STATIC LEARNING FOR AN ADAPTATIVE THEOREM PROVER

C. BELLEANNEE J. NICOLAS
IRISA-INRIA, Campus de Beaulieu, 35042 Rennes cedex. France. Email : jnicolas@irisa.fr

Abstract

An adaptative theorem prover is a system able to modify its current set of inference rules in order to improve its performance on a specific domain. We address here the issue of the generation of inference rules, without considering the selection and deletion issues. We especially develop the treatment of repeating events within a proof. We specify a general representation for objects to be learned in this framework, that is macro-connectives and macro-inference-rules and show how they may be generated from the primitive set of inference rules. Our main contribution consists to show that a form of analytical, static learning, is possible in this domain.

Key-words Theorem proving. Macro-operators. Sequent calculus. Generalization to N.

1 Introduction

Our aim is to improve the performance of the theorem prover on a given domain, given some representative sample of theorems on this domain. A theorem prover may be characterized by its set of inference rules and its set of axioms. Our methodology is to produce automatically an auxiliary new set of macro-inference rules for the prover, rendering the specificity of the semantics of the domain.

It seems that there has been very little work addressing the issue of adapting a general automated prover on a specific domain (Cohen, 1987; O'Rorke, 1987; Pastre 1989). We present these works in the last section. At a first sight, theorem proving is just a particular case of problem solving and one may wonder why known learning methods in problem solving are not widely used in theorem proving. The mutual ignorance of each research field is only a

partial answer to this situation. In order to explain the specificity of theorem proving, we need a more accurate view of each domain.

One of the most common views of problem solving is to conceive it as a search in a state space. The transitions in the space are formalized as applications of operators. To solve a problem is to find a path between an initial state and a final goal state. If we want to describe theorem proving within this framework, states are sets of formulas, the goal state being the theorem to be proven (or its negation if we are looking for unsatisfiability). Operators are rules of inference.

Learning in problem solving consists of building a new, more abstract state space. That is, learning either produces clusters of states, or/and sequences of operators (paths in the space). The first case corresponds to the learning of preconditions of operator applications. The second case describes macro-operator learning.

What makes the learning of preconditions effective in problem solving is that each step is meaningful. Each operator represents a transition between two worlds with a clear semantics conveyed by the action. In theorem proving, one faces the issue of transformations that are too atomic to represent significative steps and allows the production of uninteresting intermediate states (sets of formulas). Unlike problem solvers where operators are problem-dependent, theorem provers need a complete and general set of inference rules in order to draw their conclusions, which does not capture the semantic of the domain.

The definition of a strategy is the main problem in theorem proving. We reduce it to the learning of useful sequences of elementary steps, that is, learning of macro-inference rules. The principle is to produce a redundant set of inference rules, making up for the increase in the branching factor during the choice of an inference rule step, with a sensible decrease of the number of steps needed to achieve a proof. The formation of macro-rules has been a subject of interest since the development of problem solvers and planning systems. An early and famous study is the MACROPS system of Fikes, Hart and Nilsson (1972). More recent studies include (Porter, 1984; Korf 1983, 1985; Minton 1988; Iba 1989). The specificity of the problem for theorem provers lies in the fact that elementary operators have a very poor semantics, independant of the domain. However, it is possible to build useful macros if we consider some proof traces of known theorems. This dynamic form of learning has been previously explored. However, it seems that **static** learning, a form of learning studied by Korf (1983,1985) in the framework of problem solving, has not lead to any work in theorem proving. Static learning produce macro-operators while analysing operators themselves, without considering any trace of resolution. Since operators are very general in theorem proving, one may believe that such a technique is of no help for the problem at hand. This paper aims to show that some kind of static learning is nontheless possible. We first present a quick view of our prover.

2 Automated Theorem Proving

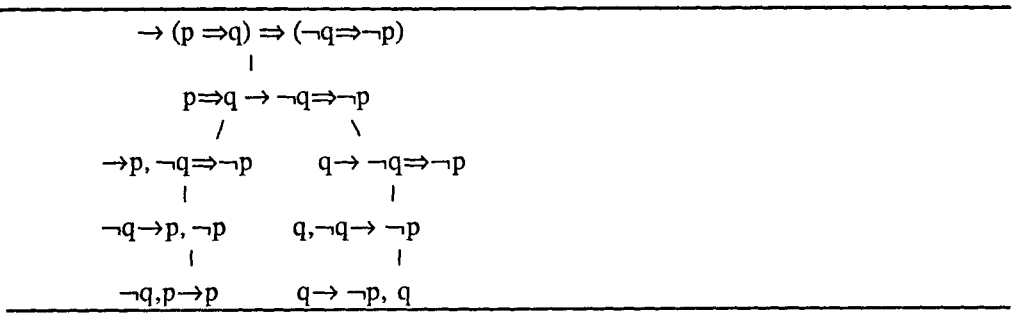
Automated deduction methods may be split into two main streams. Resolution style methods deal with formulas in a standard form by means of a single inference rule. Natural like methods accept formulas in their original form but require at least as many inference rules as connectives in the original language.

This paper deals with the second kind of methods, since we claim that keeping formulas in a more structured form allows one to reason at different levels of abstraction, which is a fundamental requirement for learning methods.

Amongst natural-like methods, we have chosen the system G, a sequent calculus defined by Gentzen (Gallier, 1986), which is well suited to mechanisation because of the convergence of its set of inference rules (due to the subformula property of the system). A sequent (SA,SB) is a pair of possibly empty sets of formulas, noted $SA \rightarrow SB$ or $A_1, ...,A_m \rightarrow B_1, ...,B_n$ if $SA = \{A_1, ...,A_m\}$ and $SB = \{B_1, ...,B_n\}$. The semantics associated to this notation is that $A_1, ...,A_m \rightarrow B_1, ...,B_n$ is true iff the formula $A_1 \wedge ... \wedge A_m \Rightarrow B_1 \vee ... \vee B_n$ is true. The inference rules of the system G directly reflect the semantics of the logical connectives. There are two inference rules for each connective, corresponding respectively to the treatment of an occurence of the connective in the left and in the right part of the sequent. For example, the following rules govern the occurences of the *and* connective:

$$\frac{A_1, A_2, SA \rightarrow SB}{A_1 \wedge A_2, SA \rightarrow SB} \qquad \frac{SA \rightarrow SB, B_1 \quad SA \rightarrow SB, B_2}{SA \rightarrow SB, B_1 \wedge B_2}$$

In the framework of the system G, a deduction of a sequent S consists in splitting up the semantics of S, by means of the inference rules. As an example, here is a deduction tree for the sequent " $\rightarrow (p \Rightarrow q) \Rightarrow (\neg q \Rightarrow \neg p)$" (It is a proof tree for the formula $F = (p \Rightarrow q) \Rightarrow (\neg q \Rightarrow \neg p)$)



3 When to build a new macro

Four issues have to be addressed when considering macros :

a) When to build a new macro. b) How to build a new macro.

c) When to cancel an old macro. d) When to select a given macro.

The focus of this paper is on the first two points. Point a) raises the issue of defining the concept of *interestingness* of a sequence of rules or operators. It is the subject of this section. Point b) addresses the issue of *composing* a sequence of rules or operators and is developed in section 4.

The generation of macros requires the prior definition of formal criteria leading to the specification of the concept of an interesting sequence of rules. This work proposes to tackle with domains where repeated applications of a same sequence of rules of inference occur during the proof of theorems. In fact, this covers three kinds of situations. First, a given theorem may have a regular structure, with repeated sequences of connectives, leading to a repeated application of some sequence of rules in the proof tree. Next, the proof of the theorem itself may be structured, leading to the natural elaboration of lemmas whose proofs may be reused for the construction of the global proof. And last, one may want to prove several instantiations of a generic formula, thus using the same sequence of rules for different theorems sharing a same structure. In this last case, the prover becomes able to solve complex theorems (problems), training itself on a set of simpler but complex similar theorems of growing complexity, thus exhibiting an ability to learn from experience. In each case, macro-rules may be detected upon an empirical basis : the observation of regularities in the data.

The problem is referred to as the "generalization to N" problem since the work of Shavlick and Dejong (1987), and is a subject of growing interest in the Machine Learning community (see, e.g., (Cheng &Carbonell, 1986; Cohen, 1987,1988; Prieditis, 1986)). It involves the extension of a given procedure accepting a fixed number of arguments, in order to obtain a new looping procedure, able to handle an arbitrary finite number of arguments. The aim of this transformation is to keep the computation inside reasonable bounds of complexity with the growing number of arguments.

We now define more precisely this notion of regularity in the framework of theorem proving. We introduce for this purpose the basic concept of homogeneity, which roughly corresponds to the invariance with permutation on the components of a logical object. The motivation for this choice comes from the observation of classical "hard" logical puzzles like the placing of queens on a board, the colouring of a complete graph (Greenwood and Gleason 1955) or the Schurr lemma (Schur 1916). They all include homogeneous formulas, and the

difficulty of their proofs stems from the incapability of general theorem provers to handle such a global property. As a consequence, the size of their proofs depends exponentially on the number of "objects" in the problem (number of queens, of nodes...). We developed a semantical and a syntaxical version of the homogeneity criterion. The detection of syntactical homogeneity leads to a form of static learning, in the sense that no proof is needed in order to build macros.

3.1 Basic definitions

Definition : A *macro-operator of arity n* M is a λabstraction of a logical object w.r.t. n variables $F_1,...F_n$, i.e. of the form $\lambda F_1,...F_n\ M(F_1,...F_n)$

This definition is primarily designed for formulas, where operators are connectives and we will focus on formulas in the rest of the paper. But the definition may be easily instantiated to handle sets of formulas, by means of a special connective ",". Finally, we represent a proof in the same way, rules of inference being treated as connectives of arity 2.

Examples: Id = λ F.F is the identity macro operator, of arity 1.

Equiv = λ A,B.(A⇒B)∧(B⇒A) is a macro-connective of arity 2 corresponding to the usual equivalence relation.

Definition : A formula F is *semantically homogeneous with respect to the components* S_C if S_C is a set of sub-formulas occurring in F such that F is invariant for every permutation on these objects (i.e every permutation on S_C leads to logically equivalent formulas). The cardinality of S_C is the *arity* of homogeneity.

(Macro-)connectors themselves may be viewed as generic formulas. We say that a connector of arity 2 is *extendable* iff it is associative and homogeneous of arity 2, that is iff it is associative and commutative.

Examples:

{p∨q∨r, ¬p∨¬q, ¬p∨¬r, ¬q∨¬r} is a set of formulas homogeneous of arity 3 with respect to the components p,q,r.

The ∨ connective is extendable.

We now provide a syntactical, stronger notion of homogeneous logical objects. This provides a **practical** definition of homogeneity. We first define the notion of homogeneous block, which is a syntactical characterization of the elementary (smallest) homogeneous objects.

Definition : A formula F is a *homogeneous block with respect to a set of components* S_C, denoted <C,M> if there exists a redex of F= (F_A F_I), such that
a) the only operator of F_A is the extendable connective C, the *link* of the block
b) each formula $f_i \in F_I$ is an instance of the macro-operator M, the *pattern* of the block
c) the set $\{f_i\}$ is invariant (unaltered) for every permutation of its components S_C.

Proposition1: If there exists a redex (FA FI) representing a formula F such that the formulas f_i of FI are homogeneous blocks with respect to a set of components S_C, then F is semantically homogeneous with respect to S_C. We say that F is a *connection of the blocks FI*, with the *block structure* FA.

Examples:

F1 = $p \vee q \vee r = ((\lambda x,y,z\,.\,x \vee y \vee z)\ p\ q\ r) = <\vee, Id>$ is a homogeneous block with respect to {p,q,r}.The extendable connective $\vee$ is the link and $\lambda x.\ x = Id$ is the pattern of the block.

F2 = $(\neg p \vee \neg q) \wedge (\neg p \vee \neg r) \wedge (\neg q \vee \neg r) = ((\lambda x,y,z\,.\,x \wedge y \wedge z)\ \neg p \vee \neg q\ \neg p \vee \neg r\ \neg q \vee \neg r) = <\wedge, \lambda x \lambda y.\neg x \vee \neg y>$ is a homogeneous block with respect to {p,q,r}.

F3 = $(\neg p \vee \neg q) \wedge (\neg p \vee \neg r)$ is not homogeneous with respect to {p,q,r}. Indeed, given the permutation $\{p \leftarrow q,\ q \leftarrow p\}$, the formula becomes $(\neg p \vee \neg q) \wedge (\neg q \vee \neg r)$ which is clearly not equivalent.

F = $(p \vee q \vee r) \wedge (\neg p \vee \neg q) \wedge (\neg p \vee \neg r) \wedge (\neg q \vee \neg r)$ is not a homogeneous block with respect to {p,q,r}, but it is a homogeneous formula with respect to {p,q,r}. It is a connection of the blocks {F1,F2} whose block structure is $\lambda x\ \lambda y.\ x \wedge y$.

The converse of the proposition is clearly false. For example, $(p \Rightarrow q) \wedge (p \vee \neg q)$ is homogeneous with respect to {p,q} but is not decomposable in homogeneous blocks. We assume that syntactical regularities are the only relevant regularities at the formula level and that the detection of equivalences occurs at the proof level. Thus, we obtain a practical, efficient way to test for homogeneity. Moreover, we have proved in (Belleannée, 1991) the following result, showing the expression power of the syntactical criterion:

Proposition2: for every semantically homogeneous formula of the propositional calculus, there exists at least one logically equivalent formula syntactically homogeneous.

The next step consists in generalizing the number of parameters a same macro-operator is able to handle, leading to the construction of an homogeneous concept.

3.2 Macro-operators of variable arity

Definition : A *macro-operator of variable arity* M(L) is an abstraction of variable length such that L denotes the set of variables occurring in the macro.

The expression of these macro-operators requires the definition of a language of operators handling sets of logical objects. We have designed for this purpose a general *set operator schema*. In order to improve the readability of paper, we only present a restriction of our schema, sufficient for the current description. In this schema, "logical object" means formula or sequent:

$\overset{\infty}{M}$((Macro-)Operator, Structure_of_selections, Set_of_logical_objects), where

$\overset{\infty}{M}(M_d,Struct(V_1...,V_d),L) \equiv \{M_d(\sigma V_1..., \sigma V_d) / \sigma$ instanciates $Struct(V_1...,V_d)$ in L$\}$

A structure with variable arity is based on a notion of repetition of patterns.
In our schema, in terms of control structures the macro is a for loop producing instances of the macro-operator M_d. $Struct(V_1...,V_d)$ may be viewed as the condition part of a while loop on the set of primitive components L.
In the schema:
-The macro-operator M_d is either a simple macro operator or a primitive macro-operator of variable arity (that is, the extension of an extendable macro of arity 2 to greater arities by means of a repeated application of it). For instance, we define $\overset{\infty}{\wedge}(L)$ to be the repeated application of $\wedge$ on the elements of L.
-The selector $Struct(V_1,....,V_d)$ designates the set of variables $\{V_1,....,V_d\}$with a particular structure (i.e. a partial ordering on the variables).

Based on this schema, we introduce simplified notations for some default values of set operator arguments such as: $\overset{\infty}{M}(M_d,L) = \overset{\infty}{M}(M_d,Var(L),L)$, where Var(L) returns a pattern of different variables the length of which is the cardinal of L, and we define primitive operators such as: $AND(S,L) = \overset{\infty}{M}(\wedge,S,L)$).

Example : At-least-2(L) = $\overset{\infty}{M}(\vee,\overset{\infty}{M}(\wedge, <A,B>, L)) = \overset{\infty}{\vee}$ (AND (<A,B>, L)) is a macro-connective of variable arity.
At-least-2({p,q,r}) $\equiv \{(p\wedge q) \vee (p\wedge r) \vee (q\wedge r)\}$.

3.3 From homogeneous formula to homogeneous concepts

The defined language allows the user to introduce his own macro-connectives in the theorem prover. However, some macro-connectives of variable arity may be automatically derived from simple ones. So, the homogeneity concept aims at specifying the simple macro-operators that may be extended to handle an arbitrary number of arguments. We now define this mapping.

The process consists in generalizing to N the number of arguments of the macro, *preserving the block structure* of this macro. Indeed, our fundamental claim is that preserving this block structure preserves the underlying semantical interesting features of the operator and hence allows it to be efficiently treated.

Formally, let F be a fixed arity homogeneous logical object for a set of components S, and let P = (FA FI) be a redex of F where $f_i \in F_I$ are homogeneous blocks. The corresponding macro-operator of variable arity is P except that

each block $f_i = <C,M_d>$ is replaced with $\overset{\infty}{f}_i = \overset{\infty}{M}(C,\overset{\infty}{M}(\mathbf{M_d},\mathbf{structure(M_d)},\ \mathbf{L}))$,

where structure is a recursive function whose result is a partially ordered set of variables depending on the primitive connectives of M_d.

Example: Let F be the homogeneous formula F of the previous section:

$F = (p \vee q \vee r) \wedge (\neg p \vee \neg q) \wedge (\neg p \vee \neg r) \wedge (\neg q \vee \neg r)$, whose block structure is $F = F1 \wedge F2$, with $S=\{p,q,r\}$, $F1 = <\vee, Id>$ and $F2 = <\wedge, \lambda x \lambda y.\neg x \vee \neg y>$.

The corresponding macro-connective of variable arity is

$\overset{\infty}{F}(L) = \overset{\infty}{F}1(L) \wedge \overset{\infty}{F}2(L)$ with

$\overset{\infty}{F}1(L) = \overset{\infty}{M}(\vee, \overset{\infty}{M}(Id,<x>,L)) = \overset{\infty}{M}(\vee,L) = \overset{\infty}{\vee}(L)$.

$\overset{\infty}{F}2(L) = \overset{\infty}{M}(\wedge,\overset{\infty}{M}(\lambda x \lambda x.\neg x \vee \neg y,<P,Q>,L)) = \overset{\infty}{\wedge}(\{\lambda x \lambda x.\neg x \vee \neg y,\ <P,Q>,L)\})$.

That is $\overset{\infty}{F}(L) = \overset{\infty}{\vee}(L) \wedge \overset{\infty}{\wedge}(\{\lambda x \lambda x.\neg x \vee \neg y,\ <P,Q>,\ L)\})$.

For instance, $\overset{\infty}{F}(\{a,b,c,d\}) = (a \vee b \vee c \vee d) \wedge (\neg a \vee \neg b) \wedge (\neg a \vee \neg c) \wedge (\neg a \vee \neg d) \wedge (\neg b \vee \neg c) \wedge (\neg b \vee \neg d) \wedge (\neg c \vee \neg d)$.

From a semantic point of view, the instance F expresses the meaning 1-among-3, and the generalized formula $\overset{\infty}{F}$ represents the concept 1-among-N. Actually, the generalization preserving the block structure also preserves the homogeneity property.

We do not claim that this mapping is the only interesting one. It has been considered for its simplicity, and in our opinion it exhibits the concept the more naturally induced from the instance, within the bias of homogeneity. But it must be clear that other transformations have to

be considered in order to fully benefit from the expressiveness of the language of macros of variable arity.

4 How to build a new macro-rule

This section deals with the integration of the new macro-connectives in the framework of the system G (see section 2), according to a *partial evaluation principle*. The problem is the following: given a macro-connective, we would like the system to be able to take it into account directly through associated macro-rules. For each macro-connective, two inference rules may be generated, one operating on a macro occurring in the left part of a sequent and the other one in the right part. A macro-rule is produced by synthesizing the deduction tree of the macro-connective. The method used to process the inference rules assigned to a macro-connective MC may be featured by the following algorithm.

---

func construct-the-inference-rules (MC):

construct-one-rule (MC $\rightarrow$) construct-one-rule ($\rightarrow$ MC)

func construct-one-rule (Sequent):

Deduction-tree := expand-tree (Sequent)

Leaves:= extract-the-leaves (Deduction-tree)

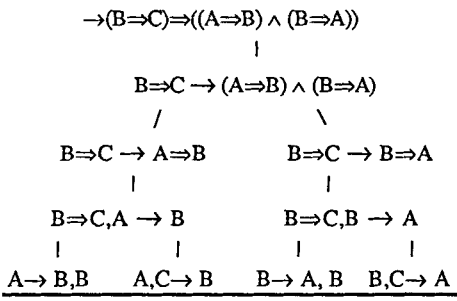
$$\frac{\text{Leaves}}{\text{Sequent}}$$

We precise now the construction of the rules for the simple and the variable arity cases.

For a macro-connective MC *with fixed arity*, the function "expand-tree" processes the deduction tree associated to the sequent MC$\rightarrow$ (or $\rightarrow$ MC) applying inference rules of the system G to the sequent. The development is achieved when all the operators occurring in the partition of MC have been completely expanded, that is, when the resulting sequents contain only logical objects of the initial partition (those sequents are the *leaves* of the tree). Then, the second procedure "extract-the-leaves" simplify the proof, removing redundant leaves such as axioms and subsumed leaves. The resulting macro-rule consists of the optimized leaves as its premise, and in the macro-connective as its conclusion.

Example: Given the macro-connective MC defined by MC(A,B,C) = (B$\Rightarrow$C)$\Rightarrow$((A$\Rightarrow$B) $\wedge$ (B$\Rightarrow$A)), the following figure illustrates the processing of the right macro-rule for MC:

deduction tree resulting from expand-tree (→MC(A,B,C)):



right macro-rule for MC resulting from construct-1-rule(→MC(A,B,C)):

$$\frac{A \rightarrow B \quad B,C \rightarrow A}{\rightarrow MC(A,B,C)}$$

In order to derive the deduction tree of a macro-connective *with variable arity*, according to the system G, the function "expand-tree" needs extended derivation rules dealing with the variable arities. For illustration purpose, we just present one of them, treating the occurence of a variable arity AND in the left part of a sequent (for more information see (Belleannée, 1991)):

$$\frac{\overset{\infty}{M}\,(\,[\,SA \rightarrow SB, X\,],\,<X>,\,L)}{SA \rightarrow SB,\,\overset{\infty}{\wedge}(L)}$$

5 An example

We show an example of a homogeneous formula handled by the system. The first step consists in detecting the simple homogeneous formula and then learning the corresponding macro-rule with variable arity. The second step illustrates the use of the learned structure.

5.1 Learning step

- Given the formula F= (a∧b) ∨ (¬a∧¬b), the system detects whether it is homogeneous with respect to its components {a,b}. The resulting block structure is $F = F_1 \vee F_2$

with F_1= (a∧b) = <∧, Id> and F_2= (¬a∧¬b) = <∧, λx .¬x>

- Then, the application of the generalization to N algorithm produces the homogeneous formula with variable arity $\overset{\infty}{F}$: $\overset{\infty}{F}(L) = \overset{\infty}{F_1}(L) \vee \overset{\infty}{F_2}(L)$

with $\overset{\infty}{F}_1(L) = \overset{\infty}{M}(\wedge, \overset{\infty}{M}(Id,<A>,L)) = \overset{\infty}{M}(\wedge,L) = \overset{\infty}{\wedge}(L)$

and $\overset{\infty}{F}_2(L) = \overset{\infty}{M}(\wedge, \overset{\infty}{M}(\lambda x .\neg x ,<A>,L)) = \overset{\infty}{\wedge}(\{ \lambda x .\neg x , <A>, L \})$

-Both macro-rules associated to $\overset{\infty}{F}$ are established applying on $\overset{\infty}{F}$ variable arity rewriting rules (see section 4). Notation: PROD is a product on sets of sequents.

a- Left rule:

derivation tree:

$\overset{\infty}{\wedge}(L) \vee \overset{\infty}{\wedge}(\{\lambda x .\neg x ,<A>,L\}) \rightarrow$

/ \

$\overset{\infty}{\wedge}(L) \rightarrow$ $\overset{\infty}{\wedge}(\{\lambda x .\neg x ,<A>,L\}) \rightarrow$

| |

PROD ($\overset{\infty}{M}(\lambda x .[x \rightarrow], <A>, L)$) PROD ($\overset{\infty}{M}(\lambda x .[\neg x \rightarrow], <A>, L)$)

| |

$L \rightarrow$ PROD ($\overset{\infty}{M}(\lambda x .[\rightarrow x], <A>, L)$)

|

$\rightarrow L$

synthesized rule:

$$\frac{L \rightarrow \qquad \rightarrow L}{\overset{\infty}{F} \rightarrow}$$

b- Right rule:

derivation tree:

$\rightarrow \overset{\infty}{\wedge}(L) \vee \overset{\infty}{\wedge}(\{\lambda x .\neg x ,<A>,L\})$

|

PROD ($\{\rightarrow \overset{\infty}{\wedge}(L)$, $\rightarrow \overset{\infty}{\wedge}(\{\lambda x .\neg x , <A>, L \})$

|

PROD ($\{ \overset{\infty}{M}(\lambda x .[\rightarrow x], <A>, L)$, $\overset{\infty}{M}(\lambda x .[\rightarrow \neg x], <A>, L) \}$)

|

PROD ($\{ \overset{\infty}{M}(\lambda x .[\rightarrow x], <A>, L)$, $\overset{\infty}{M}(\lambda x .[x \rightarrow], <A>, L) \}$)

|

$\overset{\infty}{M}(\lambda x \lambda y. x \rightarrow y, <A_1;A_2> L)$

synthesized rule:

$$\frac{\overset{\infty}{M}(\lambda x \lambda y. x \rightarrow y, <A1;A2>,L)}{\rightarrow \overset{\infty}{F}}$$

5.2 Using the learned rules

From now on, when the system detects that a formula F is an instance of $\overset{\infty}{F}$, it directly derives F by means of both previous rules. For instance, the formula $F_2 = (p \wedge q \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r)$ matches the formula $\overset{\infty}{F}$, with the instanciation L={p,q,r}, so the immediate derivation of "$F_2 \rightarrow$" using the macro-rule "$\overset{\infty}{F} \rightarrow$" is (while an indirect derivation of "$F_2 \rightarrow$" would have produced a derivation tree with 10 nodes and 7 levels):

$$(p \wedge q \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r) \rightarrow$$

$$/ \qquad \backslash$$

$$p,q,r \rightarrow \qquad \rightarrow p,q,r$$

6 Related Work

Amongst the systems addressing the issue of adapting a general automated theorem prover on a particular domain, one can distinguish two main approaches.

The system Muscadet (Pastre, 1989) is a representative of the first approach. In this system, the learning component may be classified as performing knowledge acquisition. Indeed, learning proceeds by transforming some declarative knowledge into a procedural, deductive one.

Muscadet is a general theorem prover with an expert system architecture. It is dedicated to mathematical fields requiring a lot of knowledge and know-how, like set theory, relations, topology... Specific knowledge is given to the system as facts and rules. Then some meta-rules automatically translate mathematical definitions into inference rules.

The second approach is related to explanation based learning in order to build an operational knowledge guiding deductions.

The system realized by O'Rorke consists in applying an EBL learning system to propositional calculus problems from Principia Mathematica (O'Rorke, 1987). The system generalizes the already proven theorems in order to forget the extraneous details and to remember only the main features of the specific theorems.

The generalized theorems are then stored to be reused for future proofs. For instance, the theorem $\neg (p \vee q) \Rightarrow (p \Rightarrow q)$ gives rise to the theorem $\neg (A \vee B) \Rightarrow (A \Rightarrow D)$.

In this system, the learning process consists in learning particular lemmas, characteristic to the domain, in order to augment the set of basic axioms of the theorem prover.

The system ADEPT (Acquisition of DEterministic Proof Tactic) (Cohen, 1987, 1988) performs a more sophisticated kind of EBL. It acquires some control knowledge, that is, search strategies for a theorem prover on a particular domain, given instances of "good proofs".

The method consists in analysing the inputs proofs in order to find which inference rule was used in which situation. This learned knowledge is represented in a finite state automaton, and is used to guide the choice of the inference rule to employ in future proofs, when similar situations occur. Learning belongs to generalization to N methods, which allows recognition and generalization of looping constructs in the proof instances.

Our method shares with all these systems the common concern to "operationalize" some knowledge about the kind of proofs that are requested from the prover. ADEPT is the nearest system from ours. But, we are working on the structure of the theorems themselves rather than the proofs. Both approaches are not concurrent and the full treatment of homogeneity requires a work at the proof level. However, homogeneity is a *global* criterion, and requires to work on the whole proof. The regularity criterion in ADEPT (two rules of inference belongs to a same class if they are immediately followed by identical rules) is more local and thus, potentially captures more and shortest macros.

7 Conclusion

This paper proposes a framework enabling one to introduce learning abilities in an automated theorem prover. It is founded on the detection of regular structures within formulas and proofs. We provide a representation, that is a language and operators to deal with such structures

We have not yet addressed the important issue of determinating when to cancel an old macro. Iba (1989) has developed a general framework to describe system managing sets of macro-operators. In this view, the set of macro-operators is pruned by two types of filters. Static filters are applied at the creation of a new macro to decide on analytical criteria which macro may be kept. Dynamic filters are applied after an execution requiring some macro-operators to decide on empirical criteria which macro may be kept.

In the framework of theorem proving, according to this work, we plan to experiment with the following filters:

static filters: a) Keep macros of polynomial complexity (analytical measure). b) Check for each new macro if it is not redundant with some old ones. Remove the most specific.

dynamic filters: c) Keep macros of polynomial complexity (empirical measure). d) Retain macros in an ordered list of bounded size. Remove the least frequently used.

References

Belleannée C. "Improving deduction in a sequent calculus". Proc of the 8th biennal conference of the CSCSI", pp 220-226, Ottawa, may 1990.

Belleannée C. "Vers un démonstrateur adaptatif". Thèse de l'Université de RennesI, Jan 1991.

Cheng P. & Carbonell J. "The FERMI system: Inducing Iterative Macro-operators from Experience". Proc. of AAAI, 1986 .

Cohen W. "A Technique for Generalizing Number in Explanation Based Learning". ML TR 19, Rutgers University , Sept 1987.

Cohen W. "Generalising Number and Learning from Multiple Examples in Explanation Based Learning". Proc. of 5th ICML, pp256-269, Morgan Kaufmann, Los Altos, Calif. 1988.

Fikes R., Hart P. & Nilsson N. "Learning and Executing Generalized Robot Plans ". A.I. 3, pp 251-288, 1972.

Gallier J.H. "Logic for Computer Science: Foundations of Automatic Theorem Proving". Harper & Rown, New York, 1986.

Greenwood R.E. & Gleason A.M. "Combinatorial relations and chromatic graphs" Combinatorial Journal 7, 1955

Iba G. "A Heuristic Approach to the Discovery of Macro-operators". Machine Learning 3, pp 285-317, 1989.

Korf R. "Learning to Solve Problems by searching for macro-operators". PhD Thesis Carnegie Mellon University, 1983.

Korf R. "Macro-operators : A Weak Method for Learning". A I 26, pp35-77, 1985.

Minton S. "Learning Effective Search Control Knowledge: an Explanation Based Approach". PhD Thesis Carnegie Mellon University , 1988.

O'Rorke P. "LT revisited : Experimental results of applying explanation-based learning to the logic of principia mathematica". 4th IWML Irvine,1987.

Pastre.D "Muscadet : An Automatic Theorem Proving System Using Knowledge and Meta-knowledge in Mathematics". Artificial Intelligence, vol 38, 1989 pp 257-318.

Porter.B "Learning Problem Solving". PhD Thesis University of California, Irvine 1984.

Prieditis A. "Discovery of Algorithms from Weak Methods". Proc. of International Meeting on Advances in Learning, Les Arcs France, 1986.

Schur I. "Über die kongruenz $x^m+y^m=z^m$ mod p" Jber Deutsch Verein 25, pp114-116. 1916

Shavlik J. & Dejong G. "An Explanation-Based Approach to Generalising Number". Proc. of IJCAI-87, Milan, Italy, pp236-238, 1987.