Analogy and Abduction in Automated Deduction

Gilles Defourneaux and Nicolas Peltier
46 Avenue Felix Viallet
38031 Grenoble FRANCE
Tel: (+33) (0)4-76-57-46-59
Gilles.Defourneaux@imag.fr, Nicolas.Peltier@imag.fr

Abstract

A method is presented for analogical reasoning in Automated Deduction. We focus on the abductive aspects of analogy and give a unified treatment for theorems and non-theorems. Abduction allows to deal with *partial* analogies thus strongly increasing the application field of the method. It also allows to detect "bad analogies" in several cases. Explanatory examples as well as more realistic examples quantifying the effects of using analogy (for theorem-proving *and* for counter-example building) are given.

1 Introduction

Analogy is in the very heart of human reasoning, in particular in Mathematics. Roughly speaking, *reasoning by analogy* consists in using informations deduced from the solving of a given problem *or set of problems* (the *source problems)* for solving a new one (the *target problem).* In Artificial Intelligence and Automated Deduction, the mechanization of this approach is a crucial issue (see for example [Bledsoe, 1977; Plaisted, 1981; Hall, 1989]). Moreover analogy is also an intrinsically interesting way of reasoning: discovering similitaries between existing proofs or theorems can be of highest importance (for instance in mathematical practice and teaching). As far as we know, in all the existing works in theorem-proving, analogy is used for *finding the proof of* a given target theorem from an existing one. This view of analogy is very limited since analogy can obviously be also useful if the target or source problems are *not* theorems: in this case one can try to find a counter-example of the target conjecture by using counter-examples of the source non-theorems. In this paper we give an unifying treatment of these two cases (theorems and non-theorems). *No assumption is made about the way these proofs and counter-examples are generated* (any existing method for search for proofs or counter-examples can be used: (hyper)resolution or tableaux-based methods, connexion method, but also (finite) model builder such as FINDER [Slaney, 1993], SEM [Zhang and Zhang, 1995] or the method RAMC [Bourely *et* a/., 1994] looking *simultaneously* for a proof or a counter-example of a given formula *T.*

Other approaches have been proposed to tackle the problem of analogy by second-order means. In [Boy de la Tour and Caferra, 1987], the paradigm of "propositions as types" is used and proofs are represented as terms. Higher-order functions are then applied to transform the base proof into the target one. In [Kolbe and Walther, 1995], higher-order evaluation techniques are used to refine the problem and ultimately have its premisses match with axioms of the calculus, allowing lemma speculation as a side effect. However, both approaches only deal with proofs (not counter-examples). To the best of our knowledge there is no other approaches allowing to deal with model building by analogy in first-order logic.

To make the presentation of the method shorter, we assume that the problems are specified in clausal form and that we use a refutational approach. Therefore, formulae are sets of clauses, proofs are *refutations* and counter-examples are *models.*

Analogy and abductive reasoning

According to Peirce [Hartshorne *et* a/.,], *analogy* can be seen as an *induction* and an *abduction* followed by a *deduction.* Our approach to analogy follows directly these steps. It can be summarized as follows. We assume a knowledge base *K* containing theorems with their proofs and non-theorems associated with counter-examples.

Generalization step. The first step consists in a *generalization* occurring at the presentation of a new source formula *Ts* to *K. Fs* is transformed into a *more general formula* and stored into the knowledge base. This corresponds to the *inductive* part of analogy.

Matching step. The second step applies when a new target conjecture *TT* is considered. It consists in trying to find one (or more) "analogical" formulae in *K.* Then the proof or the counter-example of the conjecture F_T is built *from* the proofs/counter-examples of the formulae in *K.* This corresponds to the *deductive* part of analogy.

However in most cases *TT cannot* be directly (dis)proved by using only the information in *K* (this is obviously possible only if *TT* is an instance of a problem in *K).* However, even if a proof or a counter-example of F_T cannot be straightforwardly deduced from the known formulae, the use of the informations stored in K will very likely provide interesting hints for finding a proof or a counter-example of $\mathcal{F}_\mathcal{T}$. For example it can guide lemmata generation that could be completed later by using any existing theorem prover or model builder. This corresponds to the *abductive* part of analogy: finding hypotheses that allows to prove *or to disprove* the target conjecture.

The *generalization step* has been precisely defined in [Bourely *et al.*, 1996], where an algorithm is given to transform any formula $\mathcal{F}$ into a "more general formula" $\mathcal{F}_{gen}$. The *matching* step is defined in [Défourneaux and Peltier, 1997]. *In this paper we focus mainly on the abductive part of analogical reasoning,* that is to say on the generation of lemmata. We propose a *partial* matching algorithm especially devoted to finding such lemmata, i.e. abduction will be incorporated to step 2.

Abductive reasoning is the process of generating the *explanations* of a given fact (see for example [Polya, 1973; Pople, 1973; Console *et al.*, 1991; Hobbs *et al.*, 1993]). It has deep connections with other forms of reasoning such as model building [Inoue *et al.*, 1993; Console *et al.*, 1991]. Aristotle calls *abduction* a syllogism "in which the major is sure and the minor only probable" (see [Lalande, 1980], page 1). More recently, it is defined by Peirce [Peirce, 1955] as the process of finding the *minor premise* from a *major premise* and the *conclusion:* for example "infer" A from $A \Rightarrow B$ and B. Peirce clearly points out the importance of abductive reasoning in science (he gave as a paradigm the discovery of Kepler's laws). From a deductive point of view, this inference is clearly *not* sound. However it provides interesting informations for proving 5, since A can be considered as a *lemma,* whose proof immediately yields a proof of B.

The aim of this paper is to show how to use analogy for generating such lemmata. Since our approach deals with simultaneous search for proofs and counter-examples, the notion of *lemma* is *much more general* than the standard one: it can be either a conjecture that (if it is true) is sufficient for proving B *or* a *partial counter-example* that must be completed and extended for finding a counter-example of B.

2 Basic notions

We assume the reader is familiar with the usual terminology of First-Order Logic and Automated Deduction. We briefly review most of the basic notions used throughout this work.

Let Σ be a set of functional symbols, Ω be a set of predicate symbols and $\mathcal{X}$ be an (countable) infinite set of variables. Let a be a function mapping each symbol in $\Sigma \cup \Omega$ to a natural number (the *arity* of the symbol). Function symbols of arity 0 are called *constants* (denoted by a, b, ...). The set of *terms* $\tau(\Sigma, \mathcal{X})$ is defined as usual over the alphabet $\Sigma, \mathcal{V}$ (where $\mathcal{V} \subseteq \mathcal{X}$). If $\mathcal{V}$ is empty, $\tau(\Sigma, \mathcal{X})$ is denoted by $\tau(\Sigma)$. An *atom* is of the form $P(t_1, \dots, t_n)$, where $P \in \Omega$, $arity(P) = n$ and $\forall i \in [1..n].t_i \in \tau(\Sigma, \mathcal{X})$. A *literal* is either an atom or the negation of an atom. If p is a literal, $\neg p$ denotes the literal with the same predicate symbol and the same arguments than p but with different sign. A *clause* is a finite set (or disjunction) of literals. First-order formulae are built as usual over atoms by using the logical symbols $\vee, \wedge, \neg, \exists, \forall \dots$. By $\mathcal{V}ar(E)$ we denote the set of (free) variables occurring in the expression (term, clause, atom, literal...) E. A term or a clause containing no variables is called *ground*. The notion of *substitution* is defined as usual. The result of applying a substitution σ to a term t is noted $t\sigma$. The domain of a substitution σ is the set of variables x such that $x\sigma \not\equiv x$ (noted $\mathcal{D}om(\sigma)$). If for all variables $x \in \mathcal{D}om(\sigma)$, $x\sigma$ is ground then σ is called *ground*.

For any set of clauses S we denote by $\mathcal{S}(S)$ the set of ground instances of clauses in S.

As is well known, a ground clause C' subsumes a ground clause C (noted $C' \leq_s C$) iff $C' \subseteq C$. If C, C' are two clauses, $C \leq_s C'$ iff for all $D' \in \mathcal{S}(C')$, $\exists D \in \mathcal{S}(C) / D \leq_s D'$.

2.1 Higher order formulae

Step 2 of the method (see Introduction) needs an algorithm transforming a given set of clauses S into a more general one. The latter is represented by second order terms and clauses. Higher-order terms and formulae are built as usual from a signature $\mathcal{V}, \Sigma, \Omega$ by using *application* and *abstraction* (see [Huet, 1975]).

The set of types $\mathcal{S}_k$ of order k is defined as follows. If $k = 1$, $\mathcal{S}_k = \{T\}$ (base type). If $\forall i \in [1..n].t_i \in \mathcal{S}_{k_i}$ and $s \in \mathcal{S}_k$, and $k = 1 + \max(k_1, \dots, k_i)$, then $t_1 \times \dots \times t_n \rightarrow s \in \mathcal{S}_k$.

Let $\mathcal{S} = \bigcup_{i=1}^{\infty} \mathcal{S}_i$. Let $(\mathcal{V})_{t \in \mathcal{S}}$ a set of disjoint infinite (countable) sets of variables (called variables of type t). Let $\mathcal{V} = \bigcup_{t \in \mathcal{T}} \mathcal{V}_t$, let $(\Sigma)_{t \in \mathcal{S}}$ a collection of disjoint set of constants of type t. We assume that $\mathcal{V} \cap \Sigma = \emptyset$.

The sets of terms $\tau_t(\Sigma, \mathcal{V})$ of type t are built as usual from $\mathcal{V}, \Sigma$ using *application* or *abstraction*. We denote by $\tau(\Sigma, \mathcal{V}) = \bigcup_{t \in \mathcal{T}} \tau_t(\Sigma, \mathcal{V})$, the whole set of terms. The definition of formulae, equational formulae, substitutions, etc. can be extended straightforwardly to higher-order term. We do not give all the definitions (see for example [Lugiez, 1995]).

Roughly speaking sets of clauses will be represented by sets of *generalized clauses* i.e. clauses where the function symbols are replaced by *variables* of order 2.

A *generalized formula* is a formula such that each bound variable is of order 1, and each free variable of $\mathcal{F}$ is of order 1 or 2 and occurs in a term of the form $X(\bar{t})$. A *generalized clause* is of the form: $\forall \bar{x}.C$ where C is a generalized formula of the form $\bigvee_{i=1}^{n} P_i$ (where P_i are generalized literals and $\bar{x}$ a n-uple, possibly empty, of variables of order 1. A *generalized sequence* is a finite sequence of generalized formulae. The set of *instances* of S is the set of sequences of clauses $S\sigma$ such that σ is a ground substitution of $\mathcal{V}ar(S)$.

3 An order among formulae

Before presenting our method we must clarify the notion of generalization and give a precise definition of it. A definition of this notion was given in [Bourely *et al*, 1996]. We give here a new refined definition of the generalization order that takes into account the *semantic* aspect of the clauses (i.e. the set of ground clauses denoted by the set) rather than its syntax.

The underlying ideas of this ordering are the following. Informally, a given theorem S will be said to be "more general" than a theorem S' iff the hypotheses of S are *weaker* than the one of S' or if the conclusion of S is *stronger than* the ones of S'. Indeed, it is obvious that S provides more information than 5". If sets of clauses are considered, the set of hypotheses is the set of clauses S itself, and the conclusion is $\Box$ (the empty clause). Hence S will be *more general* than S' iff each clause in S belong to S', i.e. if $S \subseteq S'$. However, what is important here is *not* the sets of clauses S and S' by themselves but rather the set of ground clauses they denote. Therefore, we must take into account the two following possibilities:

1. The set of clauses S' does not explicitly contain S but *any ground clause* denoted by S belongs to the set of ground clauses denoted by S', i.e. $\mathcal{S}(S) \subseteq \mathcal{S}(S')$. In this case S can be said to be "more general" than S'.
2. $C \in \mathcal{S}(S)$ and $C \notin \mathcal{S}(S')$ but C is *subsumed* by a clause C' in $\mathcal{S}(S')$.

In order to capture these two cases we introduce the following relation $\sqsubseteq$ between sets of clauses.

Definition 1 *let S and S' be sets of clauses. We note $S \sqsubseteq S'$ iff for all $C \in \mathcal{S}(S)$ there exists $C' \in \mathcal{S}(S')$ such that $C' \leq_s C$.*

Lemma 1 *If $S \sqsubseteq S'$ then $S' \models S$.*

Proof 1 *Assume that $S \sqsubseteq S'$. Let $\mathcal{I}$ be a model of S'. By definition if $C \in \mathcal{S}(S')$ then $\mathcal{I} \models C$. Let $D \in S$. We have $S \sqsubseteq S'$ hence by definition there exists $C \in S'$ such that $C \subseteq D$. $\mathcal{I} \models C$ (since $C \in S'$) hence $\mathcal{I} \models D$.*

An **unsatisfiable** set of clauses S is said to be *more general* than S' iff $S \sqsubseteq S'$. A **satisfiable** set of clauses S will be said to be *more general* than S' iff the hypotheses of S are stronger than the one of S' (this implies that S is false in more interpretations than S'). Consequently a satisfiable set of clauses S will be said to be *more general* than a set S' iff $S' \sqsubseteq S$.

Please note that these two notions of generalization are *not* equivalent. The following definition formalizes **this idea.**

Definition 2 *Let T_1 and T_2 be two sequences of sets of generalized clauses. T_1 is said to be more general than T_2 (noted $T_1 \succeq T_2$) iff for all substitution σ_2 of $\mathcal{V}ar(T_2)$, there exists a substitution σ_1 of $\mathcal{V}ar(T_1)$ such that*

- *T_1 is unsatisfiable and $T_1\sigma_1 \sqsubseteq T_2\sigma_2$*
- *T_1 is satisfiable and $T_2\sigma_2 \sqsubseteq T_1\sigma_1$.*

Proposition 1 *$\succeq$ is a pre-order.*

Example 1 *Let $t_1 : \forall x.P(x) \wedge \neg P(a)$, $t_2 : P(a) \wedge \neg P(a)$. t_1 and t_2 are two unsatisfiable sets. The set of ground clauses denoted by t_1 and t_2 are respectively $\{P(t)/t \in \tau(\Sigma, \mathcal{V})\} \cup \{\neg P(a)\}$ and $\{P(a), \neg P(a)\}$. We have $t_2 \sqsubseteq t_1$, hence t_2 is more general than t_1.*

Example 2 *Let $t_1 : \forall x.P(f(x)) \wedge \neg P(a)$ and $t_2 : P(f(a)) \wedge \neg P(a)$. t_1 and t_2 are satisfiable. We have $t_2 \sqsubseteq t_1$, hence t_2 is less general than t_1. t_1 is indeed falsified by more interpretations than t_2 (any model of t_1 is a model of t_2).*

4 Matching

We call "matching" the process of finding in the knowledge base the formulae analogous to a given target formula. It is inductively defined as follows.

Definition 3 *A* matching problem *is of the form $t \doteq s$ (where t and s are terms or formulae), $S \sqsubseteq S'$ (where S and S' are sets of clauses), $C \leq_s C'$ (where C, C' are clauses), $\mathcal{F} \vee \mathcal{G}$, $\mathcal{F} \wedge \mathcal{G}$, $\forall x.\mathcal{F}$, $\exists x.\mathcal{F}$, where $\mathcal{F}, \mathcal{G}$ are matching problems.*

In order to give a semantics to matching problems we only have to choose the semantics of atomic formulae.

Definition 4 *Let $\mathcal{F}$ be a matching problem and let $\mathcal{I}$ be an interpretation. A substitution σ is a semantic solution of $\mathcal{F}$ w.r.t. $\mathcal{I}$ iff:*

- *$\mathcal{F}$ is of the form $t \doteq s$ (where t, s are terms) and $\mathcal{I} \models t\sigma = s\sigma$.*
- *$\mathcal{F}$ is of the form $\phi \doteq \psi$ (where ϕ, ψ are formulae) and $\mathcal{I} \models \phi\sigma \Leftrightarrow \psi\sigma$.*
- *$\mathcal{F}$ is of the form $S_1 \sqsubseteq S_2$ and there exists a set of clauses S_3 such that $S_3 \sqsubseteq S_2\sigma$ and $\mathcal{I} \models (S_1\sigma \Leftrightarrow S_3)$.*
- *$\mathcal{F}$ is of the form $C \leq_s D$ and there exists a clause C' such that $C' \leq_s D\sigma$ and $\mathcal{I} \models (C\sigma \Leftrightarrow C')$.*

The set of semantic solutions of $\mathcal{P}$ w.r.t. $\mathcal{I}$ is noted $\mathcal{S}_\mathcal{I}(\mathcal{P})$.

Notice that the semantics of matching problem takes into account the *semantics* of the sets of clauses rather than their syntax. This is very important since analogy must focus on the semantic information contained in a theorem rather than in its statement.

Let S_S, S_T be two sets of clauses. Finding a substitution σ such that $S_S\sigma \sqsubseteq S_T$ (unsatisfiable case) or $S_T \sqsubseteq S_S\sigma$ (satisfiable case) is equivalent to finding a solution of the matching problem $S_S \sqsubseteq S_T$ (unsatisfiable case) or $S_T \sqsubseteq S_S$ (satisfiable case). Both cases are noted $\mathcal{P}rob(S_S, S_T)$ in the following.

Obviously finding the solutions of a matching problem is undecidable hence we cannot hope to get a general solution to this problem. In the present paper we only give a set of rules allowing to find the solutions of some matching problem. We *do not specify* here the strategy guiding the application of the rules (several different strategies can be proposed). These rules are *sound:* any solution of the obtained problem is a solution of the initial one. However they are obviously not complete.

Matching problems transformation rules

Clausal transformation rules

$$\begin{array}{lll} P \vee Q & \rightarrow & Q \vee P \\ \forall x, y.P & \rightarrow & \forall y, x.P \\ P & \rightarrow & \neg\neg P \\ P & \rightarrow & P \vee \bot \\ P & \rightarrow & P \vee P \end{array}$$

Higher-order unification rules

The following rules are simply the standard higher-order unification rules (see [Huet, 1975]).

$$\begin{array}{llll} t = t & \rightarrow & \top & \\ f(\bar{t}) = f(\bar{s}) & \rightarrow & \bar{t} = \bar{s} & \\ f(\bar{t}) = g(\bar{s}) & \rightarrow & \bot & \text{if } f \neq g \\ x = t & \rightarrow & \bot & \text{if } x \in t \end{array}$$

Imitation

$$\{X(t_1, \ldots, t_n) = f(s_1, \ldots, s_m)\}$$
$$\rightarrow \quad \{X = \lambda x_1, \ldots, x_m f((X_1 x_1 \ldots x_n), \ldots, (X_m x_1 \ldots x_n))\}$$
$$\cup \bigcup_{i=1}^{m} \{X_i(t_1 \ldots t_n) = s_i\}$$

where X_i $(1 \leq i \leq n)$ are new free variables.

$$\{X(t_1, \ldots, t_n) = x\}$$
$$\rightarrow \qquad \{X = \lambda x_1, \ldots, x_n.x\}$$

Projection

$$\{X(t_1, \ldots, t_n) = f(s_1, \ldots, s_m)\}$$
$$\rightarrow \quad \{X = \lambda x_1, \ldots, x_n.x_i, t_i = f(s_1, \ldots, s_m)\}$$

Replacement

$x = t \wedge \mathcal{P} \quad \rightarrow \quad x = t \wedge \mathcal{P}\{x \rightarrow t\}$ if $x \notin t$

$\vee$-Elimination rule

$P \vee Q \quad \rightarrow P$ If P, Q are two matching problems.

Matching rules

The following rules are new. They allow to eliminate the symbols $\sqsubseteq$ and $\leq_s$ from matching problems.

$\sqsubseteq$-Elimination rules

$$\begin{array}{llll} \sqsubseteq\text{-E}_1 & S_1 \cup S_2 \sqsubseteq S & \rightarrow & S_1 \sqsubseteq S \wedge S_2 \sqsubseteq S \\ \sqsubseteq\text{-E}_2 & \emptyset \sqsubseteq S & \rightarrow & \top \\ \sqsubseteq\text{-E}_3 & S \sqsubseteq \emptyset & \rightarrow & S = \emptyset \\ \sqsubseteq\text{-E}_4 & \{\forall \bar{x}.C\} \sqsubseteq S & \rightarrow & \forall \bar{x}.(\{C\} \sqsubseteq S) \\ \sqsubseteq\text{-E}_5 & \{C\} \sqsubseteq S_1 \cup S_2 & \rightarrow & \{C\} \sqsubseteq S_1 \vee \{C\} \sqsubseteq S_2 \\ \sqsubseteq\text{-E}_6 & \{C\} \sqsubseteq \{D\} & \rightarrow & D \leq_s C \\ \sqsubseteq\text{-E}_7 & \{A \vee B\} \sqsubseteq S & \rightarrow & \{A\} \sqsubseteq S \vee \{B\} \sqsubseteq S \end{array}$$

$\leq_s$-Elimination rules

$$\begin{array}{llll} \leq_s\text{-E}_1 & C \leq_s \forall y.D & \rightarrow & \forall y.(C \leq_s D) \\ \leq_s\text{-E}_2 & \forall x.C \leq_s D & \rightarrow & \exists x.(C \leq_s D) \\ \leq_s\text{-E}_4 & C \leq_s D_1 \vee D_2 & \rightarrow & (C \leq_s C_1) \vee (C \leq_s C_2) \\ \leq_s\text{-E}_5 & \bot \leq_s C & \rightarrow & \top \\ \leq_s\text{-E}_6 & P \leq_s Q & \rightarrow & P = Q \\ \leq_s\text{-E}_7 & C_1 \vee C_2 \leq_s D & \rightarrow & C_1 \leq_s D \wedge C_2 \leq_s D \end{array}$$

Next example illustrates the application of these transformation rules.

Example 3 *Let the source theorem be the set of clauses* $S_S = \{c_1, c_2, c_3\}$ *with* $c_1 \equiv \forall x.P(x) \vee Q(x)$, $c_2 \equiv \neg P(A)$, $c_3 \equiv \neg Q(A)$ *and the target theorem be the set of clauses* $S_T = \{c_4, c_5\}$, *where* $c_4 \equiv \forall y.l(f(y)) \vee l(y)$ *and* $c_5 \equiv \forall y.\neg l(y)$. *We solve the matching problem* $\mathcal{P}$: $S_S \sqsubseteq S_T$ *?* ***Remark:*** P, Q, A *denote* variables *(of order* $2, 2, 1$ *respectively) and* l, f *denote* constant symbols *(of order* $2, 1$*).*

$$\begin{array}{rc} & \mathcal{P} \\ \rightarrow_{\sqsubseteq-E_1} & \{c_1\} \sqsubseteq S_T \wedge \{c_2\} \sqsubseteq S_T \wedge c_3 \sqsubseteq S_T \\ \rightarrow_{\sqsubseteq-E_5} & \{c_1\} \sqsubseteq \{c_4\} \wedge \{c_2\} \sqsubseteq \{c_5\} \wedge c_3 \sqsubseteq \{c_5\} \\ \rightarrow_{\sqsubseteq-E_7} & c_4 \leq_s c_1 \wedge c_5 \leq_s c_2 \wedge c_5 \leq_s c_3 \\ \rightarrow_{\leq_s-E_1} & \forall x.(c_4 \leq_s P(x) \vee Q(x)) \\ & \wedge c_5 \leq_s c_2 \wedge c_5 \leq_s c_3 \\ \rightarrow_{\leq_s-E_2} & \forall x.\exists y.l(f(y)) \vee l(y) \leq_s (P(x) \vee Q(x)) \\ & \wedge \exists y.\neg l(y) \leq_s \neg P(a) \wedge \exists y.\neg l(y) \leq_s \neg Q(a) \\ \rightarrow_{\leq_s-E_6} & \forall x.\exists y.(P(x) \vee Q(x) = l(f(y)) \vee l(y)) \\ & \wedge \exists y.P(A) = l(y) \wedge \exists y.Q(A) = l(y) \\ \rightarrow_{unification} & P = \lambda y.l(f(y)) \wedge Q = \lambda y.l(y) \end{array}$$

We obtain a solution of the initial problem:

$$P \rightarrow \lambda y.l(f(y)), Q \rightarrow \lambda y.l(y)$$

Remark: *of course, other solutions could be obtained.*

5 Lemma generation

In this section we identify a class of matching problems from which the solutions can be obtained automatically and we show how the proof (resp. the counter-example) of the target theorem can be automatically built from the one of the source (non-)theorem. In most cases, analogy will not directly give a proof (counter-example) of the target formula from the source formula. Our method can cope with this case by generating *lemmata* that have to be proved in order to complete the proof/counter-example of the target theorem. This is done by performing a *partial resolution* of matching problems. We do not solve the problem completely, but only *partially.* Formulae that cannot be matched correspond to additional hypotheses that must be proven in order to complete the proof or the counter-example. The derivation can then be completed in two different ways. Either by proving the lemma using a theorem-prover or by a *recursive call* of the matching method (unsatisfiable case). Or by showing that the lemmata are *compatible* with the target set, *i. e.* that the adding of those lemmata to the set preserves satisfiability of the set (satisfiable case).

A matching problem is in normal form iff it is of the form $\mathcal{P} : \bigwedge_{i=1}^{n} x_i = t_i \wedge \mathcal{F} = \bot$, where, for all $i \leq n$, x_i occurs once in $\mathcal{P}$ and $\mathcal{F}$ is a first-order formula. The formula $\mathcal{F}$ corresponds to a formula that *cannot* be matched. We define new rules in order to generate problems in normal form.

Lemma generation rules

$$\begin{array}{lll} \forall y.(C = \bot) & \rightarrow & (\forall y.C) = \bot \\ \exists y.(C = \bot) & \rightarrow & (\exists y.C) = \bot \\ C = \bot \wedge D = \bot & \rightarrow & C \vee D = \bot \\ C = \bot \vee D = \bot & \rightarrow & C \wedge D = \bot \\ C = \top \vee D = \top & \rightarrow & C \vee D = \top \\ C = \top \wedge D = \top & \rightarrow & C \wedge D = \top \\ \mathcal{F} = \bot & \rightarrow & \neg\mathcal{F} = \top \\ S = \emptyset & \rightarrow & S = \top \end{array}$$

We note **matching** the system composed by the **unification and matching rules,** the usual **transformation rules for equational problems** (see for example [Comon and Lescanne, 1989; Lugiez, 1995]) and the **lemma generation** rules.

Theorem 1 *(Soundness) Let $\mathcal{P} \rightarrow_{matching} \mathcal{P}'$. For all interpretation $\mathcal{I}$, $S_{\mathcal{I}}(\mathcal{P}') \subseteq S_{\mathcal{I}}(\mathcal{P})$.*

Theorem 2 *Let S_S be a generalized sequence and S_T a set of clauses. Assume that $\mathcal{P}rob(S_S, S_T) \rightarrow_{matching} \mathcal{P} \wedge \mathcal{F} = \bot$. Let σ be a solution of $\mathcal{P}$.*

1. *If $S_S\sigma$ is unsatisfiable and if for every interpretation $\mathcal{I}$, $\mathcal{I} \models S_T \Rightarrow \mathcal{I} \models \neg\mathcal{F}\sigma$, then S_T is unsatisfiable. On the other hand, there exists an algorithm that computes a refutation of S_T from a refutation of $S_S\sigma$ and $\mathcal{F}\sigma \wedge S_T$.*
2. *If $S_S\sigma$ is satisfiable and if there exists a model $\mathcal{I}$ of $S_S\sigma$ such that $\mathcal{I} \models \neg\mathcal{F}\sigma$, then S_T is satisfiable and $\mathcal{I} \models S_T$.*

In both cases, satisfiability or unsatisfiability of S_T is deduced straightforwardly from the one of S_S. To show that $\mathcal{F}\sigma$ is false in all models of S_T amount to prove that $\mathcal{F}\sigma \wedge S_T$ is unsatisfiable. We can therefore use whatever theorem prover to prove that. The proof can also be obtained with a *recursive call* of the procedure of analogy-proving. If the source formula is satisfiable, one must (according to theorem 2) find a model $\mathcal{I}$ of S_S falsifying $\mathcal{F}\sigma$. In this aim, we try to find an extension of the (partial) model of $S_S\sigma$ falsifying $\mathcal{F}\sigma$. The difference with the unsatisfiable case is that the considered interpretation $\mathcal{I}$ *is unknown a priori* before the beginning of the search. In both cases, the derivation leading to the empty clause (resp. to a model) from the target set of clauses can be rebuilt straightforwardly from the source derivation and the proofs (or counter-example building) corresponding to $\mathcal{F}$.

Example 4 *Consider the formula S_S and S_T of Example 3. Once we have found the solution σ of the problem $S_S \sqsubseteq S_T$, it suffices according to theorem 2 to* instantiate *the proof of S_S in order to obtain a proof of S_T.*

Now assume that we replace clause c_4 in S_T by: $c_6 \equiv \forall y.l(f(y)) \vee l(y) \vee l(g(y))$. In this case the literal $l(g(y))$ cannot *be matched by the algorithm. If we apply the matching problem transformation rules, we get the problem in normal form:*

$$P = \lambda y.l(f(y)) \wedge Q = \lambda y.l(y) \wedge \exists y.l(g(y)) = \bot$$

In order to rebuild the proof of S_T we only have to prove the lemma: $\exists y.\neg l(g(y))$, i.e. to prove that $S_T \cup \{l(g(y))\}$ is not satisfiable. Any theorem prover can do that.

6 Examples

In this section we give two more realistic examples.

Example 5 (Unsatisfiable set of clauses) *Let us consider the problems* SYN310-1 *and* SYN312-1 *of* TPTP *[Suttner and Sutcliffe, 1996].*

Problem S_S is:

$$\begin{array}{l} \neg p(x2, x1, x) \vee p(x, x1, x2) \\ \neg p(x1, x, x2) \vee p(x, x1, x2) \\ \neg p(x, x1, g(x2)) \vee p(x, x1, x2) \\ \neg p(f(x), x1, x2) \vee p(x, x1, x2) \\ \neg p(a, b, c) \\ p(f(g(a)), f(g(b)), f(g(c))) \end{array}$$

S_T is:

$$\begin{array}{l} p'(x, x3, x2) \vee \neg p'(x, x1, x2) \vee \neg p'(x1, x3, x2) \\ p'(x2, x1, x) \vee \neg p'(x, x1, x2) \\ p'(x1, x, x2) \vee \neg p'(x, x1, x2) \\ p'(x, x1, f'(x2)) \vee \neg p'(x, x1, x2) \\ p'(g'(x), x1, x2) \vee \neg p'(x, x1, x2) \\ p'(a', f'(b'), c') \\ p'(f'(b'), d, c') \\ \neg p'(g'(f'(a')), g'(f'(d)), g'(f'(c'))) \end{array}$$

Ss is very easy (it can be proven in 0.17 *seconds by* OTTER 3.0 *[McCune, 1995] with "automatic" mode). ST is more difficult (*OTTER *takes* 536.31 *s to prove it tn automatic mode). We assume that a proof of Ss is known and we try to build automatically a proof of ST* by analogy *with the one of Ss*.

We apply our algorithm in order to solve the problem $S_S \sqsubseteq S_T$? No general solution can be found. Indeed S_T is not a direct instance of S_S. However we get the following partial solution: $\{p = \lambda x_1, x_2, x_3.\neg p'(x_3, x_2, x_1), g = \lambda x.f'(x), f = \lambda x.g'(x), a = a', b = b', c = c', d = d'\}$ with the lemmata:

$$\{p'(a, b, c), \forall x_1, x_2, x_3.p'(x_1, x_2, x_3) \vee \neg p'(x_1, x_3, x_1)\}$$

These two formulae are clauses belonging to S_S that cannot be matched by S_T.

*According to Theorem 2 it only remains to prove the lemmata. The first one can be proven (very easily) by adding the negation of $p'(a, b, c)$ to S_T (*OTTER *takes 0.37 s to prove it).*

*In order to transform the second lemma into a set of clauses we need to use skolemization. We get $\{\neg p'(a_1, a_2, a_3), p'(a_1, a_3, a_2)\}$. This set is added to S_T. The corresponding set can again be proven very easily (*OTTER *takes 0.18 s).*

This example suggests that reasoning by analogy can enhance significantly the power of theorem-provers. Indeed the total amount of time required to prove the lemmata is 0.55 *s and the time required to prove the whole theorem without using analogy is* 536.3*Is. Please note that the proofs of the lemmata does not require any user-interaction. However the matching process is not yet implemented, hence must be done by hand.*

Example 6 (Satisfiable set of clauses) *Let Ss be the following satisfiable set of clauses (adapted from [Church, 1940]).*

$$\begin{array}{ll} \neg f(x) \vee \neg p(x) & p(x) \vee f(x) \\ \neg f(y(x)) \vee \neg p(x) & p(x) \vee f(y(x)) \end{array}$$

The finite model builder FMC *developed in our inference laboratory builds the following model (in 0.01 s).*

```
Model:
f(0)= true     f(1)= false     p(0)= false
p(1)= true     y(0)=1          y(1)=0
```

Now, let S_T be the following set of clauses $\{\neg f'(x) \vee \neg(f'(y'(x))), f'(x) \vee f'(y'(x)))\}$. Here, if we replace f', y' by f, y, we have obviously $S_S \models S_T$. Hence we try to build a model of S_T by analogy with the one of S_S. We first solve the matching problem: $\mathcal{P}rob(S_S, S_T) = S_T \sqsubseteq S_S$. We obtain the following solution: $y = \lambda x.y'(x), f = \lambda x.f'(x)$, with the lemma: $\forall x.p(x) \vee \neg p(x)$. According to theorem 2 it only remains to check if the lemma holds in the model of S_S. This is obviously the case since $\forall x.p(x) \vee \neg p(x)$ is a tautology. Therefore the following interpretation is a model of S_T: $\{f'(0) = true, f'(1) = false, y'(0) = 1, y'(1) = 0\}$.

7 Discussion and perspectives

We have presented a calculus for the discovery and handling of analogy between sets of clauses. It is able in particular to handle *partial* analogy between statements by generating *lemmata.* Our work is the first step toward a system for simultaneous search for refutations and models using analogy and abductive reasoning. Main lines of future research are:

- Define strategies or heuristics for our calculus *matching* and study their properties (termination, efficiency, completeness for some classes of problems etc.).
- Extend this approach to first-order formulae (not in clausal form). In particular we have to give a new definition of the generalization order.

A comparative analysis with the lemmata produced by other techniques such as [Kolbe and Walther, 1995] will be performed in the future. We are currently working on the implementation of the calculus and on the definition of the knowledge base.

Acknowledgements

We thank Ricardo Caferra and the anonymous referees for their pertinent and precise comments.

References

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

[Bourely et *al,* 1994] Ch. Bourely, R. Caferra, and N. Peltier. A method for building models automatically. Experiments with an extension of Otter. In *Proceedings of CADE-12,* pages 72-86. Springer, 1994. LNAI 814.

[Bourely *et al,* 1996] C. Bourely, G. Defourneaux, and N. Peltier. Building proofs or counterexamples by analogy in a resolution framework. In *Proceedings of JELIA 96, LNIA 1126,* pages 34-49. Springer, LNAI, 1996.

[Boy de la Tour and Caferra, 1987] Th. Boy de la Tour and R Caferra. Proof analogy in interactive theorem proving: A method to express and use it via second order pattern matching. In *Proceedings of AAAI 81,* pages 95-99. Morgan Kaufmann, 1987.

[Church, 1940] A. Church. A formulation of the simple theory of types. *Journal of Symbolie Logic,* 5(l):56-68, 1940.

[Comon and Lescanne, 1989] H. Comon and P. Lescanne. Equational problems and disunification. *Journal of Symbolic Computation,* 7:371-475, 1989.

[Console *et al.,* 1991] L. Console, D. Theseider Dupre, and P. Torasso. On the Relationship between Abduction and Deduction. *Journal of Logic and Computation,* 1(5):661-690, 1991.

[Defourneaux and Peltier, 1997] G. Defourneaux and N. Peltier. Partial matching for analogy discovery in proofs and counter-examples. In Springer, editor, *Proceedings of CADE 14,* 1997.

[Hall, 1989] R.P. Hall. Computational approaches to analogical reasoning: A comparative analysis. *Artificial Intelligence,* pages 39-120, 1989.

[Hartshorne *et al,]* Hartshorne, Weiss, and Burks. *Collected Papers of C.S. Peirce (1930-1958).* Harward U. Press.

[Hobbs *et al,* 1993] J. Hobbs, M. Stickel, D. Appelt, and P. Martin. Interpretation as abduction. *Artificial Intelligence,* 63:69-142, 1993.

[Huet, 1975] G. Huet. A unification algorithm for typed A-calculus. *Theorical Computer Science,* 1:27-57, 1975.

[Inoue *et al,* 1993] K. Inane, Y. Ohta, R. Hasegawa, and M. Nakashima. Bottom-up abduction by model generation. In *Proc. IJCAI-93,* volume 1, pages 102-108, Morgan Kaufmann, 1993.

[Kolbe and Walther, 1995] Th. Kolbe and Ch. Walther. Second-order matching modulo evaluation - A technique for reusing proofs. In Chris S. Mellish, editor, *Proceedings of IJCAI 95,* pages 190 195. IJCAI, Morgan Kaufmann, 1995.

[Lalande, 1980] A. Lalande. *Vocabulaire technique et Critique de la Philosophic.* Presses Universitaires de France, 1980.

[Lugiez, 1995] D. Lugiez. Positive and negative results for higher-order disunification. *Journal of Symbolic Computation,* 1995.

[McCune, 1995] W. McCune. *Otter 3.0 Reference Manual and Guide.* Argonne National Laboratory, August 1995. Revision A.

[Peirce, 1955] C.S. Peirce. *Philosophical Writings of PEIRCE,* chapter Abduction and induction, pages 150-156. Dover Books, 1955.

[Plaisted, 1981] D.A. Plaisted. Theorem proving with abstraction. *Artificial Intelligence,* 16:47-108, 1981.

[Polya, 1973] G. Polya. *How to Solve It, a New Aspect of Mathematical Method.* Princeton University Press. Second Edition, 1973.

[Pople, 1973] H. Pople. On the Mechanization of Abductive Logic. In *Proc. of the IJCAAI 73,* pages 147-152, 1973.

[Slaney, 1993] J. Slaney. SCOTT: a model-guided theorem prover. In *Proceedings IJCAI-93,* volume 1, pages 109-114. Morgan Kaufmann, 1993.

[Suttner and SutclifTe, 1996] Ch. Suttner and G.SutclifTe. The TPTP problem library. Technical report, TU Munchen / James Cook University, 1996. V-1.2.1.

[Zhang and Zhang, 1995] J. Zhang and H. Zhang. SEM: a system for enumerating models. In *Proc. IJCAI-95,* volume 1, pages 298-303. Morgan Kaufmann, 1995.