Strategies for Logic Programming Languages

Tatjana Lutovac and James Harland

Department of Computer Science
Royal Melbourne Institute of Technology
GPO Box 2476V, Melbourne, 3001, Australia
{tanja, jah} @cs.rmit.edu.au

Abstract. Logic programs consist of formulas of mathematical logic and various proof-theoretic techniques can be used to design and analyse execution models for such programs. We briefly review the main problems, which are questions that are still elusive in the design of logic programming languages, from a proof-theoretic point of view. Existing strategies which lead to the various languages are all rather sophisticated and involve complex manipulations of proofs. All are designed for analysis on paper by a human and many of them are ripe for automation. We aim to perform the automation of some aspects of strategies for logic programming language, in order to assist in the design of these languages. In this paper we describe the first steps towards the design of such an automatic analysis tool. We investigate the usage of particular proof manipulations for the analysis of logic programming strategies. We propose a more precise specification of sequent calculi inference rules that we use as a framework for the generalisation and automation-oriented specification of the permutation process.

1 Introduction

It is well known that logic programming (LP) may be thought of as the application of the techniques of mathematical logic to programming tasks. Logic programs may be considered as collections of formulas and their computation may be identified as searching for proofs: given a program $\mathcal{P}$ and a goal G we attempt to satisfy G by searching for a proof of $\mathcal{P} \rightarrow G$ using the inference rules of a given logic. In addition, logic programs may be analysed declaratively, *i.e.* as statements of logic, without reference to a sequence of operations, and so are more amenable to program transformation and optimisation than many imperative languages. Also, programs written in LP languages tend to be somewhat more concise, and hence more comprehensible to a human, than the same programs written in an imperative language (such as C or Pascal).

There have been various LP systems implemented (Prolog [15], Lolli[6], LinLog[1], λProlog[12], Forum[11], Lygon[17], etc). But still there is no precise concept of what constitutes a LP language as well as what is the essential difference between a LP system and a theorem prover. One proof-theoretic characterisation and a criterion for identifying LP language is formalised in [10] by the identification of *goal-directed proofs* which are usually defined in terms of a *sequent calculus* system of proofs. Essentially, a goal-directed proof is a (cut-free) proof in which a goal is decomposed before the program, and hence the program is not used until the goal is reduced to just an atom. The properties of such proofs have been thoroughly investigated for various fragments of first and higher-order intuitionistic logic (classes of formulas called Horn clauses [5] and hereditary Harrop formulas [10]). Whilst this analysis is quite thorough, there remains the question of how to "lift" the concept of goal-directed proofs to other logics (for example multiple-conclusioned logic, linear logic, light linear logic, affine logic, relevant linear logic, etc.). Given the evolution of programming languages towards higher and higher level languages, with a corresponding increase in the computational power of the execution models of these languages, a natural demand is a wide range of expressive logics and more powerful inference facilities in which to write programs. The relationship between the logical fragment and the proof strategy is fundamental. The proof search strategy should be *complete* (to always find a proof, if one exists) for a given fragment of logic. For example, the strategy of *uniform proofs* [10] can prove all results for a Horn clause fragment or hereditary Harrop fragment of intuitionistic logic. At the same time, in general, we need complete and feasible classes of proofs with a view to automating and optimising proof search. The choice of

logical fragment is a compromise between the expressiveness of the logic and the difficulty of proof search automation. This compromise affects the ability of the language to express simply and naturally the solution to a given problem. In some cases, we are able to select a logical fragment on the basis of a given problem to be solved. Still there is no clear technical statement of why, and what kind of restrictions on some classes of formulas are necessary, in order to use classes of formulas in logic programs.

There have been various proof-theoretic techniques, approaches and analyses [1, 2, 3, 5, 6, 9, 13] used to design and analyse LP languages. Many of the existing approaches and analyses which lead to some of the answers are all rather sophisticated and involve complex manipulations of proofs. Many are restricted to particular logic or classes of formulae. Almost all are designed for analysis on paper by a human and many of them are ripe for automation, being formally defined in precise detail, and yet somewhat overwhelming for humans. A general answer and formal definition requires a thorough understanding of all the existing LP systems, as well as of the relationship between theorem proving and LP and, in particular, how to use proof-theoretic techniques to design and analyse execution models for LP languages.

We present initial work on the problem of automating the strategies for the design of LP languages. We propose a more precise specification of sequent calculi inference rules that we use for the generalisation and automation-oriented specification of the permutation process. The full version may be found in [8].

1.1 Aims and expected outcomes

The design of new LP languages demands detailed and systematic analyses of existing LP proof systems and precise answers to the questions:

What class of formulae can be used in programs? What strategy can be applied?

What is the influence and the actual impact of selected choices [18] on the characteristics of the corresponding LP language (for example on expressibility, efficiency, supporting modular programming, abstract data types, higher-order functions, simple operational behaviour, providing proofs as *answers* as well as all possible answer substitutions, etc)?

We plan to perform a systematic study of existing approaches, solutions, strategies and to investigate ways of modifying them to make them more suitable for automated analysis and proof-search. The plan is to design a system, an *automatic analyser, i.e.* a *prototype language compiler generator* which will be able to:

- *recognise the appropriate logical fragment which may be used as a basis for a LP language;*
- *identify what combinations of inference rules are necessary for the execution of a given fragment;*
- *recognise (and perhaps define) proofs which conform to a given strategy;*
- *identify fragments for which a given strategy is complete.*

Many implementations of LP languages have many features in common, such as the use of unification and some form of the resolution rule. The analyser would be a first step towards providing a library of tools such as unification methods and algorithms, top-down and bottom-up proof search procedures and other implementation techniques. But it would be more than just an automatic support tool for the proof-theoretic analyses. It should be an *adaptive system* capable of systematically analysing a given set of inference rules in order to extract and to determine the most appropriate and optimised way to assemble the various pieces into a LP system. It is expected that such systems in the future will be able to determine the appropriately optimised execution model.

2 Proof Manipulations as a tool for Strategy Analysis

Analysis and specification of LP strategies is not just a matter of determining whether a given goal is provable on the base of a given program, but of providing more information about the proof search

process. Unlike a theorem prover, an LP system is used to find answers, rather than proofs per se *i.e.* other properties of the proof are taken into account other than just proof existence. Thus, for example, for a non-ground query $\exists x P(x)$, the LP system is expected to return not only *yes* or *no*, but also an *answer i.e.* substitution for the (existentially quantified) variables of the goal: a value of x for which $P(x)$ is true. It is well known that a proof contains much more information than the simple fact that its conclusion is true. The so-called *meta-level* properties of a proof, include, among others, identification of the strategy applied to a given proof and subgoals where more than one strategy is applicable, recognising families of proofs which contain a common structure, finding all possible proofs as well as proofs which lead to different answers, determining the number of answers for an existentially quantified (sub)goal, etc.

There have been various *proof manipulations* which enable extracting meta-level information about a proof, and which are, therefore, a necessary tool in the analysis of existing LP systems and strategies. Proof manipulations include various procedures for modification and adaptation of a proof to different search strategies and to different requirements such as elimination of redundant steps and loops in a proof, decreasing number of rules or the amount of branching in a proof, cut elimination and elimination of other rules with the high level of non-determinism, etc. In the rest of this section we point out two proof manipulations of particular interest: *permutations* and *hypothesis elimination*.

2.1 Permutations

The *permutation* of two adjacent inference rules of a given proof is reversing their order in the proof but without disturbing the rest of the proof (the part above and below the inferences modulo a duplication of some proof branches and a renaming of certain free variables) as a result of which we get the *equivalent* proof to the given one:

$$\dfrac{\dfrac{\vdots \quad \vdots}{\vdash a, c, d, \Gamma \qquad \vdash b, \Delta}\otimes}{\dfrac{\vdash c, d, a \otimes b, \Gamma, \Delta}{\vdash c \wp d, a \otimes b, \Gamma, \Delta}\wp}\vdots \quad \Longleftrightarrow \quad \dfrac{\dfrac{\vdots}{\vdash c, d, a, \Gamma}\wp \qquad \vdots}{\vdash c \wp d, a, \Gamma \qquad \vdash b, \Delta}\otimes}{\vdash c \wp d, a \otimes b, \Gamma, \Delta}\vdots$$

Permutations are essential proof manipulations. The dynamics of the proof that results from the permutation analysis clearly has a strong influence on the operational semantics of the resulting LP language. For example, Lygon[17] is based on the search strategy that *some* permutations of right-hand side rules will lead to a proof, whereas Forum[11] is based on the search strategy that *any* permutation of right-hand rules will lead to a proof. Here we give a brief overview of some applications of permutations for the analysis of particular LP strategies.

Example 1 (Defining a proof search strategy).
Permutation properties of the inference rules of a given logical fragment, have direct impact on defining efficient proof search strategies which deal with the order of inference rules. We explain this impact through the so-called *"Normal proof"* [3] strategy which is applicable in LP. It imposes constraints for the choice of formula to be decomposed in the next step of the proof construction. These constraints are based on the permutability of inference rules and contribute to reduction of some sources of non-determinism during the proof construction. In a *bottom-up proof construction* (in which search starts at the root of the tree) this approach always first applies rule for which there is a "guarantee" that in the case of a provable sequent there is a proof with that rule in that particular place in the proof tree. Also, it is important to detect groups of rules that are easy to apply from the conclusion (when are applying in a bottom up direction). It is desirable to apply such rules *as soon as possible* in a proof construction and to postpone the others (which introduce don't know non-determinism when applied in up direction) *as late as possible*. The degree of the realization of this aim *to apply a rule I as soon as possible* or *to apply a rule I as late as possible* is directly dependent on the permutability relationships between I and other inference rules in a given logical fragment *i.e.* in a given proof.

For instance, consider the sequent $\vdash A \otimes B, C \wp D, \Gamma$. During proof construction we must have a complete strategy for selecting the next formula to be decomposed. This selection is crucial for bottom-up proof construction. On the basis of the permutation results given in [3] we have the following table, where the case (t_1, t_2) in the table contains: p iff inference t_1 *permutes down* over inference t_2, and, np iff there exists a proof in which inferences t_1 and t_2 are not permutable.

Table 1.

	t_1					
t_2	$\wp$	$\forall$	$\otimes$	$\oplus$	$\exists$	$w?$
$\wp$	p	p	np	p	p	p
$\forall$	p	p	p	p	np	p
$\otimes$	p	p	p	p	p	p
$\oplus$	p	p	p	p	p	p
$\exists$	p	p	p	p	p	p
$w?$	p	p	p	p	p	p

As a consequence of np for the case $(\otimes, \wp)$ and p for the case $(\wp, \otimes)$, for any provable sequent $\vdash A \otimes B, C \wp D, \Gamma$ there is a proof where $\wp$ is applied before (closer to the root than) the $\otimes$ rule, but not necessary the proof with $\otimes$ applied before $\wp$. For instance, the proof construction of the sequent $\vdash a \otimes b, a^{\perp} \wp b^{\perp}$ terminates successfully only when $\wp$ precedes $\otimes$:

$$\dfrac{\dfrac{\dfrac{}{\vdash a, a^{\perp}}\,Ax \qquad \dfrac{}{\vdash b, b^{\perp}}\,Ax}{\vdash a \otimes b, a^{\perp}, b^{\perp}}\,\otimes}{\vdash a \otimes b, a^{\perp} \wp b^{\perp}}\,\wp \qquad \dfrac{\dfrac{\dfrac{?}{\vdash a, a^{\perp}, b^{\perp}}\,\wp \qquad \dfrac{?}{\vdash b}\,\wp}{\vdash a, a^{\perp} \wp b^{\perp}}\,\otimes}{\vdash a \otimes b, a^{\perp} \wp b^{\perp}} \qquad \dfrac{\dfrac{\dfrac{?}{\vdash a}\,\wp \qquad \dfrac{?}{\vdash b, a^{\perp}, b^{\perp}}\,\wp}{\vdash b, a^{\perp} \wp b^{\perp}}\,\otimes}{\vdash a \otimes b, a^{\perp} \wp b^{\perp}}$$

while in a proof construction of the sequent $\vdash a \otimes b, a^{\perp} \wp ? b^{\perp}, b^{\perp}$, the order of applied rules does not matter:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a, a^{\perp}}\,Ax}{\vdash a, a^{\perp}, ?b^{\perp}}\,w? \qquad \dfrac{}{\vdash b, b^{\perp}}\,Ax}{\vdash a \otimes b, a^{\perp}, ?b^{\perp}, b^{\perp}}\,\otimes}{\vdash a \otimes b, a^{\perp} \wp ? b^{\perp}, b^{\perp}}\,\wp \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a, a^{\perp}}\,Ax}{\vdash a, a^{\perp}, ?b^{\perp}}\,w? \qquad \dfrac{}{\vdash b, b^{\perp}}\,Ax}{\vdash a, a^{\perp} \wp ? b^{\perp}}\,\wp}{\vdash a \otimes b, a^{\perp} \wp ? b^{\perp}, b^{\perp}}\,\otimes$$

Thus, in the proof construction of a sequent $\vdash A \otimes B, C \wp D, \Gamma$ the choice to decompose $C \wp D$ first has a greater chance to be successful *i.e.* to be the choice that lead to a proof.

Example 2 (Completeness of some strategies).
Permutation properties can be used to either prove that, for a given set of inference rules, a given search strategy will find all possible proofs, or construct an example of a provable sequent which can not be proved with that strategy. This is specific for strategies which deal with the order of inference rules. Strategies which pass this test(s) may be amenable to further analysis, such as minimising the amount of branching in the proof, or delaying certain choices until the optimum amount of information is available.

We explain this on a simplified version of *"normal proof"* strategy whose full version is given in [3]. Let assume that, in the logical fragment we consider, the sequents are one-sided: $\vdash \Gamma$. Γ is a multiset of goal formulas, denoted G and defined by the following grammar:

$$G := A | G \otimes G | G \wp G | G \oplus G | \forall x G | \exists x G | ?G$$

Let us assume that the allowed set of inference rules for proving the specified sequents is $\{\otimes, \wp, \oplus_1, \oplus_2, \forall, \exists, w?\}$. The strategy is defined by the following algorithm:

If $\vdash \Gamma$ contains a formula of the form $?F$
  then $\vdash \Gamma$ is the conclusion of $w?$ rule
  else
    if $\vdash \Gamma$ contains a formula whose top-most connective is $\wp$ or $\forall$
    then $\vdash \Gamma$ is the conclusion of rule which introduces that particular formula
    else $\vdash \Gamma$ contains a formula whose top-most connective is $\otimes$ or $\oplus$ or $\exists$ and
      $\vdash \Gamma$ is the conclusion of rule which introduces that particular formula

The completeness properties of the outlined strategy follow directly from the permutability properties of the concerned linear logic fragment, which are given in Table 1. Any proof, from this logical fragment, can be transformed, by permutations of its inferences, to a proof that obeys the given strategy:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash p(t), p^{\perp}(t)}\,Ax \quad \dfrac{\dfrac{\dfrac{}{\vdash a, a^{\perp}}\,Ax}{\vdash a, a^{\perp}, ?b}\,w?}{\vdash a, a^{\perp} \wp ?b}\,\wp}{\vdash p(t) \otimes a, p^{\perp}(t), ?b \wp a^{\perp}}\,\otimes}{\vdash p(t) \otimes a, \exists y p^{\perp}(y), ?b \wp a^{\perp}}\,\exists}{\vdash \forall x (p(x) \otimes a), \exists y p^{\perp}(y), ?b \wp a^{\perp}}\,\forall \quad \longmapsto \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a, a^{\perp}}\,Ax \quad \dfrac{\dfrac{}{\vdash p(t), p^{\perp}(t)}\,Ax}{\vdash p(t), \exists y p^{\perp}(y)}\,\exists}{\vdash p(t) \otimes a, \exists y p^{\perp}(y), a^{\perp}}\,\otimes}{\vdash p(t) \otimes a, \exists y p^{\perp}(y), ?b, a^{\perp}}\,w?}{\vdash p(t) \otimes a, \exists y p^{\perp}(y), ?b \wp a^{\perp}}\,\wp}{\vdash \forall x (p(x) \otimes a), \exists y p^{\perp}(y), ?b \wp a^{\perp}}\,\forall$$

Example 3 (Generating equivalent proofs which obey different strategies).
A logic program for which there are many proofs of the same goal (and hence there are goals which return the same answer substitution many times) is generally considered to be somewhat deficient. Different proofs are only interesting if they lead to different answers.

As example, consider the sequent $p(a), \forall y q(y) \Rightarrow p(y), q(b) \vdash \exists x p(x)$. All possible uniform proofs[1] can be classified in two groups in regard to the answer substitution for x:

Group 1:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{q(b) \vdash q(b)}\,Ax}{q(b), p(a) \vdash q(b)}\,w-L \quad \dfrac{}{p(b) \vdash p(b)}\,Ax}{p(a), q(b) \Rightarrow p(b), q(b) \vdash p(b)}\,\Rightarrow -L}{p(a), \forall y q(y) \Rightarrow p(y), q(b) \vdash p(b)}\,\forall - L}{p(a), \forall y q(y) \Rightarrow p(y), q(b) \vdash \exists x p(x)}\,\exists - R$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{q(b) \vdash q(b)}\,Ax \quad \dfrac{}{p(b) \vdash p(b)}\,Ax}{q(b) \Rightarrow p(b), q(b) \vdash p(b)}\,\Rightarrow -L}{\forall y q(y) \Rightarrow p(y), q(b) \vdash p(b)}\,\forall - L}{p(a), \forall y q(y) \Rightarrow p(y), q(b) \vdash p(b)}\,w - L}{p(a), \forall y q(y) \Rightarrow p(y), q(b) \vdash \exists x p(x)}\,\exists - R$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{q(b) \vdash q(b)}\,Ax \quad \dfrac{}{p(b) \vdash p(b)}\,Ax}{q(b) \Rightarrow p(b), q(b) \vdash p(b)}\,\Rightarrow -L}{p(a), q(b) \Rightarrow p(b), q(b) \vdash p(b)}\,w - L}{p(a), \forall y q(y) \Rightarrow p(y), q(b) \vdash p(b)}\,\forall - L}{p(a), \forall y q(y) \Rightarrow p(y), q(b) \vdash \exists x p(x)}\,\exists - R$$

[1] Uniform proof [10] of a sequent $\mathcal{P} \vdash G$ means a goal-directed proof, in a sense that the goal G is decomposed uniformly, based only on its structure and without reference to the program $\mathcal{P}$, until atomic goals are reached. The program is only consulted when atomic goals are to be proved.

Group 2:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p(a)\vdash p(a)}\,Ax}{p(a),q(b)\vdash p(a)}\,w-L}{p(a),\forall yq(y)\Rightarrow p(y),q(b)\vdash p(a)}\,w-L}{p(a),\forall yq(y)\Rightarrow p(y),q(b)\vdash \exists xp(x)}\,\exists-R \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{p(a)\vdash p(a)}\,Ax}{p(a),\forall yq(y)\Rightarrow p(y)\vdash p(a)}\,w-L}{p(a),\forall yq(y)\Rightarrow p(y),q(b)\vdash p(a)}\,w-L}{p(a),\forall yq(y)\Rightarrow p(y),q(b)\vdash \exists xp(x)}\,\exists-R$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{q(b)\vdash q(b)}\,Ax}{p(b),q(b)\vdash q(b)}\,w-L \quad \dfrac{}{p(a)\vdash p(a)}\,Ax}{p(a),q(b)\Rightarrow p(b),q(b)\vdash p(a)}\,\Rightarrow -L}{p(a),\forall yq(y)\Rightarrow p(y),q(b)\vdash p(a)}\,\forall-L}{p(a),\forall yq(y)\Rightarrow p(y),q(b)\vdash \exists xp(x)}\,\exists-R$$

Clearly, the question of finding all proofs which lead to different answers have some overlap with the question of finding all equivalent proofs modulo inference permutations. Since proofs from the same equivalence class lead to the same answer, it is sufficient to generate just one of them, which than will be representative for the whole class. This is reduced to the problem of finding all permutations of the proof *i.e.* of finding the appropriate equivalence class, which than will be "excluded from further consideration".

Uniformity of proofs is a key strategy for the logic programming proofs. In the above example, in the uniform proofs which provide $x \leftarrow b$ substitution, the reasons for such substitution, which is performed (because of the uniformity constraint) in the first step of proof construction, are vague:

$$\dfrac{\begin{array}{c}\vdots\\ p(a),\forall yq(y)\Rightarrow p(y),q(b)\vdash p(b)\end{array}}{p(a),\forall yq(y)\Rightarrow p(y),q(b)\vdash \exists xp(x)}\,\exists-R$$

Such substitution is result of analysing $\forall yq(y)\Rightarrow p(y),\ q(b)$ part of the program. Confronted with the question of substitution for the quantified variables, an implementation might relax the uniformity constraint, recognising the advantage of the application of some other strategy: in the outlined example the application of certain left rules before the right ones. One of possible (non-uniform) proofs that can result is:

$$\dfrac{\dfrac{\dfrac{}{q(b)\vdash q(b)}\,Ax \quad \dfrac{\dfrac{\dfrac{}{p(b)\vdash p(b)}\,Ax}{p(a),p(b)\vdash p(b)}\,w-L}{p(a),p(b)\vdash \exists xp(x)}\,\exists-R}{p(a),q(b)\Rightarrow p(b),q(b)\vdash \exists xp(x)}\,\Rightarrow -L}{p(a),\forall yq(y)\Rightarrow p(y),q(b)\vdash \exists xp(x)}\,\forall-L \qquad (*)$$

The generated proof is equivalent, modulo permutations, to the starting one. After the step $(*)$ the origin of the idea for the substitution $x \leftarrow b$ is more clear.

Permutations - some problems and solutions There are techniques to perform an appropriate permutation analysis and transformations of a proof but they are all designed for analysis on paper by a human rather than by machine. Also, all existing techniques have been adapted to particular sets of inference rules. For permutations there is no general framework capable of handling permutabilities of inferences for different logical fragments and different proof systems. There are many situations where the permutabilities can not be explained by the existing definitions and conditions. One such situation is illustrated by the example below, where $\wp'$ presents a mix of $\wp$ and w rules from linear logic, and

where the possible permutation of $c?$ and $\wp'$ in the outlined subproof can not be detected using the existing definitions ([7] or Definitions 2.1-2.4 of [3]):

$$\frac{\vdash \phi, \psi, \Gamma}{\vdash \phi \wp' \psi, \Gamma, ?\delta}\ \wp' \qquad \frac{\dfrac{\vdots}{\vdash ?A, ?A, B, \Gamma}}{\dfrac{\vdash ?A, B, \Gamma}{\vdash ?A \wp' B, \Gamma, ?A}\ \wp'}\ c? \quad \Longrightarrow \quad \frac{\dfrac{\vdots}{\vdash ?A, ?A, B, \Gamma}}{\dfrac{\vdash ?A, ?A \wp' B, \Gamma, ?A}{\vdash ?A \wp' B, \Gamma, ?A}\ c?}\ \wp'$$

schema-rule

It is our contention that a more precise specification of the structure of sequent calculus rules is needed, as it is proposed by the following definition:

Definition 1. The structure of sequent calculus inference rules:

- An *active formula* of rule I is a formula in a premise, that does not exist in the conclusion. The *active part* of the i-th premise of an inference I (denoted by $\mathcal{A}_I^i$) is the (possibly empty) multiset of its active formulas.
- A *quasi-active formula* of an inference I is a formula occurrence whose presence in a premise is necessary (but not sufficient[2]) for the rule application, and which is copied unchanged into the conclusion of the rule. The *quasi-active part* of the i-th premise of an inference I (denoted by $\mathcal{QA}_I^i$) is the (possibly empty) multiset of its quasi-active formulas.
- A *principal formula* of rule I is a formula occurrence of a conclusion that does not exist in premise(s) and that is a result of the disappearance of some active formulas of I. The *principal part* of an inference I (denoted by $\mathcal{P}_I$) is the (possibly empty) multiset of its principal formulas.
- A formula occurrence of rule I that does not exist in a premise and that is not a principal formula, is called an *extra formula*. The *extra part* of an inference I (denoted by $\mathcal{E}_I$) is the (possibly empty) multiset of its extra formulas.
- The *context* of the i-th premise of an inference I (denoted by C_I^i) is the (possibly empty) multiset - complement of its active and quasi-active part.

The above definition is illustrated through examples in the Table 2 below:

Table 2.

	$\mathcal{A}^1$	$\mathcal{A}^2$	$\mathcal{QA}^1$	$\mathcal{QA}^2$	$\mathcal{P}$	$\mathcal{E}$	C^1	C^2
$\dfrac{\vdash \Gamma \quad \vdash \Delta}{\vdash \Gamma, \Delta}\ mix$							Γ	Δ
$\dfrac{\vdash A, \Gamma \quad \vdash B, \Delta}{\vdash A \otimes B, \Gamma, \Delta}\ \otimes$	A	B			$A \otimes B$		Γ	Δ
$\dfrac{\vdash A, \Gamma \quad \vdash B, \Gamma}{\vdash A \& B, \Gamma}\ \&$	A	B			$A \& B$		Γ	Γ
$\dfrac{\vdash A, \Gamma \quad \vdash B, \Delta}{\vdash \Gamma, \Delta}\ cut$	A	$A^\perp$					Γ	Δ
$\dfrac{\vdash \Gamma}{\vdash \Gamma, ?F}\ w?$						$?F$	Γ	
$\dfrac{\vdash ?F, ?F, \Gamma}{\vdash ?F, \Gamma}\ c?$	$?F$		$?F$				Γ	

According to the context of premises and conclusion, binary rules can be distinguished as: *multiplicative* rules where the context of each premise is copied unchanged into the conclusion (rules

[2] except for the exchange rule

$\otimes, mix, cut$ in above table); *strong additive* rules where the context of each premise and of the conclusion are identical (rule $\&$). Note that there are also *weak additive* rules where some part of the context of the premise and of the conclusion are identical.

We use the proposed characterisation of the structure of sequent calculi rules for the automated-oriented specification of the permutation process. Here we outline some results which have been described in details and proved formally in [8]:

- Our framework characterizes arbitrary sets of inference rules;
- The active, quasi-active, principal and extra part of an inference rule are not change by its permutation up or down, in a given proof. The context part of an inference can change during inference permutation. The appropriate changes in the context have been specified in detail (see [8]);
- "Pre-conditions" for a permutation have been focused and restricted to particular parts of inference rules and to minimally sufficient tests.
- Some dependencies between the ways of permuting rules and their generic properties and the characteristic of a given proof have been specified and proved. For example, the *upward permutation* of a strong-additive rule requires a particular proof environment. Also, the form of the permutation result depends of the generic properties of the rules in question and therefore, can be determined independently of any particular permutation.

2.2 Hypothesis elimination

This is a transformation of a *near proof* $\mathcal{D}$(derivation where not all leaves are axioms) to the actual proof which retains (at least relatively) the order of inference rules of $\mathcal{D}$ and whose conclusion is identical or contains a conclusion of $\mathcal{D}$. This is achieved by the replacement of each non-axiom leaf $\vdash S$ with a proof of $\vdash S, \Omega$ (where Ω is possibly empty multiset of formulas dependent on S and on the structure of inference rules of a given proof system) and then by, eventually, *adaptation* of subproof below $\vdash S$.

In the resulting proof of the below example the adaptation implies inserting some intermediate steps (inserted inferences are boxed).

$$\dfrac{\dfrac{\dfrac{}{\vdash a, a^{\perp}}\,Ax \qquad \vdash a, b^{\perp}}{\vdash a, a^{\perp}\&b^{\perp}}\,\&}{\vdash a \oplus b, a^{\perp}\&b^{\perp}}\,\oplus}{\vdash (a \oplus b)\wp(a^{\perp}\&b^{\perp})}\,\wp \quad \longmapsto \quad \dfrac{\dfrac{\dfrac{}{\vdash a, a^{\perp}}\,Ax}{\vdash a, a^{\perp}, ?(a^{\perp} \otimes b)}\,\boxed{\text{w}} \qquad \dfrac{\dfrac{\dfrac{}{\vdash a, a^{\perp}}\,Ax \qquad \dfrac{}{\vdash b, b^{\perp}}\,Ax}{\vdash a, b^{\perp}, a^{\perp} \otimes b}\,\boxed{\otimes}}{\vdash a, b^{\perp}, ?(a^{\perp} \otimes b)}\,\boxed{?}}{\vdash a, a^{\perp}\&b^{\perp}, ?(a^{\perp} \otimes b)}\,\&}{\vdash a \oplus b, a^{\perp}\&b^{\perp}, ?(a^{\perp} \otimes b)}\,\oplus}{\vdash (a \oplus b)\wp(a^{\perp}\&b^{\perp}), ?(a^{\perp} \otimes b)}\,\wp$$

If we allow interpretation of a non-axiom leaf $\vdash S$ as a hypothesis of, say, a proof $\mathcal{D}$ then the above transformation is known as *hypotheses elimination* [2].

Axioms are absolute truth or static knowledge. *Hypotheses* are auxiliary assumptions or lemmas of a proof which present an extra-logical knowledge in the language of a particular logical system. They are a temporary knowledge that is assumed to be true in some stages of the deduction, or belong to a catalogue of already proven results.

There are several potential benefits of allowing hypotheses in proofs. First, for the analysis of meta-properties of proofs it could be useful to reduce the branching factor at some search node or to replace some branches or sub-proofs (whose investigation is not of interest) with sequent(s) that is(are) proven in those branches/sub-proofs. It could be also useful to adapt proof systems to retain the results of a previous proof search and to apply and combine them to a new situation. Information whether a given proof can be relieved of hypothesis and what the resulting proof looks like is important. They have a direct impact, for example, on a Cut elimination because mostly, in the systems with a Cut elimination property, the Cut can be eliminated from proofs without hypotheses and just reduced to a certain normal form for proofs with hypotheses.

Elimination of a hypothesis $\vdash \mathcal{H}$ from a proof Π is the problem of searching for a proof (without any hypotheses) of hypothesis $\vdash \mathcal{H}$, which will replace it in Π. For decidable proof systems it is possible to find, at least, one such proof iff $\vdash \mathcal{H}$ is provable. It seems reasonable to expect that in a decidable proof system every provable hypothesis $\vdash \mathcal{H}$ can be eliminated and replaced by its proof. The situation is more complicated when a hypothesis $\vdash \mathcal{H}$ (or $\mathcal{H}_2 \vdash \mathcal{H}_1$) is an unprovable sequent, or a given proof system is not decidable. It is interesting, in such situations, to consider the finding of the simplest proof (without hypotheses) whose conclusion incorporates $\mathcal{H}$ $(\mathcal{H}_1, \mathcal{H}_2)$. It is obvious, that this process depends on the hypothesis itself, as well as on the structure of inference rules of a given proof system. Andreoli has shown this on the example of one fragment of propositional classical logic [2].

Developing general methods for hypothesis elimination is research still in progress. It is possible to generalise some dependences and to extract certain, general structures of rules that are *suitable* and *useful* for hypothesis elimination. For example, the structure of weakening rule (in LK), the structure of mix rule, structure of some multiplicative rules, etc.

Also, given the considerations above we can address another interesting question: what is the minimal extension (that is sufficient, a guarantee) to provide provability of a given unprovable sequent $\mathcal{S}$ of a given proof system? The example below points out some possible extensions and additions of the unprovable sequent $\vdash p, q$ in a different logic:

classical logic:

$$\dfrac{\dfrac{}{p \vdash p}\ Ax}{p \vdash p, q}\ w$$

two-sided linear logic:

$$\dfrac{\dfrac{}{p \vdash p}\ Ax \quad \dfrac{}{q \vdash q}\ Ax}{p \wp q \vdash p, q}\ \wp \qquad \dfrac{}{\vdash p, q, \top}\ \top \qquad \dfrac{}{0 \vdash p, q}\ 0$$

3 Conclusions and Future Work

We have outlined the first steps in the investigation of the automation of the design of LP languages. The aim is to develop a logic-independent system, which will contain various implementing and proof-theoretics techniques together with the general strategic planner, capable of determining the most appropriate way to assemble the various pieces into a LP system, but in a formalism and framework which is designed and intended for humans.

The first steps towards such an automated analyser are dedicated to machine-oriented specifications for the encapsulation, manipulation and automation of existing LP strategies. Various properties of search strategies have direct influence on the design of LP languages. We exploit some proof manipulations and their usage in the analysis of the particular search strategies. We have proposed a more detailed specification of inference rules, for a better understanding of their structure and their generic properties. Making use of the generic properties of inference rules rather than particular instances of rules will allow analyses on various logics and various classes of formulae within each logic. On the basis of introduced specifications we have generalised some definitions and meta-rules for the permutation analyses. Also, it seems that it is possible to exploit some similarities in the structure of sequent calculi inference rules from different proof systems for the extenuation and generalisation of some strategies or techniques originally designed and dedicated to particular sets of rules. For example, proof-nets, Girard's "parallel syntax for proof theory" [4], are among the most fascinating constructions for a concise graph-theoretic representation of deductions by "decontextualization of inferences". They have arisen and are based on some fragments of linear logic, principally first order $\otimes, \wp, cut$ and $\otimes, \wp, cut, mix$ fragments. The question that arises is: are the same principles for the construction and analysis of proof-nets in, say, $\wp, cut, mix$ fragment of linear logic applicable in $\vee, cut, mix$ fragment of LK, in regard to the same structure (same number of active, quasi-active, principal and extra formulas, the same type of restrictions on the structure of involved formulas and multisets, etc) of rules:

$$\text{CLL:}\quad \dfrac{\vdash A, B\ ;\ ;\ \Gamma\ ;}{\vdash A \wp B\ ;\ ;\ \Gamma\ ;}\ \wp \qquad\qquad \text{LK}\quad \dfrac{\vdash A, B\ ;\ ;\ \Gamma\ ;}{\vdash A \vee B\ ;\ ;\ \Gamma\ ;}\ \vee$$

References

1. Andreoli J.-M.: Logic Programming with focusing proofs in linear logic, Journal of Logic and Computation, 2(3) (1992) 297-347.
2. J.M. Andreoli, R. Pareschi: Logic programming with sequent systems: A linear Logic Approach. In P.Schröder-Heister ed., Proceedings of Workshop to Extensions of Logic Programming, Tübingen, LNAI 475 (1989) 1-30.
3. Galmiche D. and Perrier G.: On proof normalisation in Linear Logic, Theoretical Computer Science 135 (1994) 67-110.
4. Girard J.-Y.: Proof-nets: the Parallel syntax for Proof-theory, Logic and Algebra, Marcel Dekker, New York (1995).
5. Harland J.: A proof-theoretic Analysis of Goal-directed Provability, Journal of Logic and Computation, Vol.4 No 1 (1994) 69-88.
6. Hodas J.S. and Miller D. *Logic programming in a Fragment of Intuitionistic Linear Logic*, Journal of Information and Computation 110:2 (1994) 327-365.
7. Kleene S.C.: Permutability of Inferences in Gentzen's Calculi LK and LJ, Memoirs of the American Mathematical Society 10 (1952) 1-26.
8. Lutovac T., Harland J.: Towards the Automation of the Design of Logic Programming Languages, Technical Report TR-97-30, Department of Computer Science, RMIT University (1997).
9. Miller D.: A logical analysis of modules in logic programming, Journal of Logic Programming 6 (1989) 79-108.
10. Miller D., Nadathur G., Pfenning F., Ščedrov A.: Uniform proofs as a Foundation for Logic Programming, Annals of Pure and Applied Logic 51 (1991) 125-157.
11. Miller D.: Forum: A multiple-conclusion specification-logic, Theoretical Computer Science 165, 201-232.
12. Nadathur G. and Miller D.: An overview of λProlog, In Kenneth A. Bowen and Robert A. Kowalski, editors, Fifth International Logic Programming Conference, Seattle, Washington, August 1988, MIT Press, 810-827.
13. Pym J.D., Harland A.J.: A Uniform Proof-theoretic Investigation of Linear Logic Programming, Journal of Logic and Computation, Vol. 4 No 2, (1994) 175-207.
14. Pfenning F.: Analytic and Non-analytic Proofs, in M.E. Stickel ed., 10th International Conference on Automated Deduction, Kaiserslautern, FRG, LNAI 449 (1990) 394-413.
15. Sterling L., Shapiro E.: The Art of Prolog, MIT Press, 1986.
16. Tammet T.: Proof Search Strategies in Linear Logic, Journal of Automated Reasoning no. 12, (1994) 273-304.
17. Winikoff M., Harland J.: Implementing the Linear Logic Programming Language Lygon, Proceedings of the International Logic Programming Symposium, Portland, Oregon, December, 1995, 66-80.
18. Winikoff M., Harland J.: Deriving Logic Programming Languages, Technical Report 95/26, Department of Computer Science, The University of Melbourne, (1996).

This article was processed using the LaTeX macro package with LLNCS style