Mathematical Theorem Proving, from MUSCADET0 to MUSCADET4*, Why and How

Dominique Pastre
University Paris Descartes
pastre@math-info.univ-paris5.fr

1st International Workshop about Sets and Tools (SETS 2014) Affiliated to ABZ 2014

June 3, 2014

Abstract : This paper presents the ideas and the choices which have been made throughout the development of the MUSCADET theorem prover. We will first see the principles and main ideas which lead to a first prover in the context of the time, influenced by a famous paper by Woody Bledsoe. This program used natural methods and was applied to set theory. It was then rewritten as a knowledge based-system where an inference engine applied rules, given or automatically built by metarules which expressed general or specific mathematical knowledge. It has been applied to some difficult problems. In order to allow more flexibility for expressing knowledge, it has been rewritten in Prolog, allowing the knowledge to be more or less declarative or procedural. To work with the TPTP Library the system had to work without knowing anything about mathematics except predicate calculus. All mathematical concepts had to be defined with mathematical statements, and the membership relation handled as any other binary relation. To avoid translating knowledge to TPTP syntax, TPTP syntax has been used (this unfortunately forbade the use of some abbreviations mathematicians are comfortable with). Last but not least, the relevant trace has been extracted to give a proof easily read by anyone, except in the case of failure, when all steps may be displayed to understand (manually) the reasons for the failure. MUSCADET has participated to CASC competitions. The results show its complementarity with regard to resolution-based provers.

1 Introduction

The MUSCADET theorem prover is a knowledge-based system. Based on "natural deduction", following the terminology of Bledsoe [1, 2], it uses methods which look like those used by humans, contrary to most provers.

This famous paper by W.Bledsoe [1] has mostly influenced the implementation of the first version of MUSCADET. The emergence of expert systems and the work on knowledge and metaknowledge made in J.Pitrat's team is the second event which has the most influenced the development of MUSCADET.

We will first see the main methods of DATTE which are still the most important strategies of the current version of MUSCADET. Then we will see why and how MUSCADET has been rewritten or improved in its successive versions.

As a conclusion, we will see why MUSCADET is an efficient system in a number of circumstances and how theorem proving with MUSCADET could progress.

*MUSCADET (http://www.math-info.univ-paris5.fr/~pastre/muscadet/muscadet.html) has been developed at Laforia (University Pierre et Marie Curie, then in Crip5/Lipade (University Paris Descartes)

2 A bit of history

2.1 First provers

I like to remind that the first Artificial Intelligence program was a theorem prover, the "Logic Theorist" [7] by Newell, Shaw and Simon in 1957. The objective was a program to play chess, but they found that it was necessary to first write this Logic Theorist theorem prover. Then this lead to the chess playing program NSS [10] in 1958 which had the same structure, and to GPS (General Problem Solver) [11] in 1961 with the notions of goals and subgoals.

Then, among the pioneers, I will quickly cite provers for geometry by Gelertner [4], for propositional and predicate calculus by Wang [25] and for integration by Slagle [21]. These programs were promising first steps. They used methods which look like human methods. Sequents were used in [25].

2.2 Revolution and counter-revolution Resolution Principle vs Natural Deduction

With the "Resolution Principle" in 1965, Robinson [20] modified research directions and those first experiments were somewhat forgotten. The Resolution Principle works on sets of clauses. Clauses are disjunctions of literals. The proof of a theorem T consists in finding a contradiction of a set of clauses (that is a derivation of the empty clause) obtained from the negation of T. The conjunction of these clauses is equivalent to the prenex skolemised conjunctive normal form (CNF) of $\neg T$.
Its advantages are the followings :

- it uses only one simple deduction rule which generates new clauses from preceding clauses;
- it is complete for refutation in predicate calculus.

Its disadvantages are the followings :

- the growth of the number of generated clauses is exponential, as well as the necessary time to obtain a proof;
- the sets of clauses and the proof (derivation) are not easily readable by humans.

On the other hand, in 1971, Bledsoe [1] reintroduced natural methods. He showed that proofs obtained by "natural deduction" [1] were shorter than those obtained by Resolution. His program PROVER works on first-order formulas. It puts emphasis on the notions of hypotheses and conclusions. It uses splitting and rewriting general rules general (SPLIT) or specific for set theory (REDUCE). Here are some examples of rules

	$A \wedge B$	is divided into	two sub-theorems A et B
	$A \leftrightarrow B$	is changed to	$(A \rightarrow B) \wedge (B \rightarrow A)$
	$\forall x P(x)$	"	$P(x)$
	$p \rightarrow (A \rightarrow B)$	"	$p \wedge A \rightarrow B$
SPLIT	$p \rightarrow A \wedge B$	"	$(p \rightarrow A) \wedge (p \rightarrow B)$
	$p \vee q \rightarrow A$	"	$(p \rightarrow A) \wedge (q \rightarrow A)$
	$A \rightarrow \forall x P(x)$	"	$A \rightarrow P(y)$ où y is a new variable
	$\exists x P(x) \rightarrow D$	"	$P(y) \rightarrow D$ "
	$x = y \rightarrow P(x, y)$	"	$P(y, y)$ if y is a variable

Moreover, a subroutine OR-OUT tries to convert formulas of type $H \rightarrow C$ to the types $p \vee q \rightarrow A$ or $p \rightarrow A \wedge B$.

For example, it converts $A \wedge (B \vee C) \rightarrow D$ to $(A \wedge B) \vee (A \wedge C) \rightarrow D$ and $(A \rightarrow B) \rightarrow C)$ to $(\neg A \vee B) \rightarrow C$ which are then split.

Here are some examples of rules for set theory

[1] For Bledsoe, the meaning of the terms "natural deduction" differs from what it is in Gentzen's rules; it is both larger and less formal

REDUCE	$s \in A \cap B$	is changed to	$s \in A \wedge s \in B$
	$s \in A \cup B$	"	$s \in A \vee s \in B$
	$s \in \mathcal{P}(A)$	"	$s \subset A$
	$C \subset A \cap B$	"	$C \subset A \wedge C \subset B$
	$A \cup B \subset C$	"	$A \subset C \wedge B \subset C$
and some other general rules	$\neg(A \wedge B)$	is changed to	$\neg A \vee \neg B$
	$\neg(A \vee B)$	"	$\neg A \wedge \neg B$
	$\neg\ \forall x A$	"	$\exists x \neg A$
	$\neg\ \exists x A$	"	$\forall x \neg A$

Other major parts of the program are its heuristic handling of equality and equivalence and the use of induction. If a subtheorem is not proved, Resolution is then called for this subtheorem.

An example of a proof

Theorem to be proved $\mathcal{P}(A) \cap \mathcal{P}(B) = \mathcal{P}(A \cap B)$

Proof : Since the main connective is "=", it is changed to

$$\mathcal{P}(A) \cap \mathcal{P}(B) \subset \mathcal{P}(A \cap B) \ \wedge \ \mathcal{P}(A \cap B) \subset \mathcal{P}(A) \cap \mathcal{P}(B)$$

Splitting

(1)	(2)
$\mathcal{P}(A) \cap \mathcal{P}(B) \subset \mathcal{P}(A \cap B)$	$\mathcal{P}(A \cap B) \subset \mathcal{P}(A) \cap \mathcal{P}(B)$
$t \in \mathcal{P}(A) \cap \mathcal{P}(B) \rightarrow t \in \mathcal{P}(A \cap B)$	
$t \in \mathcal{P}(A) \wedge t \in \mathcal{P}(B) \rightarrow t \in \mathcal{P}(A \cap B)$	proof
$t \subset A \wedge t \subset B \rightarrow t \subset A \cap B$	similar
$t \subset A \wedge t \subset B \rightarrow t \subset A \wedge t \subset B$	
true because each part of the conclusion is contained in the hypotheses	*true*

A proof obtained by Resolution is long and tedious (11 clauses at the beginning).

3 A first program based on "natural deduction" (DATTE, alias MUSCADET0)

We will see in this section the main principles and ideas which lead to DATTE, a first prover written in Fortran (!). They are still the same in MUSCADET. DATTE may be seen as the version 0 of MUSCADET.

DATTE used "natural" methods, as Bledsoe's program [1], and some other still more efficient human-like methods. The representations were really ugly from a modern point of view and the programming style obsolete. For these reasons, I will not give details about them but only describe the methods and the results in an informal manner.

Rather than rewriting rules, conditional actions are used. Some of them are general for logics and general mathematics, other ones are automatically built from the definitions.

3.1 General rules

Here are some general high-priority rules.

To prove $\forall x P(x)$ take any x_1 and prove $P(x_1)$
To prove $A \Rightarrow B$, assume A and prove B
To prove $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ prove all the A_i one after the other
To prove C if $A \vee B$ is a hypothesis, prove $(A \Rightarrow C) \wedge (B \Rightarrow C)$
To prove $\neg A$ assume A and search for a contradiction (i.e. prove $false$)

To assume A :
- if A is elementary (of the form $p(\ldots)$), or is an existential formula assume it as a new hypothesis;
- if A is a conjunction $A_1 \wedge A_2 \wedge \ldots \wedge A_n$, assume each A_i ;
- if A is an universal hypothesis, it is not added as a hypothesis; in the place, new rules are built, as explained in sections and 3.2 and 3.4.

Then, hypotheses are :
- either elementary (of the form $p(\ldots);$)
- either a disjunction;
- either an existential formula;
- either the constant $false$ (a contradiction has been found).

The conclusion (only one) is :
- either elementary;
- either an existential formula;
- either a disjunction;
- either the constant $true$ (the (sub)theorem has been proved) or $false$ (search for a contradiction)

A (sub)theorem is proved if the conclusion is $true$, or if a hypothesis is $false$, or if the conclusion is the same as one of the hypotheses.

Some other rules handle some special cases of disjunctive or existential hypotheses, or conclusion.

More over, here is an other example of a general rule which has a lower priority
- if there is, among the given definitions, a definition of the predicate of the conclusion, the conclusion may be replaced by its definition.

Notice that this is not the case for the hypotheses. For example, a conclusion $A \subset B$ may be replaced by its definition $\forall X(X \in A \Rightarrow X \in B)$ but a hypothesis $A \subset B$ is never replaced.

3.2 Built rules

By some transformations analogous to preceding ones, other rules are automatically built from the definitions, lemmas and universal hypotheses. The idea is to split and to simplify the statements as much as possible.

For example, from the definition of inclusion
$\forall A\ \forall B\ (A \subset B \Leftrightarrow\ \forall X\ (X \in A \Rightarrow X \in B)$
the following rule is built :
- if $A \subset B$ and $X \in A$ are hypotheses then add the hypothesis $X \in B$

A first example of a proof

Theorem to be proved $\forall A\ \forall B\ \forall C\ (A \subset B \wedge B \subset C \Rightarrow A \subset C)$

Proof (remember that new hypotheses are *added*, but a conclusion is *replaced*) by a new one.

introduce	hypotheses	conclusion
		$\forall A\ \forall B\ \forall C\ (A \subset B \wedge B \subset C \Rightarrow A \subset C)$
a, b, c		$a \subset b \wedge b \subset c \Rightarrow a \subset c$
	$a \subset b$	
	$b \subset c$	$a \subset c$
		$\forall X(X \in a \Rightarrow X \in c)$
x		$x \in a \Rightarrow x \in c$
	$x \in a$	$x \in c$
	$x \in b$	
	$x \in c$	*true*

3.3 Functional symbols

Strategies are designed to work with predicates rather than with functional symbols. DATTE worked only with predicates.

For example $inter(C, A, B)$ was used to mean that C is the intersection of A and B.
In following versions (see section 4), functional symbols have been authorized, but they are automatically "eliminated" in the following sense, $C : A \cap B$ is the same as $inter(C, A, B)$ and is handled as a predicate formula.

Easily done if arguments are constants (the prover has just to create a new constant), it is more complicated if there are variables. In particular, it depends if the functional symbol appears in a hypothesis or in a conclusion, as we will see in the next section.

3.4 Rules for functional symbols

From the definition of powerset $\forall A\ \forall X(X \in \mathcal{P}(A) \Leftrightarrow X \subset A)$
(for DATTE $\forall A\ \forall B\ (powerset(B, A) \Rightarrow \forall X(X \in B \Leftrightarrow X \subset A))$
the following rule is built :

- if $\mathcal{P}(A) : B$ and $X \in B$ are hypothesis
  then add the hypothesis $X \subset A$

From the definition of intersection $\forall A\ \forall B\ \forall X(X \in A \cap B \Leftrightarrow X \in A \wedge X \in B)$
(for DATTE $\forall A\ \forall B\ \forall C\ (inter(C, A, B) \Rightarrow \forall X(X \in C \Leftrightarrow X \in A \wedge X \in B))$
the following rules are built :

- if $A \cap B : C$ and $X \in C$ are hypotheses
  then add the hypothesis $X \in A$ if it is not yet a hypothesis.
- if $A \cap B : C$ and $X \in C$ are hypotheses
  then add the hypothesis $X \in B$ if it is not yet a hypothesis.
- if $A \cap B : C$, $X \in A$ and $X \in B$ are hypotheses
  then add the hypothesis $X \in C$ if it is not yet a hypothesis.

Notice that the condition "$C : A \cap B$" [resp. "$B : \mathcal{P}(A)$"] is a hypothesis which is verified only if the intersection of A and B [resp. the powerset of A] has already been introduced.

Moreover, the replacement of the conclusion by its definition is expressed by the following rule :

- if the conclusion to be proved is $X \in B$
  and if there is a hypothesis $b : f(a, \ldots)$
  and a definition $\forall A \forall B(B : f(A) \Rightarrow \forall X(X \in B \Rightarrow < definition >))$
  then replace the conclusion by its definition

Another example of a proof

Theorem to be proved $\forall A \forall B \forall C (\mathcal{P}(A \cap B) \subset \mathcal{P}(A) \cap \mathcal{P}(B)$

introduce	hypotheses	conclusion	
		$\forall A \forall B \forall C\ \mathcal{P}(A \cap B) \subset \mathcal{P}(A) \cap \mathcal{P}(B)$	
a, b		$\mathcal{P}(a \cap b) \subset \mathcal{P}(a) \cap \mathcal{P}(b)$	
c	$c : a \cap b$		
pc	$pc : \mathcal{P}(c)$		
pa, pb	$pa : \mathcal{P}(a)$, $pb : \mathcal{P}(b)$		
pd	$pd : pa \cap pb$	$pc \subset pd$	
		$\forall X (X \in pc \Rightarrow X \in pd)$	
x		$x \in pc \Rightarrow x \in pd$	
	$x \in pc$	$x \in pd$	
	$x \subset c$		
		$x \in pa \wedge x \in pb$	
subtheorem (1)			*subtheorem* (2)
		$x \in pa$	$x \in pb$
		$x \subset a$	
		$\forall X (X \in x \Rightarrow X \in a)$	
t		$t \in x \Rightarrow t \in a)$	analogous
	$t \in x$	$t \in a$	
	$t \in c$		
	$t \in a$		
		true	*true*

3.5 Processing of existential hypotheses

An existential hypothesis $\exists X p(X)$ may lead to the creation of a new element a verifying the property p. This treatment is not systematically done because it could be expansive.

For example, if f is a mapping from E to E and there is an element a in E, a rule (built from the definition of mappings) states that $\exists Y (Y \in E \wedge Y : f(a))$. Its treatment leads to the creation of an element y_1 such that $y_1 \in E$ and $y_1 : f(a)$. This could lead to an infinite sequence of elements $a \rightarrow y_1 \rightarrow y_2 \rightarrow \ldots$. More over, if f is a surjection, each element in A could lead to an infinite sequence of antecedents $\ldots \rightarrow x_2 \rightarrow x_1 \rightarrow a$. So, this treatment has a low priority and new elements are created one by one, and successively for all existential properties in the order where they have been added. The high-priority rules are applied again between two creations.

An example on mappings

Theorem to be proved

> Consider three mappings $f : A \rightarrow B$, $g : B \rightarrow C$, $h : A \rightarrow C$; if any two of the three mappings $h_o g_o f$, $f_o h_o g$, $g_o f_o h$ are injections (resp. surjections) and the third is a surjection (resp. injection), then f, g and h are bijections [one-to-one].

There are six different independent theorems to be proved. Each of them is split into two subtheorems (one for injection and one for surjection). Each of these 12 subtheorems needs to introduce several elements (5 for the easiest, 9 for the most difficult).

For example, for one of the easiest subtheorem, with $h_o g_o f$ injective, $g_o f_o h$ and $f_o h_o g$ surjective, to prove that h is injective, let $c1$ and $c2$ in C with the same image $a3 = h(c1) = h(c2)$ in A, the conclusion to be proved is $c1 = c2$.

The following elements must be created :
$c_4 = (g_o f_o h)^{-1}(c_1)$ (surjectivity of $g_o f_o h$), $a_5 = h(c_4)$, $b_6 = f(a_5)$, $c_1 = g(b_6)$ (surjectivity of h), $c_7 = (g_o f_o h)^{-1}(c_2)$ (surjectivity of $g_o f_o h$), $a_8 = h(c_7)$, $b_9 = f(a_8)$, $c_2 = g(b_9)$ (surjectivity of h),

then $h_o g_o f(a_5) = a_3 = h_o g_o f(a_8)$
$a_5 = a_8$ (injectivity of $h_o g_o f$)
$b_6 = b_9$ and $c_1 = c_2$ (definition of mapping), QED

DATTE proved theses theorems. It introduced some more useless elements (1 for the easiest, 11 for the most difficult).

Detailed proofs may be found in [12, 14].

3.6 Processing of disjunctive hypotheses

An existential property $A \vee B$ leads to splitting into two subtheorems, one of them has hypothesis A, the second one has hypothesis B. As existential hypotheses, they are not systematically treated. If an existential hypothesis had be treated too early, some parts of the proof which do not depend on it would be done in each subtheorem. Worse, if it is useless for the proof, both subtheorem proofs would be repeated identically. And the worst, if it is useless and its application may be expansive it would lead to infinite splitting.

There are really no reason to process disjunctive hypotheses before or after existential hypotheses. Only experiments showed that it was better to begin with existential hypotheses. It is easy to modify the priority to choose the opposite.

3.7 Negation

Negations are removed as much as possible. In my opinion, "positive" properties are more important than "negative" ones because they give more information and are easier to handle. Rules built from definitions are made to work with positive properties rather than negative ones. There are some exceptions where negative properties must be considered as positive.
Here are two usual relations for which mathematicians have a special notation :

$x \neq y$ for $\neg(x = y)$ and $x \notin y$ for $\neg(x \in y)$

$x \neq y$ is frequently useful. $x \notin y$ is useful for problems on complements of sets.
In natural language there are also specific words for these relations : *different* and *non-membership* (at least in French).
It is also the case when we speak of *non-empty* sets. This property is *positive* (there *exists* as least an element) and *being empty* is *negative* (there does *not* exist *any* element). So the defined property is

$\forall A(\text{nonempty}(A) \Leftrightarrow \exists x(x \in A))$

It is the same for non-disjoint sets

$\forall A \forall B(\text{nondisjoint}(A, B) \Leftrightarrow \exists x(x \in A) \wedge x \in B))$

We will see later (section 4) that, if the prover receives definitions

$\forall A(\text{empty}(A) \Leftrightarrow \neg\exists x(x \in A))$
$\forall A \forall B(\text{disjoint}(A, B) \Leftrightarrow \neg\exists x(x \in A) \wedge x \in B))$

it replaces them by

$\forall A$ $(\text{nonempty}(A) \Leftrightarrow \exists x(x \in A))$
$\forall A$ $(\text{empty}(A) \Leftrightarrow \neg$ $\text{nonempty}(A))$
$\forall A \forall C(\text{nondisjoint}(A, B) \Leftrightarrow \exists x$ $(x \in A \wedge x \in B))$
$\forall A$ $(\text{disjoint}(A, B) \Leftrightarrow \neg$ $\text{disjoint}(A, B))$

it is able to define $\text{disjoint}(A, B)$ as above.

3.8 High-order

DATTE was able to work in second-order [2]. For example, it is possible to define properties of a

[2] More exactly in "pseudo second order"; as formulas are written in Polish prefix notation, R cannot have an arity 0 and 2. A ternary symbol "." has been introduced, we write the prefix formula ". R x y" instead of R(x,y). It is the same for mappings which are handled as relations. ". f x y" means "f(x)=y". In the following version (see next section) it is really second order.

relation (variable R), such as transitivity, order, congruence, etc., and then express that a particular relation verifies theses properties.

For example, a relation may be transitive, an order relation, a congruence relation, ... Inclusion and inequality $\leq$ are order relations, equality of sets is a congruence. This was particularly useful in axiomatic set theory where the membership relation is a strict order on ordinal numbers.

It is the same for mappings. A particular mapping may be injective, surjective or one-to-one, etc.

These properties are defined for a relation as a variable.
For example, the definition

$\forall R$ (transitive(R) $\Leftrightarrow \forall X\ \forall Y\ \forall Z\ (R(X,Y) \wedge R(Y,Z) \Rightarrow R(X,Z))$) [3]
may be applied to the inclusion of sets or to any other defined binary relation.

3.9 Results

About 150 theorems have been proved in set theory, mappings, congruence relations, orderings, ordinal numbers. Most of them come from [5, 6, 17, 18]. Some of them are difficult and were not proved by other provers until a few years ago.. Many examples of proofs may be found in [12].

4 A knowledge-based prover using knowledge and metaknowledge (MUSCADET1)

The second great influence in the development of MUSCADET was the emergence of expert systems. DATTE has been rewritten as a knowledge-based system.

MUSCADET1 is composed of an inference engine, which interprets and executes rules, and of one or several bases of facts (bases of rules and metarules, internal representations of "theorems to be proved"). The inference engine was written in PL1 and was later translated in Pascal. Mathematical knowledge is expressed in a language close to usual mathematical language. A specific language has been defined to express rules.

Rules are either universal and put into the system, or built by the system itself by metarules from data (definitions and lemmas) given by the user. They are in the form

if <list of conditions> then <list of actions>.

Conditions are normally properties that are quickly verified. Actions may be either elementary actions which are quickly executed, or super-actions which are defined by packs of rules.

The representation of a theorem to be proved (or a sub-theorem) is a description of its state during the proof. It is composed of objects that were created, of hypotheses, of a conclusion to be proved, of rules called active rules, possibly of sub-theorems, etc. At the beginning, it is only composed of a conclusion, which is the initial statement of the theorem to be proved, and of a list of rules, which are called active, i.e. relevant for this theorem, and which have been automatically built.

Active rules, when applied, may add new hypotheses, modify the conclusion, create new elements ("objects" in the new terminology of MUSCADET1), create sub-theorems or build new rules which are local for a (sub-)theorem.

All the strategies which were programmed in DATTE are expressed as rules and metarules. They are more easily expressed and handled.

It is possible to write formulas really in second order and also to use predicates or functions with a flexible arity.[4] This last possibility is particularly useful for conjunctions and disjunction. For example, to prove a conjunction of n terms, instead of n-1 splitting into 2 sub-terms, we have one splitting into n terms. It is the same for a hypothesis which is a disjunction. As splitting is expensive in time, this is very useful. Not so important but convenient is the fact that the same relation symbol may be binary, ternary or more.

[3] for DATTE . $R\ X\ Y \wedge$. $R\ Y\ Z \Rightarrow$. $R\ X\ Z$

[4] Formulas are written as lists, (P X Y) for P(X,Y), (AND C1 C2 C3) for $C1 \wedge C2 \wedge C3$

Some new mechanisms have been defined. Here is a very useful one which allows the use of functional symbols in the data.

Elimination of functional symbols

As explain in section 3.3, the prover works with predicates instead of functional symbols. If arguments are constant, it is easy, $p(f(a))$ may be replaced by

$b : f(a) \wedge p(b)$

where b is a new object.

If X is a variable, $p(f(X))$ may be replaced either by

$\forall Y\ (Y : f(X) \Rightarrow p(Y))$,

or by

$\exists Y\ (Y : f(X) \wedge p(Y))$.

The better form depends on the position of $p(f(X))$ in a statement (*positive* or *negative*).

So, a new quantifier noted "!" has been created, which means "for the only ... equal to ...".

$!Y : f(X)\ p(Y)$ [5] means "for the only Y equal to $f(X)$ then $p(Y)$ holds"

This transformation is systematically done in the beginning of the proof for the theorem to be proved and for definitions and lemmas. Then specific rules are applied while building rules, then during the proof.

Sometimes "!" is handled as $\forall$

To prove $!Y : f(X)\ p(Y)$ create a new object $Y1$ such that $Y1 : f(X)$ and prove $p(Y1)$.[6]

Other times, it is handled as $\exists$

If $!Y : f(X)\ p(Y)$ is a hypothesis,

then create a new object $Y1$ and add the hypotheses $Y1 : f(X)$ and $p(Y1)$.[7]

This last treatment is systematically done, contrary to the same treatment for existential hypotheses.

5 Application to difficult mathematical domains

All the mechanisms described in the preceding section have been expressed as rules and metarules, instead of being programmed. Moreover, it allowed to easily express specific mathematical know-how, general or for specific mathematical domains. It has been applied to topological linear spaces and to cellular automata. For these domains, it has been necessary to explicit specific know-how

For this, I observed myself while trying to prove theorems in these domains. I also observed mathematicians trying to solve problems. The problems were chosen by myself or by them. I have asked them to think aloud and when they paused I encouraged them to speak, or I asked them questions on what they were thinking about. It is not sure that the reasons they gave to explain why they did something or another one was really the right ones. But the aim was not to exactly simulate the human reflexion (partly subconscious) but to find legitimate and general reasons to do something useful (see [13, 14]).

After these observations I have been able to write rules expressing know-how, and Muscadet has been able to prove some rather difficult theorems.

5.1 Topological Linear Spaces

Muscadet worked on some problems of [24]. After some easy theorems I worked in particular on the two following difficult theorems.

ThA. *If U is a neighbourhood of the origin O of a TLS (Topological Linear Space), then there exists a neighbourhood V of O, included in U, star-shaped and symmetric relative to O.*

[5] p(f(X)) is written (p(f X)), $!Y : f(X)\ p(Y)$ is written (only (:(f X)Y)(p Y))

[6] There does not remain any free variable in a conclusion to be proved, so X has already been instantiated when this rule is applied.

[7] It is the same for the hypotheses.

ThB. *If U is a neighbourhood of the origin O of a TLS (Topological Linear Space), then there exists a neighbourhood V of O, included in U, star-shaped and symmetric relative to O, such that* $x \in V$, $y \in V$ *implies* $[x, y] \subset U$.

The observed researchers were authorized to use, and later MUSCADET received, knowledge and know-how about linear spaces and topological spaces but only the definition of topological linear spaces. It is interesting to remark that as one of the researchers was just about giving up, I reminded him, and insisted, that he was supposed to know nothing excepted the definition. Then he immediately had the good idea. Probably he was too knowledgeable about TLS to think to the elementary definition. Then he quickly applied it to the good operation. With some difficulty he was able to justify the fact that he did not even think to apply it to the other operation.

MUSCADET proved this theorem with a specific knowledge base. This allowed handling difficult notions, and using non-trivial knowledge and know-how about these notions. This knowledge is not formal, but rather operational. For example, the formal definition of neighbourhoods is of little help, it is more useful to know methods to prove that a set is a neighbourhood, and heuristics to choose one of these methods in different situations, and also to know the particular bases of neighbourhoods usually used in particular situations. The knowledge base also contained rules for symbolic calculation, handling product sets and special cases of products, for example products of two equal sets.

The knowledge base and the proofs of some theorems are given in detail and analysed in [13].

5.2 Kuratowski theorem

The Kuratowski theorem about ordered-pair $(a, b) = (c, d) \Rightarrow a = c \wedge c = d)$ was known as a difficult theorem to be automatically proved although it is trivial for mathematicians.

The definition of ordered pair (a, b) is that used in axiomatic theory :
ordered-pair $(a, b) = \{\{a\}, \{a, b\}\}$
singleton $\{a\} = \{x \mid x = a\}$
unordered-pair $\{a, b\} = \{x \mid x = a \vee x = b\}$
Mendelson [9] says "The definition of (a, b) does not have any intrinsic intuitive meaning. It is just a convenient way (discovered by Kuratowski) to define ordered pairs so that one can prove the characteristic property of ordered pairs expressed in the proposition".

Mendelson's proof contains 4 splittings, 6 subtheorems.
Resolution proofs are very long.
Brown's proof [3] by the "Fundamental Deduction Principle" is also very long.
MUSCADET proof is rather long : 8 splittings, 13 subtheorems.

With an additional know-how knowledge, MUSCADET proved the theorem with only one splitting and 2 subtheorems. The idea is to give knowledge about degenerate cases. In this example the hint is to split into $a = b$ and $a \neq b$, and to know that $\{a, a\} = \{a\}$ and $\{a, b\}$ is different from any unit set if $a \neq b$.

MUSCADET proof may be found in [14].

5.3 Cellular automata

Work on Cellular automata has been done on request of a mathematician. In this domain, researchers create automata to solve problems. The creative part is to imagine and formalize the automata. Although they may intuitively justify their results and verify them for many values of n. They also must formally and rigorously verify the correctness of their automata. This is tedious for them. Would MUSCADET or a future prover be able to do it ?

As an experiment, MUSCADET worked on a contribution of Mazoyer [8].

n machines are all in the quiescent state L at time 0. The problem is to define a set of states and transition rules so that all machines fire, i.e. enter a special distinguished state for the first time at the very same moment in the minimal time $(2n-1)$.

Mazoyer found a 6-state minimal time solution. After having created the automaton, he had to prove that it is a solution of the problem. He decomposed the proof into 11 lemmas which lead to the final theorem.

Here are some notations and an example of definition.

Notations
If $\leq m < n$, we let DD_m be the set $\{(m-i, m+i), (m-i, m+i+1) | i \in \{-1, 0, ..., m-1\}\}$
If $\leq m < n$ and $1 \leq k < j \leq m+1$, we let $DD_{m,j,k}$ be the set
$DD_m \cap \{(l,t) | k \leq l \leq j\} = \{(m-i, m+i), (m-i, m+i+1) | i \in \{m-j, 0, ..., m-1\}\}$
Definition
Suppose $j \geq k$ and $k \geq 1$; we say that $DD_{m,j,k}$ is basic in state x if
all sites $(i, 2m-i), (i, 2m-i+1)$ with $i \in \{m-j+1, ..., m-k-1\}$ have state x.

First lemma (trivial for the researcher, easy for MUSCADET)
Every site(K,t) with $K > t$ has state L.

A simplification of the second lemma to be proved
Let $1 < m$, $1 < k$, $\leq k$ and $j \leq m+1$. If $DD_{m,j,k}$ is basic in A and if site $(k-1, 2m-k+1)$ has state A, then the first diagonal of $DD_{m+1,j,k}$ is also basic in A.

MUSCADET spent an unreasonable time while working on the order relation on natural numbers, whereas observed researchers reflexively applied the order properties. In particular as order may be large or strict ($x \leq y \Leftrightarrow x < y \vee x = y$), a great many splittings are done. It had to know that properties as *state* or *site* were more important that ordering.

Moreover, MUSCADET was not able to calculate, even with concrete numbers, it had to receive some arithmetic rules.
To help MUSCADET, I also gave it the following knowledge :
- introduce the concept of *successor* suc(x) for $x+1$ and the following properties
$x < y \Rightarrow \text{suc}(x) \leq y$
$x < y \Rightarrow x \leq y$
- introduce a special function $f(m,i)$ for $2m-i$ and the following properties
$f(m, i+1) = f(m,i) - 1$
$f(m+1, i) = f(m,i) + 1 + 1$
details may be found in [14].

It would have been necessary to combine theorem proving MUSCADET and symbolic computation. A work has been started in this direction, then continued with MUSCADET2 where Prolog made it easier, then given up.

6 A Prolog prover using declarative as well as procedural knowledge (MUSCADET2)

To make the expression of know-how easier to write, and in particular allowing the expression of procedural strategies, MUSCADET has been rewritten in Prolog. This allowed more flexibility for knowledge from really declarative rules to quite procedural rules, as well as more or less declarative rules, or mixing declarative and procedural items in a rule.

It is useful for example to scan a conjunctive conclusion and prove each element instead of doing many splittings. And it is crucial to avoid to split a disjunctive hypothesis if the nth term of the disjunction is trivially verified.

Each time that a developer discovers that a task takes more time that it should do, he/she may define a Prolog predicate to speed up.

The inference engine of MUSCADET1 has been replaced by the Prolog interpreter (unification, manipulation of expressions) completed by a small inference engine written with few Prolog clauses.

Prolog conventions are used for variables and constants. Formulas are written in infix and partially parenthesized notations, almost as mathematicians do.
For example

```
A subset B <=> forall (X, X in A => X in B)
```

and also

```
A inter B = [X, X in A and X in B]
forall (X in A, exists(Y in B, p(X,Y,A,B)))
forall(X in real,(forall Y in real,(forall(Z in real,X$<$Y and Y$<$Z => X$<$Z)))
```

A consequence of this choice is that, as in DATTE before, second-order has to be a pseudo second-order. A Prolog predicate is defined for this. Its name is "`..`" and the argument is a list the first argument of which is the relation or the function symbol. This choice has been made for the following reason. In Prolog we have the predefined predicate "`=..`" with

```
r(a,b)  =..  [r,a,b]
```

`r(a,b)` and `..[r,a,b]` are not unified by Prolog but

```
r(a,b)  =.. H      gives      H = ..[r,a,b]
```

Here are some examples of how rules and actions are written [8].

```
rule(N, =>) :- concl(N, A => B), addhyp(N, A), newconcl(N, B)
rule(N, forall) :- concl(N, forall(X,C)), create_object(X1)
                   addobj(N, X1), replace(C, X, X1, C1),
                   newconcl(N, C1).
newconcl(N, C) :- retractall(concl(N, _)), assert(concl(N, C))
ajhyp(N, H) :- ( H = A and B -> addhyp(N, A), addhyp(N,B)
                ...
                ; H = ..[R, X, Y] -> H1 =.. [R, X, Y], addhyp(N, H1)
                ; H = forall(_, _) ->  buildrule(H,N)
                ...
                ; assert(hyp(N, H))
                )
```

7 A prover knowing nothing about mathematics apart logics Application to TPTP problems – CASC competitions – MUSCADET3

To work on problems of the TPTP [9] Library, MUSCADET had to first translate the TPTP statements into MUSCADET statements. Since they were first-order formulas, they were not very different, except for some points.

First, some abbreviations mathematicians are comfortable with were forbidden, for example relativized quantifiers could not be used. Formulas such as

```
forall(X in A, p(X))
```

had to be translated into

```
(! [X] : member(X,A) => p(X))
```

in TPTP syntax. On the opposite, TPTP uses the following abbreviation

[8] Slightly simplified for readability

[9] Thousands of Problems for Theorem Provers. Created in 1993 by Sutcliffe and Suttner, this Library first contained only problems expressed as sets of clauses (CNF). In 1999 some problems expressed as first-order formulas (FOF) have been introduced : 217 FOF problems among 3330, 5 of them in set theory (SET). Now (in 2014), there are about 20000 problems, about 8000 of them in FOF and about 1400 of them in SET+SEU (SET continued).

(! [X,Y,Z] : p(X,Y,Z))
instead of
`forall(X, forall(Y, forall(Z, p(X,Y,Z))))`

The most important constraint was the fact that TPTP knows logical notations but knows nothing about mathematics. Even the membership relation, which is a primitive notion in set theory, had to be handled as any other binary relation. This added a difficulty for MUSCADET which had to be adapted. In particular, methods using the membership relation had to be rewritten in a more general manner (for example, methods handling functional symbols).

Still more important, in mathematical knowledge bases, each mathematical statement was labeled as a *definition* or a *lemma*. Lemmas are axioms [10] (in an axiomatic context) or preliminary known results necessary to prove a theorem (for example known results of a sub-domain of the studied domain). MUSCADET had to recognize statements which are definitions. If a definition is not recognized as a definition, it will still be used but in a less efficient manner.

The Library is fed with contributions of researchers. I proposed some problems.

Since 1999, MUSCADET has participated to the CADE ATP System Competition (CASC) and has proved some theorems that no other prover could prove. This shows its complementarity with other provers.

Then, in MUSCADET3, the syntax used to express formulas is the TPTP syntax, not only for input formulas and output, but also for internal representations, and results display. To avoid confusions, although it was not necessary, some internal symbols have been modified, such as `:` which became `::` in `only(f(X)::Y,p(Y))` to avoid confusion [11] with `! [X] : p(X)`

This option is still available for non TPTP problems.

8 Relevant proofs – Utilities and tools (MUSCADET4)

Until then, MUSCADET displayed all the steps of its search or, on option, only the result (proved, unsatisfiable, gave up or time out). It was a great weakness because to verify that the proof is correct, or to try to understand the reasons of failure, one had to manually extract the relevant steps of the displayed trace. It might be long and difficult. Moreover, in CASC competitions, systems that do output solutions have been highlighted in the presentation of results.

This new version is now able to extract relevant steps. It has been difficult to implement because of the calls to procedural actions in the rules.

Many modifications had to be done :
- step numbers are new parameters, in facts and rules (the number of the current step in the actions and the preceding step numbers of each fact in the conditions),
- an explanation is given or automatically built for each step,
- these parameters are passed in many Prolog predicates,
- all steps are memorized with these informations,
- the path from the success step up to the first step is extracted.

Examples of MUSCADET proofs may be found in
http://www.math-info.univ-paris5.fr/~pastre/muscadet/examples (exemples for the French version).

[10] TPTP uses the keyword axioms with another meaning
[11] In the reader's mind, the machine doesn't matter

The prover may be simply called by an executable C program, with the address or the name of a theorem to be proved, and options if needed (see [15]).

It may also be called under Prolog. In this case it is possible to display all the facts, and also all the memorized steps and examine them, with the help of Prolog commands.

There are also some shell scripts to analyse the proofs.

9 Conclusion

MUSCADET functions in a manner which is quite different from resolution-based provers. It uses methods based on natural deduction and is a knowledge-based system.
The reasons for its efficiency in a number of circumstances have been analysed in details with many examples in [16]. These include
• the fact that the growth of the bases of facts in linear, not exponential ;
• the importance and efficiency of the chosen representations of the problem ;
• the splitting of a (sub)theorem in many subtheorems easier to prove, independent or not ;
• the treatment of functional symbols which flattens the handled expressions ;
• the replacement of definitions and universal hypotheses by natural and efficient rules ;
• the treatment of equalities and negations which removes them as far as possible.

MUSCADET is efficient for everyday mathematical problems which are expressed in a natural manner, for example in naive set theory. It is not efficient for problems which are defined axiomatically, from a logician's point of view, for instance in the fields of axiomatic geometry or axiomatic set theory.
MUSCADET is efficient to solve problems which involve many axioms, definitions or lemmas. It is not efficient at all to solve problems which involve only one large conjecture and no intermediary definitions.

MUSCADET cannot prove some theorems that resolution-based provers can easily prove. Nowadays, the best generalist theorem provers are based on the resolution principle, as shown by the results of the CASC competitions [22]. The winner of the FOF[12] division has been the resolution-based prover Vampire [19] for several years.
However MUSCADET showed its complementarity with resolution-based provers. More ever, a great advantage is that it displays proofs easily readable by a human reader.

Nowadays, MUSCADET is no longer being actively improved, except the interface to make it more and more easy to use.
Nevertheless, the analyses of failures (obtained from CASC or by users) often lead to the correction of bugs or to some improvements of some rules, metarules or heuristics.

Another way to have theorem proving progress with MUSCADET is to have it cooperate with other provers.

References

[1] W. W. Bledsoe, Splitting and reduction heuristics in automatic theorem proving, *Journal of Artificial Intelligence*, 2:55–77, 1971.

[2] W. W. Bledsoe, Non-resolution theorem proving, *Journal of Artificial Intelligence*, 9:1–35, 1977.

[3] F.M. Brown, An experimental logic based on the fundamental deduction principle, *Journal of Artificial Intelligence*, 30:117–263, 1986.

[12] First Order Formula

[4] H. Gelertner, realization of a geometry-theorem proving machine, In: Feigenbaum and Feldman, editors, *Computers and thought*, 135–152, McGraw-Hill, 1963

[5] P.R. Halmos. *Naive set theory*, Van Nostrand, Princeton, 1960

[6] J.L. Krivine, *Théorie axiomatique des ensembles*, PUF, Paris, 1969

[7] A Newel, J.C. Shaw, and H.A. Simon, Empirical explorations with the logic theory machine: a case study in heuristics. *Proc. Western Joint Computer Conf.*, 1957; In: Feigenbaum and Feldman, editors, *Computers and thought*, 109–133, McGraw-Hill, 1963.

[8] J. Mazoyer, A six-state minimal time solution to the firing squad synchronisation problem, *Theoretical Computer Science*, 50:183–238, 1987

[9] E. Mendelson, *Introduction to mathematical logic*, Van Nostrand, 1964

[10] A. Newel, J.C. Shaw, and H.A. Simon, Chess playing programs and the problem of complexity, *IBM Journal of Research and development*, 1958; In: Feigenbaum and Feldman, editors, *Computers and thought*, 39–70, McGraw-Hill, 1963.

[11] A. Newel and H.A. Simon, GPS - a program that simulates human problem solving, *Proc. of a Conference on Learning Automata*, 1961; In: Feigenbaum and Feldman, editors, *Computers and thought*, 279–293, McGraw-Hill, 1963.

[12] D. Pastre, Automatic theorem proving in set theory, *Journal of Artificial Intelligence*, 10:1-27, 1978

[13] D. Pastre, MUSCADET: an automatic theorem proving system using knowledge and metaknowledge in mathematics, *Journal of Artificial Intelligence*, 38(3):257–318, 1989.

[14] D. Pastre, Automated theorem proving in mathematics, *Annals on Artificial Intelligence and Mathematics*, 8(3-4):425–447, 1993

[15] Pastre D., Muscadet version 4 : User's manual, http://www.math-info.univ-paris5.fr/∼pastre/muscadet/manuel-en.ps, 22p, 2011
Muscadet version 4 : Manuel de l'utilisateur, http://www.math-info.univ-paris5.fr/∼pastre/muscadet/manuel-fr.ps, 23p, 2011

[16] D. Pastre, Complementarity of a natural deduction knowledge-based prover and resolution-based provers in automated theorem proving, internal report, http://www.math-info.univ-paris5.fr/∼pastre/compl-NDKB-RB.pdf

[17] M. Queysanne, *Algebre M.P. et Spéciales A A'*, Collection U, Armand Colin, Paris, 1969

[18] L.E. Sigler, *Exercises in set theory*, Van Nostrand, London, 1966

[19] A.Riazanov and A.Voronkov, The design and implementation of Vampire, *AI Communications*, 15(2-3):91-110, 2002,

[20] J.A. Robinson, A machine oriented logic based on the resolution principle, *J.ACM* 12:23-41, 1965

[21] A heuristic program that solves symbolic integration in fresmann calculus MIT, 1961 In: Feigenbaum and Feldman, editors, *Computers and thought*, 191–203, McGraw-Hill, 1963

[22] G. Sutcliffe, C. Suttner, The state of CASC, *AI Communications*,19(1):35-48, 2006

[23] G. Sutcliffe, The TPTP problem library and associated infrastructure: the FOF and CNF parts, v3.5.0, *Journal of Automated Reasoning*, 43(4):337-362, 2009 http://www.cs.miami.edu/∼tptp

[24] F. Valentine, *Convex sets*, McGraw-Hill, New-York, 1964

[25] H. Wang, Towards mechanical mathematics, *IBM J.Res.Develop*, 4, 2–22, 1960.