Formulae-as-Resources Management for an Intuitionistic Theorem Prover

D. Galmiche and D. Larchey-Wendling
LORIA - Université Henri Poincaré
Campus Scientifique, BP 239
Vandœuvre-lès-Nancy, France

Abstract

This paper outlines new concepts for an alternative implementation of the intuitionistic contraction-free LJT system (and consequently proof search) in imperative programming languages with a good and efficient management of formulae (as resources). Logic programming languages provide natural support to implement automated proof search, without necessity to have explicit knowledge about the form and the number of formulae arising in LJT proofs. By the introduction of a new notion of subformula (and of subformula property) for the LJT system, we obtain interesting and usable results about the possible management of formulae in a proof. A derived structure (a direct acyclic graph), including sharing of subformulae, is proposed to deal with formulae during the proof search and the application of logical rules. The corresponding proof search method is independent of the possible strategies. Therefore we can obtain an efficient implementation of LJT in imperative programming languages that is based on a formulae-as-resources management.

1 Introduction

A sequent-style formulation of a logic is convenient to automate proof search with inference rules being local operations on proofs. Its implementation is simple in logic programming languages because they provide support for depth-first proof search and also considerably simplify the implementation of a logical calculus. But they have a limited support for instance for user interfaces that could make them unsuitable for larger applications [11]. To provide a simple implementation of a logical calculus (and of proof search methods) in imperative programming languages like C or Java, we have to consider new techniques leading to the clarity of the implementation (for instance, success continuations in [11]) but also to a good and efficient explicit management of formulae (considered here as resources). The potential support for user interfaces or for a natural parallelization of the proof search process could be expected. In this paper we focus on intuitionistic logic, for which the LJ sequent calculus is a simple formulation having the subformula property, namely, (cut-free) proofs only contain occurrences of subformulae of the formulae composing the initial sequent. On the other hand, it lacks termination and thus implementing this calculus involves some kind of look-checking mechanism to ensure the termination of computation. In fact, the LJT contraction-free sequent calculus [2] has the nice property that a backward search does not loop and it is a system sound and complete w.r.t. provability in LJ. The rules of LJT are presented in figure 1, X corresponding to atomic formulae, A, B, G representing for formulae and Γ a multiset a formulae. The LJT prover, implemented in Prolog [2], is based on an analytic approach where the size of the formulae decreases but a potential big number of subformulae might appear, and the lack of a subformula property is not a real obstacle for it. But in this case, we have no explicit knowledge on the formulae used during the proof search or on the depth of proofs, the formulae being implicitly managed by Prolog. Our aim here is not to analyze the advantages or limits of the LJT system compared with other intuitionistic theorem provers [8, 10] but to consider some new concepts or techniques for a clear and efficient implementation in an imperative language.

Our approach includes an explicit and management of the formulae (considered as resources) and it is based on new concepts and results such as a new notion of subformula in LJT. It provides knowledge

$$\frac{}{\Gamma, X \vdash X}\ \mathrm{Id} \qquad \frac{}{\Gamma \vdash \mathsf{T}}\ \mathsf{T} \qquad \frac{}{\Gamma, \mathsf{F} \vdash G}\ \mathsf{F}$$

$$\frac{\Gamma, A, B \vdash G}{\Gamma, A \wedge B \vdash G}\ \wedge\text{-L} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\text{-R} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee\text{-R}_1$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee\text{-R}_2 \qquad \frac{\Gamma, A \vdash G \qquad \Gamma, B \vdash G}{\Gamma, A \vee B \vdash G}\ \vee\text{-L} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ \supset\text{-R}$$

$$\frac{\Gamma, X, B \vdash G}{\Gamma, X, X \supset B \vdash G}\ \supset\text{-L}_1 \qquad \frac{\Gamma, A \supset (B \supset C) \vdash G}{\Gamma, (A \wedge B) \supset C \vdash G}\ \supset\text{-L}_2$$

$$\frac{\Gamma, A \supset C, B \supset C \vdash G}{\Gamma, (A \vee B) \supset C \vdash G}\ \supset\text{-L}_3 \qquad \frac{\Gamma, B \supset C \vdash A \supset B \qquad \Gamma, C \vdash G}{\Gamma, (A \supset B) \supset C \vdash G}\ \supset\text{-L}_4$$

Figure 1: The LJT calculus

about the number and form of the formulae in a proof and also allows to construct a priori the formulae arising by backward application of inference rules. For a given sequent S, we can prove that the number of subformulae in a proof of S is less than twice the S size, this result justifying the effective use of this notion. All this information about the formulae involved in the proof search is stored in a directed acyclic graph (*dag*) that represents both the subformula relation and the logical structures of the formulae. Then it is easier to compare, duplicate or decompose formulae and also to propose a complementary compact encoding of the sequents. The corresponding proof search method for LJT is based on the use of this *dag* and is independent of the proof search strategy that is defined at another level. From this method and these results for LJT we have also considered, with a similar approach, proof search in the multi-succedent calculus LJT* [2] and also in the related refutation calculus CRIP [7]. We emphasize the importance of the explicit management of formulae as resources (justified by the new subformula property) that leads to efficiency and also to a possible and natural parallelization of the proof search process.

2 New concepts for LJT proof search

Let us consider some remarks on proof search in LJT. At first, it is not evident that the premise sequents in a rule are always simpler than the concluding sequent. For instance, in the $\supset$-L$_3$ rule, the premises are not smaller (in the usual meaning) than the conclusion. But they are in fact if we consider the multiset ordering (see [1]). LJT has the property that the depth-first proof search does not loop and is also a system sound and complete w.r.t. provability [2]. However, it does not satisfy the *subformula property:* for instance, in rule $\supset$-L$_4$, the formula $B \supset C$ is introduced in the left premise even if it is not (in general) a subformula of $(A \supset B) \supset C$. Moreover some rules duplicate formulae as in the $\supset$-L$_3$ rule where C is duplicated and in rule $\supset$-L$_4$ where B is duplicated. Therefore, the idea to share formulae naturally arises so that the duplication would be harmless in terms of space. But we have to know which are the formulae that might appear during such backward proof search. For that, we propose an adequate definition of subformula and show how to compute a superset of the formulae appearing in an LJT proof and finally we analyze the implementation issues.

2.1 A LJT subformula property

The usual notion of subformula does not provide a superset of the formulae appearing in an LJT proof, i.e., LJT does not satisfy the subformula property, because for instance of the $\supset$-L$_4$ rule. We propose to define an adequate notion of subformula for LJT and to prove that this new notion is reasonable, i.e., there are not too many subformulae. Let us give at first a useful modified notion of the weight of formulae as follows:

Definition 1
The weight $\omega(A)$ of formulae is defined by the following rules:
$\omega(X) =_{def} 2$ *(X is atomic),*
$\omega(A \otimes B) =_{def} \omega(A) + \omega(B) + \delta_\otimes$ *($\otimes$ binary operator) with $\delta_\wedge = 2$, $\delta_\vee = 2$ and $\delta_\supset = 1$*

It is easy to check that for each LJT rule, the principal formula (the formula that is decomposed) is always replaced by one or two smaller (w.r.t. ω) formulae. Thus a proof tree cannot have an infinite branch and then the depth-first backward proof search algorithm always terminates in LJT (see [2]).
Let us consider now the case of the formula $(A \supset B) \supset Q$ where A, B, Q might be compound formula as well. Two rules might have this formula as the principal one, namely $\supset$-L$_4$ and $\supset$-R. By backward application of the both rules we obtain new formulae: $B \supset Q, A \supset B, Q$ for $\supset$-L$_4$ and $A \supset B, Q$ for $\supset$-R. Thus, we define the subformulae of $(A \supset B) \supset Q$ to be itself and $A \supset B, B \supset Q, Q$ and all their subformulae (recursively). We point out that the weight of $A \supset B$, $B \supset Q$ and Q are smaller than the weight of $(A \supset B) \supset Q$ and then the recursive call is well-founded. By considering all the other cases, we get, by induction on $\omega(A)$, the following definition of $\mathrm{s}_n(\mathrm{A})$, the set of subformulae of A:

Definition 2
The notion of sub-formulae *of LJT is inductively defined as follows:*

$\mathrm{s}_n(X) =_{def} \{X\}$ *(X is atomic)*
$\mathrm{s}_n(P \wedge Q) =_{def} \mathrm{s}_n(P) \cup \mathrm{s}_n(Q) \cup \{P \wedge Q\}$
$\mathrm{s}_n(P \vee Q) =_{def} \mathrm{s}_n(P) \cup \mathrm{s}_n(Q) \cup \{P \vee Q\}$
$\mathrm{s}_n(X \supset Q) =_{def} \mathrm{s}_n(X) \cup \mathrm{s}_n(Q) \cup \{X \supset Q\}$ *(X is atomic)*
$\mathrm{s}_n((A \wedge B) \supset Q) =_{def} \mathrm{s}_n(A \wedge B) \cup \mathrm{s}_n(Q) \cup \{(A \wedge B) \supset Q\} \cup \mathrm{s}_n(A \supset (B \supset Q))$
$\mathrm{s}_n((A \vee B) \supset Q) =_{def} \mathrm{s}_n(A \vee B) \cup \mathrm{s}_n(Q) \cup \{(A \vee B) \supset Q\} \cup \mathrm{s}_n(A \supset Q) \cup \mathrm{s}_n(B \supset Q)$
$\mathrm{s}_n((A \supset B) \supset Q) =_{def} \mathrm{s}_n(A \supset B) \cup \mathrm{s}_n(Q) \cup \{(A \supset B) \supset Q\} \cup \mathrm{s}_n(B \supset Q)$

We remark that s_n is an extension of the usual notion of subformula — immediate proof by induction on $\omega(A)$. It is also trivial to prove that LJT now satisfies the s_n-subformula property. What is non-trivial is how many s_n-subformulae of A there are ? In fact, we are going to prove that the number of s_n-subformulae of A — denoted by $\#\mathrm{s}_n(A)$ — is less than $\omega(A)$ and thus than $2\,\overline{A}$ ($\overline{A}$ being the usual notion of size).

Theorem 3
Let S be a sequent, the number of distinct formulae appearing in a LJT proof of S is less than twice the size (in the usual meaning) of S.

Indeed, we can prove that $\omega(A) \leqslant 2\,\overline{A}$, by induction on A, simply because the weight of the connectives is always less than 2. Then the number of formulae occuring in a proof of a sequent S is less than the number of subformulae of all the formulae occuring in S which itself is less than twice the size of S.
The proof of the lemma saying that $\#\mathrm{s}_n(A) \leqslant \omega(A)$ is done by giving a new definition s'_n of s_n by structural induction (opposed to the previous general induction) which is linear. For that we suppose that s'_n can be defined in the following way, for some expression Φ to be discovered:

$\mathrm{s}'_n(X) =_{def} \{X\}$ for X atomic
$\mathrm{s}'_n(A \otimes B) =_{def} \mathrm{s}'_n(A) \cup \mathrm{s}'_n(B) \cup \{A \otimes B\}$ for $\otimes \in \{\wedge, \vee\}$
$\mathrm{s}'_n(A \supset B) =_{def} \Phi_{A,B} \cup \mathrm{s}'_n(B) \cup \{A \supset B\}$

The linearity means that the components of the decomposed formula appear at most once in the new definition and the linearity of s'_n is obtained as soon as $\Phi_{A,B}$ is linear in A too.
We define $\Phi_{P,Q}$ by primitive induction on P (that is the trick):

$\Phi_{X,Q} =_{def} \{X, X \supset Q\}$ for X atomic
$\Phi_{A \wedge B,Q} =_{def} \Phi_{A,B \supset Q} \cup \Phi_{B,Q} \cup \{A \wedge B, (A \wedge B) \supset Q\}$
$\Phi_{A \vee B,Q} =_{def} \Phi_{A,Q} \cup \Phi_{B,Q} \cup \{A \vee B, (A \vee B) \supset Q\}$
$\Phi_{A \supset B,Q} =_{def} \Phi_{A,B} \cup \Phi_{B,Q} \cup \{(A \supset B) \supset Q\}$

With this new definition s'_n, the property $\#\mathrm{s}'_n(A) \leqslant \omega(A)$ is easy to prove. We first show , by induction on P, that $\#\Phi_{P,Q} \leqslant \omega(P)$ and then that we have $\#\mathrm{s}'_n(A) \leqslant \omega(A)$, by induction on A.
It remains to prove that this new definition is coherent with the older one, i.e., $\mathrm{s}'_n(A) = \mathrm{s}_n(A)$ for all A. For that it is sufficient to prove that $\mathrm{s}_n(P \supset Q) = \Phi_{P,Q} \cup \mathrm{s}_n(Q)$. Let us notice that the old definition s_n is considered here and that the proof is finally done by induction on P. For the case of $P \equiv A \supset B$ we have

$$
\begin{aligned}
\Phi_{A \supset B,Q} \cup \mathrm{s}_n(Q) &= \Phi_{A,B} \cup \{(A \supset B) \supset Q\} \cup \big(\Phi_{B,Q} \cup \mathrm{s}_n(Q)\big) \\
&= \Phi_{A,B} \cup \{(A \supset B) \supset Q\} \cup \mathrm{s}_n(B \supset Q) && (1) \\
&= \Phi_{A,B} \cup \{(A \supset B) \supset Q\} \cup \big(\mathrm{s}_n(B \supset Q) \cup \mathrm{s}_n(B) \cup \mathrm{s}_n(Q)\big) && (2) \\
&= \Phi_{A,B} \cup \big(\mathrm{s}_n(B) \cup \mathrm{s}_n(B \supset Q)\big) \cup \{(A \sup setB) \supset Q\} \cup \mathrm{s}_n(Q) \\
&= \mathrm{s}_n(A \supset B) \cup \mathrm{s}_n(B \supset Q) \cup \{(A \supset B) \supset Q\} \cup \mathrm{s}_n(Q) \\
&= \mathrm{s}_n((A \supset B) \supset Q)
\end{aligned}
$$

The transition from (1) to (2) is justified, from the definition of s_n, by the fact that $\mathrm{s}_n(B)$ and $\mathrm{s}_n(Q)$ are both included in $\mathrm{s}_n(B \supset Q)$. The other cases ($P \equiv X$, $P \equiv A \wedge B$ or $P \equiv A \vee B$) are similar.
As a consequence of this new subformula property, we propose a specific encoding of the formula (or sequent) into a *dag* (direct acyclic graph) and an adequate proof search algorithm.

2.2 Encoding

Considering theorem 3, it seems reasonable to compute the set of s_n-subformulae of a given sequent before doing the proof search. An appropriate structure to encode the s_n-subformulae is a graph where nodes are formulae and edges represent the s_n-subformula relation on formulae. Then it represents the way a formula is decomposed through the computation of s_n. In fact, we obtain a dag (direct acyclic graph) from the initial formula to prove, as illustrated in the following example with the sequent $\vdash ((A \supset A) \supset C) \supset C$ where A, C are atomic formulae:

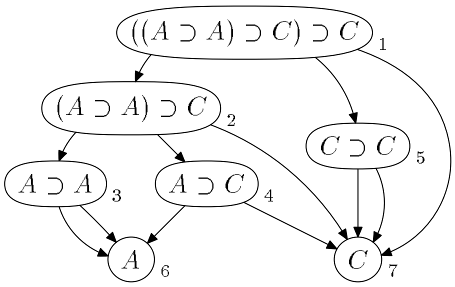

Figure 2: *dag* representing the subformulae of $\vdash ((A \supset A) \supset C) \supset C$

We have three initial decompositions, namely $(A \supset A) \supset C$, $C \supset C$ and C. Such graph representation allows to encode the sequents in a very compact way. For example, the sequent $A \supset C, C \vdash C \supset C$ can be encoded by $4, 7 \vdash 5$. More precisely, a sequent $\Gamma \vdash A$ of LJT is a multiset Γ of formulae representing the hypotheses together with a formula A representing the conclusion. But in the hypotheses the number of occurrences of formulae does not matter because of the admissibility of structural rules: either one has at least one occurrence of B in Γ, or no occurrence of B in Γ. Then, we consider Γ as a set but consequently the inference rules are not so easy to represent. However, the translation is quite immediate. For example the $\wedge$-L rule is translated as follows:

$$
\frac{\Gamma, A, B \vdash G}{\Gamma, A \wedge B \vdash G} \quad \rightsquigarrow \quad \frac{\Gamma \cup \{A, B\} \vdash G}{\Gamma \cup \{A \wedge B\} \vdash G} \ \text{where } A \wedge B \notin \Gamma
$$

As Γ can be considered as a set and as we already know all the formulae appearing in Γ (since we have computed the s_n-subformulae before starting the proof), Γ is naturally encoded as a boolean function expressing whether a formula is in Γ or not. The concluding part is simply coded by a reference into the corresponding *dag*. To summarize, we have a graph $G = (\{1, \ldots, n\}, E)$ where all the formulae that might appear during the computation are stored. A sequent $\Gamma \vdash A$ is a pair (γ, a) where $\gamma : \{1, \ldots, n\} \rightarrow \texttt{bool}$ is the characteristic function of Γ and $a \in \{1, \ldots, n\}$ is a reference in G.

2.3 Proof search

Let us illustrate the induced proof search with the following sequent: $\vdash ((A \supset A) \supset C) \supset C$. The first step consists in building the *dag* of subformulae of the sequent, which is represented in figure 2 and then in encoding the sequent in $\vdash 1$. In this step, there is only one possible principal formula, namely 1. Looking into the *dag* we know which LJT rule can be applied — there is always at most one rule for each principal formula. In our example the rule is $\supset$-R. Therefore we obtain the sequent $2 \vdash 7$. Now we have two possible principal formulae, namely 2 and 7 and with the *dag* we know that 7 is atomic and so, as it is not present in premises, no rule is applicable with it as principal formula. We also have to try 2, that has the form $(\supset)$ (information immediately given by the *dag*) and then we can apply rule $\supset$-L_4 and obtain two sequents: $4 \vdash 3$ and $7 \vdash 7$. The last sequent is an axiom and for the other one, two principal formulae are possible. In the case of 4, we could apply rule $\supset$-L_1 as 6 is an atomic formula but this is not possible because 6 is not in the premises. Thus the only applicable rule is $\supset$-R for which the principal formula is 3. We get the sequent $6, 4 \vdash 6$ which is again an axiom. Here is the resulting proof (which is unique in this case) in a compact form and in a more human readable form:

$$\dfrac{\dfrac{\dfrac{6, 4 \vdash 6}{4 \vdash 3} \quad 7 \vdash 7}{2 \vdash 7}}{\vdash 1} \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{A, A \supset C \vdash A}{A \supset C \vdash A \supset A} \quad C \vdash C}{(A \supset A) \supset C \vdash C}}{\vdash ((A \supset A) \supset C) \supset C}$$

Finally, let us discuss now the possible strategies for the backward application of rules, knowing that it is not possible, for instance to design for LJT a proving method based on the systematic search of uniform proofs [6]. Given a sequent, the first step is obviously to test whether it is an axiom or whether it is provable with the T or F rules. Then we observe that all the rules except $\supset$-L_4 and $\vee$-R_i are invertible, i.e, the premises are provable if the conclusion is provable. This means that when one of these rules is applied, it is not necessary to backtrack and to try another rule for the same sequent. Therefore, a good strategy consists in applying first the invertible rules until the only rules that can be applied are $\supset$-L_4 and $\vee$-R_i. The order of application of invertible rules can also be tuned. Secondly, a non-invertible rule is applied, meaning that an "uncertain" branch is explored. Should the premises be unprovable, another non-invertible rule might be tried on the same sequent. Moreover, one cannot decide that the $\vee$-R_i rules can be applied before $\supset$-L_4 rules (or the converse). Such non-determinism has to be handled.

With our approach, the rules of LJT become partial functions on such pairs, returning a list of pairs representing premises. This list is empty for rules Id, T and F and then the proof is closed in the current branch. Some rules, like $\vee$-L, duplicate the context but more compact the context is, then shorter the duplication process is. Other rules, like Id, require to test whether a formula X is in the hypotheses part of the sequent. It can be done in constant time, even if X is not atomic. This comparison is usually done in linear time on the size of the hypotheses part of the sequent. Consequently, we get an optimization in space and time because of our encoding of sequents as characteristic functions.

The non-deterministic application of these functions until each branch is closed corresponds to an obvious proof-search algorithm. However, optimizations can be proposed to reduce some non-determinisms, by taking into account some strategies mentioned in section 1. Invertible rules are applied first and an order of application can be fixed such that the application of the rules that duplicate the context, like $\vee$-L, is postponed (with space save as a consequence). Other optimizations have been done like the generalization of the rules Id and $\supset$-L_1 for the case where X is not atomic. With our representation, the equality test between two formulae is efficient if whether they are atomic or not.

3 Concluding remarks

A first implementation of LJT based on these results has been developed in C language, using dynamic allocation of structures. To test it, we have considered various formulae of propositional intuitionistic logic and in each case the time of computation was under 20 milliseconds. In fact the tests confirm the interest of such implementation. From the case of LJT, we can expect that such technique would lead to clear and efficient implementation of provers in imperative languages, in addition to other known techniques [11]. We have considered a similar approach for proof search and implementation of a contraction free multi-succedent calculus for intuitionistic logic, knowing that such calculus could be convenient for implementation and proof search, and also for the related refutation calculus CRIP [7]. Then comparisons with the related techniques proposed in [9] will be possible from both provability and refutation point of views. More comparisons about the performances of our intuitionistic prover with others systems [8, 10] would be also necessary and some generic strategies, based for instance on pruning techniques, could optimize our implementation. Another attempt could be to consider similar logical systems and implementations for fragments of linear logic, for instance intuitionistic linear logic (ILL), and to compare with improved proof search methods [3, 5]. Let us mention that we have already used a similar notion of *dag* to represent subformulae and then design a polynomial decision procedure for the additive linear logic [4]. As in this specific case, it can lead to an adequate implementation of a tableau method. The new subformula property could be also used for studies about a new tableau method for the LJT system.

References

[1] N. Derschowitz and Z. Manna. Proving termination with multiset ordering. *Communications of ACM*, 22:465–479, 1979.

[2] R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, 57:795–807, 1992.

[3] D. Galmiche and E. Boudinet. Proof search for programming in intuitionistic linear logic. In *CADE-12 Workshop on Proof search in type-theoretic languages*, Nancy, France, June 1994.

[4] D. Galmiche and J.Y. Marion. Semantic proof search methods for ALL - a first approach -. In *4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, St Goar am Rhein, Germany, May 1995.

[5] D. Galmiche and G. Perrier. Foundations of proof search strategies design in linear logic. In *Logic at St Petersburg '94, Symposium on Logical Foundations of Computer Science, LNCS 813*, pages 101–113, St Petersburg, Russia, July 1994.

[6] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[7] L. Pinto and R. Dyckhoff. Loop-free construction of counter-models for intuitionistic propositional logic. In Behara and al., editors, *Symposia Gaussiana*, pages 225–232, 1995.

[8] N. Shankar. Proof search in the intuitionistic sequent calculus. In *11th Conference on Automated DEduction, LNAI 607*, pages 522–536, Saratoga Springs, June 1992.

[9] A. Stoughton. Porgi: a proof-or-refutation generator for intuitionistic propositional logic. In D. Galmiche, editor, *CADE-13 Workshop on Proof search in type-theoretic languages*, pages 109–116, Nancy, June 1996.

[10] T. Tammet. A resolution theorem prover for intuitionistic logic. In *13th Conference on Automated Deduction, CADE-13, LNAI 1104*, pages 2–16, New Brunswick, NJ, USA, 1996.

[11] C. Urban. Implementation of proof search in the imperative programming language pizza. In *Int. Conference on Analytic Tableaux and Related Methods, TABLEAUX'98, LNAI 1397*, Tilburg, The Netherlands, May 1998.