STRIP: Structural sharing for efficient proof-search

D. Larchey-Wendling and D. Méry and D. Galmiche

LORIA UMR 7503 - Université Henri Poincaré
Campus Scientifique, BP 239 Vandœuvre-lès-Nancy, France

Abstract. The STRIP system is a theorem prover for intuitionistic propositional logic with two main characteristics: it deals with the duplication of formulae during proof-search from a fine and explicit management of formulae (as resources) based on a structural sharing and it builds, for a given formula, either a proof or a countermodel.

1 Introduction

In recent years there was a renewed interest in proof-search for constructive logics like intuitionistic logic (IL), mainly because of research in intuitionistic type theories and their relationships with programming through proof-search. Differents methods (based on resolution, connections, translation in classical logic, constraints calculus) and implementations have been already designed for IL but our aim in this work is to focus on two main problems: to avoid the duplication of formulae during proof-search and to efficiently build countermodels in case of non-provability. Firstly, we consider the propositional fragment of IL (IPL) but our main goal is to define structural solutions general enough to be applicable to other substructural or intermediate logics [1]. A good and efficient explicit management of formulae (as resources), both in the logical system and in the implementation, is important to have reliable and efficient implementation techniques of logical calculi (and connected proof-search methods), for instance in imperative programming languages like C or Java. We have already studied this point in [3] for the contraction-free sequent calculus LJT [2] for which there are refinements in order to solve the duplication problem [1,2,5]. The STRIP system, available at `http://www.loria.fr/~larchey/STRIP` decides the provability of a given IPL sequent and then builds a proof or a countermodel (as a Kripke tree). It is based on a new logical system, named SLJ [4] and on a structural solution of the duplication problem (without the introduction of new formulae and variables like in [2,5]). In order to illustrate and emphasize the interest and the results of structural sharing and its implementation we have compared, from various IPL formulae, the STRIP system with Porgi[1], a similar prover for IPL written in SML, and then with the **ft**[2] system that is not based on LJT but is written in C like our system.

[1] available at `http://www.cis.ksu.edu/~allen/porgi.html`
[2] available at `http://www.sics.se/isl/ft.html`

$$\frac{\Gamma, A, \boxed{B} \to C \vdash \boxed{B} \qquad \ldots}{\Gamma, (A \to B) \to C \vdash G}\ [(\to)\to] \qquad\qquad \frac{\Gamma, A \to \boxed{C}, B \to \boxed{C} \vdash G}{\Gamma, (A \vee B) \to C \vdash G}\ [(\vee)\to]$$

Fig. 1. Rules and duplication

2 Formulae duplication and sharing

In LJT [2], two kinds of formulae duplication appear even though the system is contraction-free: these are illustrated in the rules of figure 1. Let us give a brief overview of the results and techniques presented in [4]. The duplication on the lhs is treated as in [5], introducing a mark and a so-called *boxed sequent*: $\Gamma, A, B^\star \to C \vdash \blacksquare$ with the intended meaning of $\Gamma, A, B \to C \vdash B$.

The duplication on the rhs is addressed on a *structural* way. Whereas the lhs part of a sequent is usually considered to be a *flat* list a formulae, we use a list of trees, i.e. a *formulae-indexed forest*. Thus, the sequents are represented by specific trees in which formulae are paths from roots to leaves and logical operations are operations on the tree leaves.

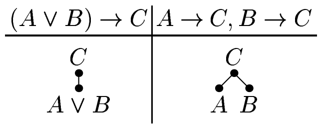

Fig. 2. Logical rule

The problem of duplication is then a problem of structural sharing in such trees. Similar ideas can be applied to a refutability system in order to generate countermodels in case of non-provability [6]. By such an approach, there is no formulae duplication anymore: each subformula is used at most once in a proof-search branch. During proof-search the structure of the forest changes but not its size. The STRIP system, that provides proofs or countermodels for IPL formulae, is based on structural sharing techniques with the following results: no dynamic memory allocation, a finer control on the resources and a $\mathcal{O}(n \log n)$-space algorithm for provability [4].

3 Structures and strategies

The formulae-indexed forest data structure has to be implemented in such a way that the *administrative*, *logical* operations, and those related to *strategies* take the less time possible. The leaves are chained into a list to provide fast access to active formulae (which are those indexing leaves). The STRIP system includes two different implementations of this structure depending on the way to deal with the operations of cutting or pasting subtrees. In the first one, called *lrmost*, one memorizes for each node the indexes of the leftmost and rightmost leaves under this node. In the second one, called *index-scope*, one computes for each node its scope that is the greatest index among the indexes that could potentially be under the current node. Moreover the system proposes two proof-search methods (or *strategies* for the choice of the leaf to develop at each step). The strategy, called *first-leaf*, chooses the first active leaf from a left-to-right search in the set of leaves. The strategy, called *rule-prec*, also considers such a left-to-right search

4	$((A \to (B \vee (B \to C))) \to C) \to C$
6	$A \to B \to ((A \to B \to C) \to C) \to (A \to B \to C)$
7	$(((A \wedge B) \vee C) \to (C \vee (C \wedge D))) \to (\neg A \vee ((A \vee B) \to C))$
8	$((A \leftrightarrow B \vee A \leftrightarrow C \vee B \leftrightarrow C) \to (A \wedge B \wedge C)) \to (A \wedge B \wedge C)$
13	$\neg\neg((\neg A \to B) \to (\neg A \to \neg B) \to A)$
14	$(\neg(A \to (B \vee C))) \to (B \vee C) \to A$
15	$\neg\neg A \vee (A \to \neg\neg B \vee (B \to \neg\neg C \vee (C \to (\neg\neg D \to D) \vee \neg D \vee \neg\neg D)))$
20	$(((G \to A) \to J) \to D \to E) \to (((H \to B) \to I) \to C \to J)$ $\to (A \to H) \to F \to G \to (((C \to B) \to I) \to D) \to (A \to C)$ $\to (((F \to A) \to B) \to I) \to E$
21	$\neg\neg(((A \leftrightarrow B) \leftrightarrow C) \leftrightarrow (A \leftrightarrow (B \leftrightarrow C)))$
22	$((\neg\neg(\neg A \vee \neg B) \to (\neg A \vee \neg B)) \to (\neg\neg(\neg A \vee \neg B) \vee \neg(\neg A \vee \neg B)))$ $\to (\neg\neg(\neg A \vee \neg B) \vee \neg(\neg A \vee \neg B))$
24	Pigeonhole 2-3
25	Pigeonhole 3-4

		Time in 10^{-4} seconds					Number of operations					
	p/u	T_{porgi}	T^{lr}_{fl}	T^{is}_{fl}	T^{lr}_{rp}	T^{is}_{rp}	SS_{fl}	SS_{rp}	SC^{lr}_{fl}	SC^{is}_{fl}	SC^{lr}_{rp}	SC^{is}_{rp}
4	p	0.220	0.280	0.170	0.290	0.210	9	9	71	41	76	46
6	u	0.480	0.390	0.270	0.290	0.230	14	9	104	76	87	71
7	u	0.540	0.280	0.170	0.310	0.240	11	11	82	48	93	59
8	u	1.240	1.180	0.840	0.780	0.570	42	24	253	155	191	131
13	p	0.600	0.570	0.460	0.470	0.340	20	15	133	81	124	75
14	p	0.420	0.390	0.300	0.260	0.200	13	9	99	55	69	44
15	u	0.910	0.560	0.440	0.630	0.510	26	26	158	117	190	149
20	p	10.500	16.880	13.700	2.720	2.220	573	69	4253	3129	900	757
21	p	10.910	3.680	2.820	3.500	2.650	126	97	723	463	918	680
22	u	3.500	15.980	12.660	8.930	7.130	479	220	4104	2935	2677	2157
24	p	21.570	1.390	1.380	1.520	1.450	100	99	452	424	586	560
25	p	225300	26.310	25.550	32.860	31.850	2248	2238	10432	10180	15488	15256

Fig. 3. Comparison : STRIP vs Porgi

but the set of leaves is split in different groups having different priorities and then it selects the first active leaf in the group with the highest priority. With the *rule-prec* strategy one always builds a countermodel in case of non-provability because the invertible rules are applied before the non-invertible rules.

4 Results and comparisons

For a given sequent, STRIP can decide its provability and then build a proof or a countermodel (as a Kripke tree). The user can select the forest implementation (`lrmost` or `index-scope`) and the strategy (`first-leaf` or `rule-prec`). The system provides various statistics about the search like the number of rules applications (in order to evaluate the efficiency of strategies) or the number of performed operations (in order to determine if the forest implementation induces a huge overhead). We have compared the STRIP system with two provers, namely Porgi [8] and **ft** [7].
Porgi is a proof-or-refutation generator, written in SML, that is based on LJT with a strategy closed to the `rule-prec` strategy. It has a very simple lexical analyzer and thus we have only been able to test it with small formulae.
Some comparisons between Porgi and STRIP are given in figure 3 with formulae that are provable (p) or unprovable (u). The left part of the table presents execution times for both systems (T^x_y in seconds for 10^4 loops CPU time only). The

Dom.	Size	STRIP	ft	ft/STRIP
$\neg\neg\forall x(p(x) \vee \neg p(x))$				
7	257	1.5 ms	0.41 s	270
8	291	3.1 ms	2.64 s	850
9	325	6.4 ms	26.50 s	4100
10	359	12.8 ms	6 m 28 s	30000
21	733	31 s		
22	767	62 s		
23	801	1 m 50 s		
$\neg\neg(A \wedge B \to C)$ where $\begin{cases} A \equiv \exists x(p(x) \wedge \forall y(q(y) \to r(x,y)) \\ B \equiv \neg\exists x(q(x) \wedge \forall y(p(y) \to r(x,y))) \\ C \equiv \exists x(p(x) \wedge \neg q(x)) \end{cases}$				
4	1254	8.4 ms	130 ms	15
5	1870	90.0 ms	4.62 s	50
6	2610	1.53 s	4 m 19 s	170
7	3474	29.5 s	+ 1 h20 m	
8	4462	11 m 26 s		
$\neg\neg\boxed{\neg\exists x\forall y\exists z\, p(x,y,z) \leftrightarrow \forall x\exists y\forall z\, \neg p(x,y,z)}$				
2	495	0.42 ms	10 ms	24
3	1597	13.6 ms	+ 9 h	$+\ 10^6$
4	3743	1.07 s		
Pigeonhole x-y				
5-6	1788	2.23 s	0.33 s	0.14
6-7	2918	64 s	16 s	0.25
7-8	4446	32 m 44 s	19 m 27 s	0.60

Fig. 4. Comparison : STRIP vs **ft**

exponent 'lr' (resp. 'is') corresponds to the `lrmost` (resp. `index-scope`) implementation of the forest. The suffix 'fl' (resp 'rp') corresponds to the `first-leaf` (resp. `rule-prec`) strategy. The right part presents measures of the space size and search costs for STRIP proof-search. The expressions SS_y and SC_y^x respectively represent the number of logical rules applications and the total number of operations during the proof-search.
The STRIP system is always more efficient with the `rule-prec` strategy, which is almost the same as the one implemented in Porgi. Moreover, the difference is significant for the pigeon-hole formulae. We can also compare the strategies. In examples 20 and 22, we see that `first-leaf` is much less efficient than `rule-prec` because of the size of the proof-search space and thus in this case STRIP is slower than Porgi. The pigeon-hole examples illustrate the actual impact of sharing techniques and that some tasks, like forest management, are implemented much more efficiently in our system. From the SC statistics, i.e. measures of the search cost, we observe that the `index-scope` implementation of the forest is always better than the `lrmost` one but that the difference does not grow over factor 2, whichever strategy is chosen. However it is clear that efficient management of formulae becomes crucial on larger scales.

The **ft** system is suited for first-order logic but has a propositional subsystem for IPL. This system is not based on LJT and does not build countermodels but it is written in C like STRIP. Some comparisons between STRIP and **ft** are given in figure 4. In order to analyze the impact of sharing techniques on proof-search,

we have considered instances on finite domains of provable first-order formulae and also pigeonhole examples. The Dom. column represents the size of the domain or the number of pigeons and the Size column represents the size of the generated formulae[3]. We observe that, in general, STRIP is much faster than **ft** for the first kind of examples. But for the pigeonhole examples both systems are on par with a slight but decreasing advantage to **ft**. In fact they include very few implications ($\rightarrow$) and the structural sharing is such that left implication rules may cut down the problems by large amounts. In this case, the choice of a light strategy is important. We observe that with the `rule-prec` strategy, STRIP spends 85 % of its computation time looking for the active formulae. With some cyclic variants of `first-leaf`, we can cut-down this time to 65 % which is not optimum but nevertheless better, thus being close to **ft** and even better on the larger case (7-8). Anyway, we see that the greater the pigeon problem is, the better STRIP behaves, compared to **ft**.
Further work will be devoted to other tests and comparisons but regarding our positive results, our main goal is to apply or extend these implementation techniques (forest representation, structural sharing, forest implementation) to other substructural or intermediate logics [1] and thus to provide efficient provers that build proofs or countermodels for such logics.

References

1. A. Avellone, M. Ferrari, and P. Miglioli. Duplication-free tableau calculi and related cut-free sequent calculi for the interpolable propositional intermediate logics. *Logic Journal of the IGPL*, 7(4):447–480, 1999.
2. R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, 57:795–807, 1992.
3. D. Galmiche and D. Larchey-Wendling. Formulae-as-resources management for an intuitionistic theorem prover. In *5th Workshop on Logic, Language, Information and Computation, WoLLIC'98*, Sao Paulo, Brazil, July 1998.
4. D. Galmiche and D. Larchey-Wendling. Structural sharing and efficient proof-search in propositional intuitionistic logic. In *Asian Computing Science Conference, ASIAN'99, LNCS 1742*, pages 101–112, Phuket, Thailand, December 1999.
5. J. Hudelmaier. An O(n log n)-space decision procedure for intuitionistic propositional logic. *Journal of Logic and Computation*, 3(1):63–75, 1993.
6. 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.
7. D. Sahlin, T. Franzén, and S. Haridi. An intuitionistic predicate logic theorem prover. *Journal of Logic and Computation*, 2(5):619–656, 1993.
8. A. Stoughton. Porgi: a proof-or-refutation generator for intuitionistic propositional logic. In *CADE Workshop on Proof-search in Type-theoretic Languages*, pages 109–116, Rutgers University, New Brunswick, USA, 1996.

[3] The Size grows in n^p where p is the number of variables and n the size of the domain. For the pigeonhole, the size grows in n^3.