeed not possible. We show one of the cases (reformatted to fit the page margins):

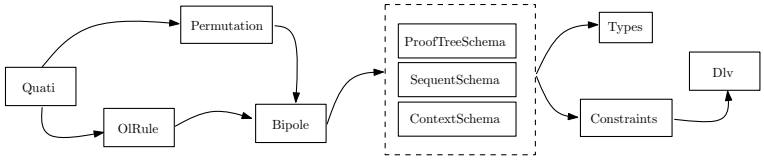
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\dot{\imath}\ \Gamma_l^0, imp(a)(b), c \vdash \dot{\imath}\dot{r}\ d}}{\dot{\imath}\ \Gamma_l^0, imp(a)(b) \vdash \dot{\imath}\dot{r}\ \Delta_r^0, imp(c)(d), a}\, imp_R \quad \dfrac{}{\dot{\imath}\ \Gamma_l^0, imp(a)(b), b \vdash \dot{\imath}\dot{r}\ \Delta_r^0, imp(c)(d)}}{\dot{\imath}\ \Gamma_l^0, imp(a)(b) \vdash \dot{\imath}\dot{r}\ \Delta_r^0, imp(c)(d)}\, imp_L}{\dot{\imath}\ \Gamma_l^0, imp(a)(b) \vdash \dot{\imath}\dot{r}\ \Delta_r^0, imp(c)(d)}$$

$$\rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\dot{\imath}\ \Gamma_l^7, imp(a)(b), c \vdash \dot{\imath}\dot{r}\ a, d} \quad \dfrac{}{\dot{\imath}\ \Gamma_l^7, imp(a)(b), c, b \vdash \dot{\imath}\dot{r}\ d}}{\dot{\imath}\ \Gamma_l^7, imp(a)(b), c \vdash \dot{\imath}\dot{r}\ d}\, imp_L}{\dot{\imath}\ \Gamma_l^7, imp(a)(b) \vdash \dot{\imath}\dot{r}\ \Delta_r^9, imp(c)(d)}\, imp_R$$

Once again, this proof figure is very similar to the proof figure that one would find in a standard proof theory textbook. Notice that it uses the fact that the contexts are unbounded, i.e. formulas can be contracted or weakened, to infer the permutation above (see [13] for more discussion on how this works).

3 Implementation Details

Quati is implemented in OCaml[4] and makes use of DLV[5] externally to compute minimal models for the answer-set programs generated. It is part of a bigger project, called `sellf`[6] which also includes the machinery for TATU mentioned above. The following diagram provides an overview of the main modules in `sellf` used by Quati for checking permutations.



The basic data structure, defined in the module `Types`, is linear logic formulas with subexponentials. The bipoles in Quati are represented by *proof tree schemas*, defined in the module `ProofTreeSchema`, which uses the modules `SequentSchema` and `ContextSchema`. As the name suggests, these are schematic representations of proof trees, sequents and contexts that use generic contexts [13] to represent possibly non-empty sets of formulas. The constraints that will later compose the answer-set program are implemented in the module `Constraints`. The application of linear logic rules with constraints is implemented in the `ProofTreeSchema` module. The computation of possible bipoles of a formula is in the module `Bipole`. The `Permutation` module makes use of the bipole generation to construct the derivations of two rules. Given the constraints of a derivation, module `Dlv` contains the code for executing DLV externally, parsing the result and returning the minimal models. The translation of a proof tree schema and constraints into an object logic derivation is done in the `OlRule` module. It contains data structures to represent proof trees, sequents and contexts of an object logic and the rewriting algorithm described in [13] (module `Derivation`).

[4] `http://ocaml.org/`
[5] `http://www.dlvsystem.com/dlv/`
[6] `https://code.google.com/p/sellf/`

Quati was tested using some proof systems including LK, LJ, MLJ, LL, S4, G1m and LAX. On most cases, each permutation lemma can be checked in less than one second. The implementation can be downloaded at

`http://www.logic.at/staff/giselle/quati`.

4 Conclusions and Future Work

This paper introduced Quati, an automated tool for proving permutation lemmas. Besides briefly commenting on its implementation, we illustrated its syntax, usage and features. Besides MLJ, in the download one can find the specification of all proof systems tested, as well as system requirements and installation instructions.

There are several directions we are currently investigating for continuing this work. One is to come up with more graphical ways of writing proof systems and how to translate such representations into SELL specifications. Another possibility is the derivation of completeness of focusing strategies in an automated fashion, since such theorems rely heavily on permutation lemmas. Finally, we are investigating ways to construct machine-readable proof objects for permutation lemmas.

References

1. Andreoli, J.-M.: Logic programming with focusing proofs in linear logic. J. of Logic and Computation 2(3), 297–347 (1992)
2. Crolard, T.: Subtractive logic. Theor. Comput. Sci. 254(1-2), 151–185 (2001)
3. Gelfond, M., Lifschitz, V.: Logic programs with classical negation. In: ICLP (1990)
4. Gentzen, G.: Investigations into logical deductions. The Collected Papers of Gerhard Gentzen (1969)
5. Herbrand, J.: Recherches sur la Théorie de la Démonstration. PhD thesis (1930)
6. Maehara, S.: Eine darstellung der intuitionistischen logik in der klassischen. Nagoya Mathematical Journal, 45–64 (1954)
7. Miller, D., Saurin, A.: From proofs to focused proofs: a modular proof of focalization in linear logic. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 405–419. Springer, Heidelberg (2007)
8. Nigam, V., Pimentel, E., Reis, G.: An extended framework for specifying and reasoning about proof systems. Accepted to Journal of Logic and Computation, `http://www.nigam.info/docs/modal-sellf.pdf`
9. Nigam, V., Reis, G., Lima, L.: Quati: From linear logic specifications to inference rules (extended abstract). In: Brazilian Logic Conference, EBL (2014), `http://www.nigam.info/docs/ebl14.pdf`
10. Nigam, V.: On the complexity of linear authorization logics. In: LICS (2012)
11. Nigam, V., Miller, D.: Algorithmic specifications in linear logic with subexponentials. In: PPDP (2009)
12. Nigam, V., Miller, D.: A framework for proof systems. J. Autom. Reasoning 45(2), 157–188 (2010)
13. Nigam, V., Reis, G., Lima, L.: Checking proof transformations with ASP. In: ICLP (Technical Communications) (2013)
14. Rauszer, C.: A formalization of the propositional calculus h-b logic. Studia Logica (1974)
15. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory (1996)