DETECING AND CORRECTING FAULTY CONJECTURES

MOUSSA DEMBA KHALED BSAIES

Faculty of Sciences of Tunis
Department of Computer Sciences
Tunisia

ABSTRACT: *We present a method for patching faulty conjectures and program diagnosis in automatic theorem proving during a proof attempt. In this paper we discuss correctness of the approach and we show the application of corrective predicates to program diagnosis when conjectures are unprovable. The approach is implemented in an interactive theorem prover called SPES.*

Keywords: *Corrective predicates, Program synthesis, Automatic theorem proving, Abduction, Folding/unfolding.*

1 INTRODUCTION

We present a method for patching faulty conjectures in automatic theorem proving. The conjectures we are interested in here are implicative formulas that are of the following form : $\forall x\ \phi(x) : \forall x\ \exists y\ \Gamma(x, y) \leftarrow \Delta(x)$. A faulty conjecture is a statement $\forall x\ \phi(x)$, which is not provable in some given theory $\mathcal{T}$, defining all the predicates occurring in ϕ, i.e, $\mathcal{M}(\mathcal{T}) \not\models \forall x\ \phi(x)$ where $\mathcal{M}(\mathcal{T})$ means the least Herbrand model of $\mathcal{T}$ but it would be if enough conditions, say P, were assumed to hold, i.e., $\mathcal{M}(\mathcal{T} \cup \mathcal{P}) \models \forall x\ (\phi(x) \leftarrow P(x))$, where $\mathcal{P}$ is the definition of P. The missing hypothesis P is called a (candidate) corrective predicate for ϕ. To construct P we use the abduction mechanism. According to Peirce [9], abduction is the process of hypothesis formation and is described as follows: Given ϕ and $\phi \leftarrow P$, hypothesize P as a possible justification of the formula ϕ. In the case of automatic theorem proving we are sometimes faced with unprovable conjectures. A classical theorem prover will do nothing more but signals simply this failure. However, when a proof attempt fails either the formula is false or the program has a bug or the attempted proof is insufficient. For example, if we have to prove the (true) formula about sorting linear lists

$$sort(sort(x)) = sort(x) \tag{1}$$

me have to (over)generalize it to the faulty conjecture [10]

$$sort(y) = y \tag{2}$$

A theorem prover will do nothing more but reject it without explaining and identifying the source of the error. A possible solution to this problem is to try to modify the conjecture (2) back into a theorem, for example by adding a condition on the list y as follows:

$$(sort(y) = y) \leftarrow ordered(y) \tag{3}$$

where $ordered(y)$ means that y is an ordered list. The main problem is to know what condition has to be added. Now to verify (1) it is sufficient to prove

$$ordered(sort(x)) \tag{4}$$

The predicate $ordered$ is then a corrective predicate for the conjecture (2). One of the advantages of this technique is that we can continue with the use of generalization instead of having to find an alternative approach. This problem can also be seen as program synthesis from incomplete specification [6]. For example, giving only the definition of the function $sort$, the conjecture (3) cannot be proved, the definition of the function $ordered$ has to be constructed somehow.

In this paper, our aim is to turn an unprovable conjecture into a theorem by synthesizing the missing hypothesis. The missing hypothesis is represented by a (corrective) predicate, say P, defined by some program $\mathcal{P}$. $\mathcal{P}$ is obtained by exploiting information derived from a failed proof attempt of ϕ. In order to synthesize $\mathcal{P}$, we have proposed in [2] an extension of the program synthesis method of Fribourg [4]. Fribourg considers the proof system named "extended execution" of [8] and a restricted form of structural induction. Note that his method extracts programs from true conjectures and does not deal with faulty conjectures.

The rest of the paper is organized as follows : In section 2 we describe our proof system. Section 3 presents

a general outline of our approach. We present in section 4 significant examples for program diagnosis. The last section is a conclusion and limitations of the proposed approach.

1.1 PRELIMINARIES

Notation 1 *Throughout the paper existentially quantified variables are distinguished from universal variables by giving them uppercase names.* Γ, Δ *and* Λ *denote conjunctions of atoms;* ϕ *and* π *denote implicative formulas;* A *and* B *denote atoms, and* θ *and* σ *denote substitutions.* mgu *means most general unifier and* $< \pi_i \mid Pi >$ *denotes the formula* π_i *and its corrective predicate* Pi*. Henceforth, the term formula (resp. program) will often be used as an abbreviation for implicative formula (resp. definite logic program).*

Definition 2 (Partial correctness) *Let* $\phi : \Gamma(x, Y) \leftarrow \Delta(x)$ *be an implicative formula whose predicates are defined by the program* $\mathcal{T}$*. Let* $\mathcal{P}$ *be a program defining a corrective predicate* P*. The program* $\mathcal{P}$ *is partially correct for* $\mathcal{T}$ *with respect to* ϕ *iff* $\mathcal{M}(\mathcal{T} \cup \mathcal{P}) \models (\Gamma(x, y) \leftarrow \Delta(x), P(x, y))$.

Definition 3 *Suppose the formulas* $< \pi_1|P1 >, ..., < \pi_k|Pk >$ *are obtained from the formula* $< \pi_0|P0 >$ *by the application of one-step deduction rule* R*. A procedure associated with* R *is a program* $\mathcal{Q}_\mathcal{R}$ *which defines* $P0$ *in terms of* $P1$*,...,*Pk *and in terms of* $P0$ *if* $\mathcal{Q}_\mathcal{R}$ *is recursive.*

Definition 4 *Let* P *and* Q *be two corrective predicates for some conjecture* ϕ*.* (i) P *is more plausible than* Q *iff for all* x $P(x) \leftarrow Q(x)$ *holds.* (ii) *if for any* Q*,* P *is more plausible than* Q*, then* P *is a maximal corrective predicate for* ϕ*.*

By the definition (4), we have the proposition.

Proposition 5 *A corrective predicate* P *for the formula* ϕ *is maximal if the formula* $\forall x \ (P(x) \leftarrow \phi(x))$ *holds.*

1.2 AN INTUITIVE PRESENTATION

Let us consider the program defining the predicates $plus$ and nat:

$$\mathcal{PLUS} \begin{cases} plus(0, x, x) & \leftarrow \\ plus(s(x), y, s(z)) & \leftarrow plus(x, y, z) \\ nat(0) & \leftarrow \\ nat(s(x)) & \leftarrow nat(x) \end{cases}$$

where s and 0 are constructors. The atom $plus(x, y, z)$ is true if $z = x + y$ and $nat(x)$ is true if x is a natural number. Let us consider the following specification for the subtraction function in natural numbers : given two natural numbers v and w, find X such that $v + X = w$. To this specification corresponds the implicative formula :

$$plus(v, X, w) \leftarrow nat(v), nat(w) \quad \mid P(v, X, w) \quad (5)$$

which is false, as we discover while attempting to prove it, for example there is no X verifying $2 + X = 1$. Nevertheless, there are particular values for the universally quantified variables for which the formula (5) is true. We are then looking for an hypothesis P(v,X,w) such that the formula

$$plus(v, X, w) \leftarrow nat(v), nat(w), P(v, X, w)$$

be a theorem. To do that, we try to prove (5) and to keep track of substitutions on P. After some unfolding steps on (5) w.r.t the atom $nat(v)$, we get the following formulas (without quantifiers) :

$$\begin{aligned} plus(0, X, w) &\leftarrow nat(w) \ \ |P(0, X, w) \\ plus(s(v), X, w) &\leftarrow nat(v), nat(w) \\ &\qquad |P(s(v), X, w) \end{aligned} \quad (6)$$

and an unfolding step on (6) w.r.t the atom $nat(w)$ yields :

$$plus(0, X, w) \leftarrow nat(w) \quad |P(0, X, w) \quad (7)$$

$$plus(s(v), X, 0) \leftarrow nat(v) \ \ |P(s(v), X, 0) \quad (8)$$

$$\begin{aligned} plus(s(v), X, s(w)) &\leftarrow nat(v), nat(w) \\ &\qquad |P(s(v), X, s(w)) \end{aligned} \quad (9)$$

- the formula (7) can be simplified into true with the existential substitution $\{X/w\}$; and the corresponding corrective predicate is the unit clause $P(0, w, w) \leftarrow$.
- the formula (8) is fully false thus the corresponding corrective predicate is set to false, i.e. $P(s(v), X, 0) = false$.
- by the definition of $plus$, the formula (9) can be transformed into the formula

$$\begin{aligned} plus(v, X, w) &\leftarrow nat(v), nat(w) \\ &\qquad \mid P(s(v), X, s(w)) \end{aligned} \quad (10)$$

- Finally, the formula (5), the induction hypothesis, is an instance of the formula (10). An obvious folding step between (10) and (5) allows us to yield the formula true, and the recursive clause for P is generated: $P(s(v), X, s(w)) \leftarrow P(v, X, w)$.

We have then synthesized a definition, say $\mathcal{P}$, of P :

$$P(0, w, w) \leftarrow \quad (11)$$

$$P(s(v), X, s(w)) \leftarrow P(v, X, w) \quad (12)$$

Note that in the recursive clause (12) the existential variable X remains unchanged. We can eliminate this variable. For instance, a truncation of P w.r.t. its second argument yields the following program $\mathcal{P}'$:

$$P'(0, w) \leftarrow$$
$$P'(s(v), s(w)) \leftarrow P'(v, w)$$

One can remark that P' is exactly the relation $\leq$ over natural numbers, and we have

$\mathcal{M}(\mathcal{PLUS} \cup \mathcal{P}') \models (plus(v, X, w) \leftarrow P'(v, w))$

Therefore P' is a corrective predicate for (5), i.e., (5) is true if $P'(v, w)$ holds. Strategies of projection are discussed in [4]. The predicate P' is also maximal :

$\mathcal{M}(\mathcal{PLUS} \cup \mathcal{P}') \models (P'(v, w) \leftarrow plus(v, x, w))$

This explanation is significant (in the case of failure) for the user because it enables him to know the source of the error and to fix it. It is also different from explanations by yes/no provided by classical theorem provers.

2 DESCRIPTION OF THE PROOF SYSTEM

The system presented here uses deduction rules, that include unfolding and folding, that allow us to prove implicative formulas. Intuitively, unfolding is an extension of SLD-resolution and folding applies the induction hypotheses. Indeed, whereas an unfold step replaces a term that "matches" the conclusion of a definition in the program by the corresponding hypothesis, a folding right (resp. left) step replaces a conjunction of atoms that match the hypothesis (resp. conclusion) of an induction hypothesis by the corresponding conclusion (resp. hypothesis). Each inference rule is associated with a procedure construction of corrective predicates. The application of an inference rule on a formula π generates a finite set of formulas π_i, i=1,...,k, such that π follows from the π_i's in the least Herbrand model of the program under consideration. The process is iterated until all the formulas newly generated are trivial. We present the main rules of our proof system when applied to implicative formulas and define the associated corrective predicates.

Definition 6 (Negation as failure inference (nfi)) *Let $\mathcal{P}$ be a program, $\pi_0 : \Gamma \leftarrow \Delta, A$ a formula and $C = \{c_1, \ldots, c_k\}$ the set of clauses of $\mathcal{P}$ such that $c_i : B_i \leftarrow \Delta_i$. Suppose that $\theta_i = mgu(B_i, A)$. Then nfi on π_0 w.r.t to the atom A yields a conjunction of k formulas :*

$$< \pi_0 : (\Gamma \leftarrow \Delta, A) \mid P0 >$$
$$\downarrow \texttt{nfi}$$
$$< \pi_i : (\Gamma \leftarrow \Delta, \Delta_i)\theta_i \mid Pi >_{i=1,\ldots,k}$$

If for all i=1,...,k Pi is a corrective predicate for π_i (i.e. $\pi_i \leftarrow Pi$ holds), then $P0$ is a corrective predicate for π_0. Hence $Q_{nfi} = \{P0\theta_i \leftarrow Pi\}_{i=1,\ldots,k}$.

Example 7 *Consider the formula π_0 :*
$plus(u, v, w) \leftarrow plus(v, u, w)$ and the corresponding corrective predicate $P0(u, v, w)$.
The application of nfi on π_0 with $\theta_1 = \{v/0, u/x, w/x\}$ and $\theta_2 = \{v/s(x), u/y, w/s(z)\}$ yields the following two formulas:
$\pi_1 : plus(x, 0, x) \leftarrow \mid P1(x)$
$\pi_2 : plus(y, s(x), s(z)) \leftarrow plus(x, y, z) \mid P2(y, x, z)$
and Q_{nfi} defines $P0$ in terms of $P1$ and $P2$ as follows:

$$P0(x, 0, x) \leftarrow P1(x)$$
$$P0(y, s(x), s(z)) \leftarrow P2(y, x, z).$$

Next we have to synthesize the definitions of $P1$ and $P2$ by proving π_1 and π_2.

Definition 8 (Definite clause inference (dci)) *Let $\mathcal{P}$ be a logic program and c a definite clause in $\mathcal{P}$ of the form $B \leftarrow \Delta'$. Let π be an implicative formula of the form $\Gamma, A \leftarrow \Delta$ and suppose that A is unifiable with B by an existential substitution[1] θ, i.e., θ=$mgu(B, A)$. The rule of dci applied on π w.r.t the atom A generates the singleton $\{\pi'\}$:*

$$< \pi : (\Gamma, A \leftarrow \Delta) \mid P >$$
$$\downarrow \texttt{dci}$$
$$< \pi' : ((\Gamma, \Delta')\theta \leftarrow \Delta) \mid P' >$$

and $Q_{dci} = \{P\theta \leftarrow P'\}$.

Example 9 *Consider the formula*
$\pi : plus(s(u), s(v), s^2(w)) \leftarrow plus(u, v, w)$
and $P(u, v, w)$ the corresponding corrective predicate. Then dci on π yields:
$\pi' : plus(u, s(v), s(w)) \leftarrow plus(u, v, w) | P'(u, v, w)$
and the clause $P(u, v, w) \leftarrow P'(u, v, w)$ that defines P in terms of P' is generated.

We define the folding rules that apply induction hypotheses.

Definition 10 (Cut right (cutr)) *Let $\pi_1 : \Gamma \leftarrow \Delta_1, \Delta_2$ and $\pi_0 : \Lambda \leftarrow \Pi$ be two formulas satisfying the following conditions : (i) θ is a substitution such that $\Pi\theta = \Delta_1$, (ii) for any local variable x in Π, $x\theta$ is a variable and does not occur other than in $\Pi\theta$, and (iii) θ replaces different local variables in Π with different local variables in Δ_1. Then $cutr$ on π_1 using*

[1] θ substitutes only existential variables of A.

π_0 yields $\{\pi_2\}$:

$$\begin{array}{c} < \pi_0 : (\Lambda \leftarrow \Pi) \mid P0 > \\ \downarrow \\ < \pi_1 : (\Gamma \leftarrow \Delta_1, \Delta_2) \mid P1 > \\ \downarrow \texttt{cutr} \\ < \pi_2 : (\Gamma \leftarrow \Lambda\theta, \Delta_2) \mid P2 > \end{array}$$

If $P2$ (resp. $P0$) is a corrective predicate for π_2 (resp. π_0) then $P1$ is a corrective predicate for π_1. Hence $Q_{cutr} = \{P1 \leftarrow P0\theta, P2\}$ that defines $P1$ in terms of $P0$ and $P2$.

Example 11 *Going back to the example 7, one can remark that the right hand side of π_2 is an instance of the right hand side of π_0 with the substitution $\theta = \{v/x, u/y, w/z\}$. We can therefore apply the rule of cutr on π_2 using π_0, and we get the formula:*
$\pi_3 : plus(y, s(x), s(z)) \leftarrow plus(y, x, z) | P3(y, x, z)$ and the definite clause $P2(y, x, z) \leftarrow P0(y, x, z), P3(y, x, z)$ is generated.

Definition 12 (Cut left (cutl)) *Let $\pi_1 : \Gamma_1, \Gamma_2 \leftarrow \Delta$ and $\pi_0 : \Lambda \leftarrow \Pi$ be two formulas satisfying the following conditions: (i) θ is a substitution such that $\Lambda\theta = \Gamma_1$, (ii) for any local variable z in Λ, $z\theta$ is a variable and does not occur other than in $\Lambda\theta$, and (iii) θ replaces different local variables in Λ with different local variables in Γ_1. Then the application of cutl on π_1 using π_0 yields $\{\pi_2\}$:*

$$\begin{array}{c} < \pi_0 : (\Lambda \leftarrow \Pi) \mid P0 > \\ \downarrow \\ < \pi_1 : (\Gamma_1, \Gamma_2 \leftarrow \Delta) \mid P1 > \\ \downarrow \texttt{cutl} \\ < \pi_2 : (\Pi\theta, \Gamma_2 \leftarrow \Delta) \mid P2 > \end{array}$$

and $Q_{cutl} = \{P1 \leftarrow P0\theta, P2\}$.

Definition 13 (Simplification (simp)) *Let $\pi : A, \Gamma \leftarrow B, \Delta$ be a formula such that there exists θ satisfying $A\theta = B$ and θ substitutes only existential variables of A. Then simp on π yields the singleton $\{\pi'\}$:*

$$\begin{array}{c} < \pi : (A, \Gamma \leftarrow B, \Delta) \mid P > \\ \downarrow \texttt{simp} \\ < \pi' : (\Gamma\theta \leftarrow \Delta) \mid P' > \end{array}$$

and P is defined in terms of P' by
$Q_{simp} = \{P\theta \leftarrow P'\}$.

Example 14 *Consider the formula*
$\pi : plus(x, y, X), plus(X, z, V) \leftarrow plus(x, y, t) \mid P(x, y, X, z, V, t)$
With the existential substitution $\theta = \{X/t\}$, π can be simplified into
$\pi' : plus(t, z, V) \leftarrow \mid P'(t, z, V)$
The clause $P(x, y, t, z, V, t) \leftarrow P'(t, z, V)$ is then generated.

Definition 15 (Postulate (post)) *Let $\pi : \Gamma \leftarrow$ be an implicative formula and P be a corrective predicate associated with π. Then the application of the rule of postulate on π yields the formula $true$.*

$$\begin{array}{c} < \Gamma \leftarrow \mid P > \\ \downarrow \texttt{post} \\ < true \mid true > \end{array}$$

and $Q_{post} = \{P \leftarrow \Gamma\}$, i.e. P is true if Γ holds. One can remark that $\leftarrow \Gamma$ is a lemma.

Example 16 *In the example (7), to complete the proof of π_1 we can postulate $plus(x, 0, x)$, and we obtain the corrective clause $P1(x) \leftarrow plus(x, 0, x)$.*

Definition 17 (Failure (fail)) *Let $\mathcal{P}$ be a program, $\pi : \Gamma \leftarrow \Delta$ a formula and P a corrective predicate for π. If Γ contains an atom that is not unifiable with all the clause heads in $\mathcal{P}$ and that $\mathcal{M}(\mathcal{P}) \models \Delta$ then the rule of failure is applied and yields the formula $false$:*

$$\begin{array}{c} < \Gamma \leftarrow \Delta \mid P > \\ \downarrow \texttt{fail} \\ < false \mid false > \end{array}$$

and Q_{fail} is the empty set. This rule allows us to detect totally false conjectures.

Example 18 *Suppose we have to prove the formula : $\pi : plus(s(v), U, 0) \leftarrow nat(v)$. π is false because in one hand the left hand side cannot be reduced using the program $\mathcal{PLUS}$ and on the other hand we have $\mathcal{M}(\mathcal{PLUS}) \models nat(x)$. The formula $plus(s(v), U, 0) \leftarrow nat(v)$ is then false and the corresponding corrective predicate is set to $false$.*

Proposition 19 ([4, 2]) *The procedures Q_{nfi}, Q_{dci}, Q_{cutr}, Q_{cutl}, Q_{simp}, Q_{post} and Q_{fail} preserve partial correctness.*

3 A GENERAL OUTLINE OF THE METHOD

To illustrate the main idea behind our method we present it with non-trivial examples. Let's define first the notion of counterexample that allows us to detect incorrect conjectures and to exhibit counterexamples.

Definition 20 (Counterexample) *Let $\mathcal{P}$ be a program. An example of an implicative formula $\Gamma \leftarrow \Delta$ is a substitution σ such that: (i) all the universally quantified variables in the formula are instantiated to ground terms by σ, i.e., $\Delta\sigma$ is ground, and (ii) $\mathcal{M}(\mathcal{P}) \models \Delta\sigma$.*
A counterexample is an example σ but $\mathcal{M}(\mathcal{P}) \not\models \Gamma\sigma$.

For example, if we consider the formula

$$plus(v, X, w) \leftarrow nat(v), nat(w)$$

then the substitution $\sigma = \{v/s(0), w/0\}$ is a counterexample, because $nat(s(0))$ and $nat(0)$ both hold in $\mathcal{M}(\mathcal{PLUS})$ and $\mathcal{M}(\mathcal{PLUS}) \not\models plus(s(0), X, 0)$.

Theorem 21 (Propagation of a counterexample) *If there is a counterexample on a node N in a proof tree, there is at least one successor node of N on which there is a counterexample (assuming that the successor is not obtained by the rule of postulate).*

Proof 22 *We have cases according to what deduction rule is applied. Let σ be a counterexample on the node N and σ' be the one on a successor node.*

- *If $\Gamma \leftarrow \Delta$ is the formula $false \leftarrow true$, then the rule of failure is applied and the son is marked false. if $\Gamma \leftarrow \Delta$ is the formula $true \leftarrow \Delta$, then we do not have σ.*
- *The last rule is nfi: σ is counterexample, i.e., $\mathcal{M}(\mathcal{P}) \models \Delta\sigma$ and $\mathcal{M}(\mathcal{P}) \not\models \Gamma\sigma$. Since nfi is equivalence preserving [7], there is $i \in [1, k]$ such that $\mathcal{M}(\mathcal{P}) \not\models (\Gamma \leftarrow \Delta_i, \Delta_2)\theta_i\sigma$. We can always find a ground substitution σ'' such that $(\Delta_i, \Delta_2)\theta_i\sigma\sigma''$ is a ground term and we have $\mathcal{M}(\mathcal{P}) \models (\Delta_i, \Delta_2)\theta_i\sigma\sigma''$ and $\mathcal{M}(\mathcal{P}) \not\models \Gamma\theta_i\sigma\sigma''$. $\sigma' = \sigma\sigma''$ is then a counterexample of $(\Gamma \leftarrow \Delta_i, \Delta_2)\theta_i$.*
- *The last rule is dci: obviously $\sigma = \sigma'$.*
- *The last rule is $cutr$: suppose that $\Delta \equiv \Delta_1, \Delta_2$ and a substitution θ such that $\Pi\theta \equiv \Delta_1$. σ is a counterexample, then $\mathcal{M}(\mathcal{P}) \models (\Delta_1, \Delta_2)\sigma$, and $\mathcal{M}(\mathcal{P}) \not\models \Gamma\sigma$. Whe have two scenarios : (i) the induction hypothesis is true, that is $\mathcal{M}(\mathcal{P}) \models (\Lambda \leftarrow \Pi)\theta\sigma$. Let σ'' be a ground substitution such that $\mathcal{M}(\mathcal{P}) \models \Lambda\theta\sigma\sigma''$, then we have $\mathcal{M}(\mathcal{P}) \not\models (\Gamma \leftarrow \Lambda\theta, \Delta_2)\sigma\sigma''$. Therefore $\sigma' = \sigma\sigma''$ is a counterexample of $\Gamma \leftarrow \Lambda\theta, \Delta_2$. (ii) $\mathcal{M}(\mathcal{P}) \not\models (\Lambda \leftarrow \Pi)\theta\sigma$. Therefore $\sigma' = \theta\sigma$ is a counterexample of $\Lambda \leftarrow \Pi$.*
- *The last rule is $cutl$: the proof is similar to $cutr$.*
- *The last rule is $simp$: given $\Delta \equiv \Delta', B$ and $\Gamma \equiv \Gamma', A$ such that $A\theta \equiv B$ for a given substitution θ. As σ is a counterexample we have $\mathcal{M}(\mathcal{P}) \models (\Delta', B)\sigma$ and $\mathcal{M}(\mathcal{P}) \not\models (\Gamma', A)\theta\sigma$. Since $\mathcal{M}(\mathcal{P}) \models B\sigma$ then $\mathcal{M}(\mathcal{P}) \models A\theta\sigma$. In other words $\mathcal{M}(\mathcal{P}) \not\models \Gamma'\theta\sigma$, and σ is a counterexample of the formula $\Gamma'\theta \leftarrow \Delta'$.*

Example 23 *We describe our method by a non trivial example. Consider the conjecture (2), but in an implicative form:*

$$sort(x, x) \leftarrow list(x) \tag{13}$$

Clearly, this conjecture is true only if the list x is $ordered$. The missing hypothesis is then $ordered(x)$ and we want to synthesize this information via a corrective predicate. To do that we consider the program $\mathcal{SORT}$ where the predicate $insert$ inserts an element in a sorted list and $inf(x, y)$ is true if $x \leq y$.

$$\begin{cases} sort([], []) \leftarrow \\ sort([a|x], z) \leftarrow sort(x, y), insert(a, y, z) \\ insert(a, [], [a]) \leftarrow \\ insert(a, [b|x], [a, b|x]) \leftarrow inf(a, b) \\ insert(a, [b|x], [b|y]) \leftarrow inf(b, a), insert(a, x, y) \\ inf(o, x) \leftarrow \\ inf(s(x), s(y)) \leftarrow inf(x, y) \end{cases}$$

The Figure (1) shows the proof tree of (13) and the Figure (2) shows the proof in the system SPES. Finally, the synthesized program is:

$$\begin{array}{lll} (1)\ P0([]) & \leftarrow P1() \\ (2)\ P1() & \leftarrow \\ (3)\ P0([a|x]) & \leftarrow P2(a, x) \\ (4)\ P2(a, x) & \leftarrow P3(a, x) \\ (5)\ P3(a, x) & \leftarrow P0(x), P4(a, x) \\ (6)\ P4(a, x) & \leftarrow P5(a, x) \\ (7)\ P5(a, []) & \leftarrow P6(a) \\ (8)\ P5(a, [b|x]) & \leftarrow P7(a, b, x) \\ (9)\ P7(a, b, x) & \leftarrow P8(a, b, x) \\ (10)\ P8(a, b, x) & \leftarrow P9(a, b) \\ (11)\ P6(a) & \leftarrow \\ (12)\ P9(a, b) & \leftarrow inf(a, b) \end{array}$$

In the next step we simplify the intermediate predicates (predicates defined only by one clause). By unfolding [13], this program is transformed into the equivalent one :

$$\mathcal{Q} \begin{cases} P0([]) & \leftarrow \\ P0([a]) & \leftarrow \\ P0([a, b|x]) & \leftarrow P0([b|x]), inf(a, b) \end{cases}$$

When analyzing this program, one can remark that the predicate $P0$ is the predicate $ordered$ and we have the correctness property :

$\mathcal{M}(\mathcal{SORT} \cup \mathcal{Q}) \models (sort(l,l) \leftarrow P0(l))$
Moreover, the predicate $P0$ *is maximal because we have* $\mathcal{M}(\mathcal{SORT} \cup \mathcal{Q}) \models (P0(l) \leftarrow sort(l,l))$

4 CONCLUSION

4.1 RELATED WORKS

Franŏvà et al. [3] have investigated the problem of patching faulty conjectures and proposed a method called $PreS$. No formal system is clearly defined and no system is described.
Protzen [10] proposed a method which allows to synthesize a corrective predicate during the proof attempt of a faulty conjecture. His approach is similar to ours, but uses rewriting rules and induction rules, he gives some correctness results and dealt with universally quantified formulas.
Also Monroy et al. have introduced a method for correcting faulty conjectures[12]. However, they only partially deal with the problem of correcting faults. For example, they cannot build a corrective predicate, only identify it as long as it is present in the working theory. Monroy proposed in [11] another method that consists of a collection of construction commands and is able to synthesize corrective predicates. His approach is also based on the proofs-as-programs paradigm and guarantees the correction and the termination. There is a similarity between his predicates and ours, but his predicates are refined incrementally during the proof process. Monroy poses the problem of automation of the process and suggests to use a proof planning approach. His technique deal with universally quantified formulas. None of these methods deals with true conjectures.

4.2 Final Remarks

We have presented a method for patching faulty conjectures by synthesizing definite programs. The approach presented is integrated in the interactive theorem prover SPES [1] using the functional language OCaml. So if the system is used to prove a faulty conjecture, it will on the fly build a candidate corrective predicate. Our approach can be used as a machine learning technique because it allows in one hand to add clauses to the theory until every positive example is covered and in the other hand to discard clauses that contain negative examples.
An important result of this work is that I have been able to integrate abductive reasoning in an inductive theorem prover and to learn logic programs. The main limitation of this approach is the combinatorial explosion of proof trees.
Among interesting topics which we have not discussed in this paper is how to reduce the degree of non-determinism occurring in the proof process. A possible solution to this problem is to introduce an order between atoms, and allow on a proof tree only conjectures reduced according to this order.

REFERENCES

[1] F. Alexandre, J.P. Finance and A. Quéré. SPES un système de transformation de programmes logiques. 7ième séminaire de programmation en logique de Trégastel, CNET, 69-84, 1988.

[2] M. Demba and F. Alexandre and K. Bsaïes. Correction of faulty conjectures and programs extraction. Proceedings of the 20th International Workshop on Disproving Non-Theorems, Non-Validity, Non-Provability, CADE'05,Tallinn, Estonia, 2005.

[3] M. Frănová and Y. Kodratoff. Predicate synthesis from formal specifications. In B. Neumann, editor, *proceedings of the 10th European Conference on Artificial Intelligence ECAI'92*, pages 87–91, Chichester, England, 1992.

[4] L. Fribourg. Extracting Logic Programs from Proofs that Use Extended Prolog Execution and Induction. In J.M. Jaquet, editor*Constructing Logic Programs, Chapter 2, pages 39–66, Wiley, 1993.*

[5] L. Fribourg. Equivalence-Preserving Transformations of Inductive Properties of Prolog Programs. In the 5th Conference and Symposium on Logic Programming, pages 893-908, Seattle, USA,1988.

[6] P. Flener and Y. Deville. Logic Program Synthesis from Incomplete Specifications. Journal of Symbolic Computation, 15, 775–805,1993.

[7] T. Kanamori. Soundness and Completeness of Extended Execution for Proving Properties of Prolog Programs. Technical Report 175, ICOT, 1986.

[8] T. Kanamori and H. Seki. Verification of Prolog Programs Using an Extension of Execution. In *3rd International Conference on Logic Programming*, volume 225 of *Lecture Notes in Computer Science*, pages 475–489. Springer-Verlag, 1986.

[9] C. S. Peirce. Collected Papers of Charles Sanders Peirce. C. Harston and P. Weiss. editors, Harvard University Press, 1959.

[10] M. Protzen. Patching faulty conjectures. In M. McRobbie and Slaney, editors, *Proceedings of the 13th Int. Conf. on Automated Deduction , CADE13*, volume 1104 of LNAI, pages 77–91, New Brunswick, NJ,USA, 1996.

[11] R. Monroy. The use of Abduction and Recursion-Editor Techniques for the Correction of Faulty Conjectures. In *Automated Software Engineering*, pages 91–100, 2000.

[12] R. Monroy, A. Bundy, and A. Ireland. Proof plan for the correction of false conjectures. In F. Pfenning, editor, *Proceedings of the 5th Int. Conf. on Logic Programming and Automated Reasoning, LPAR'94*, volume 822 of *LNAI*, pages 54–64, Kiev, Ukraine, 1994. Springer-Verlag.

[13] H. Tamaki and T. Sato. Unfold/Fold Transformation of Logic Programs. In *Proceedings of the 2nd International Logic Programming Conference*, Uppsala, 1984.

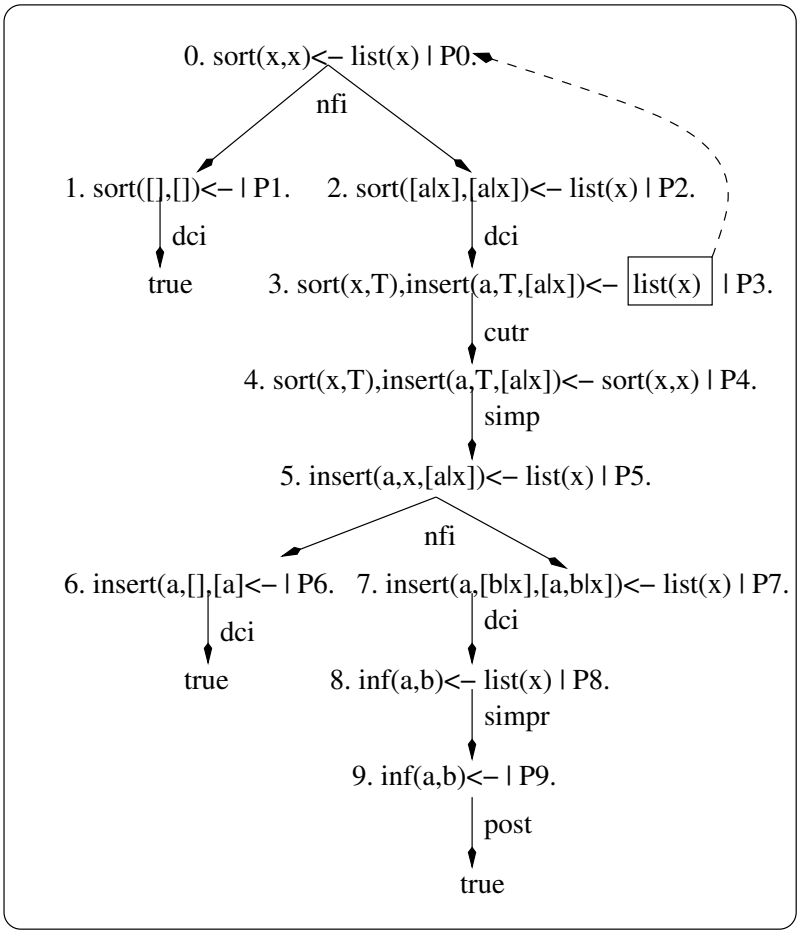

Figure 1: Proof tree of $sort(x, x) \leftarrow list(x)$.

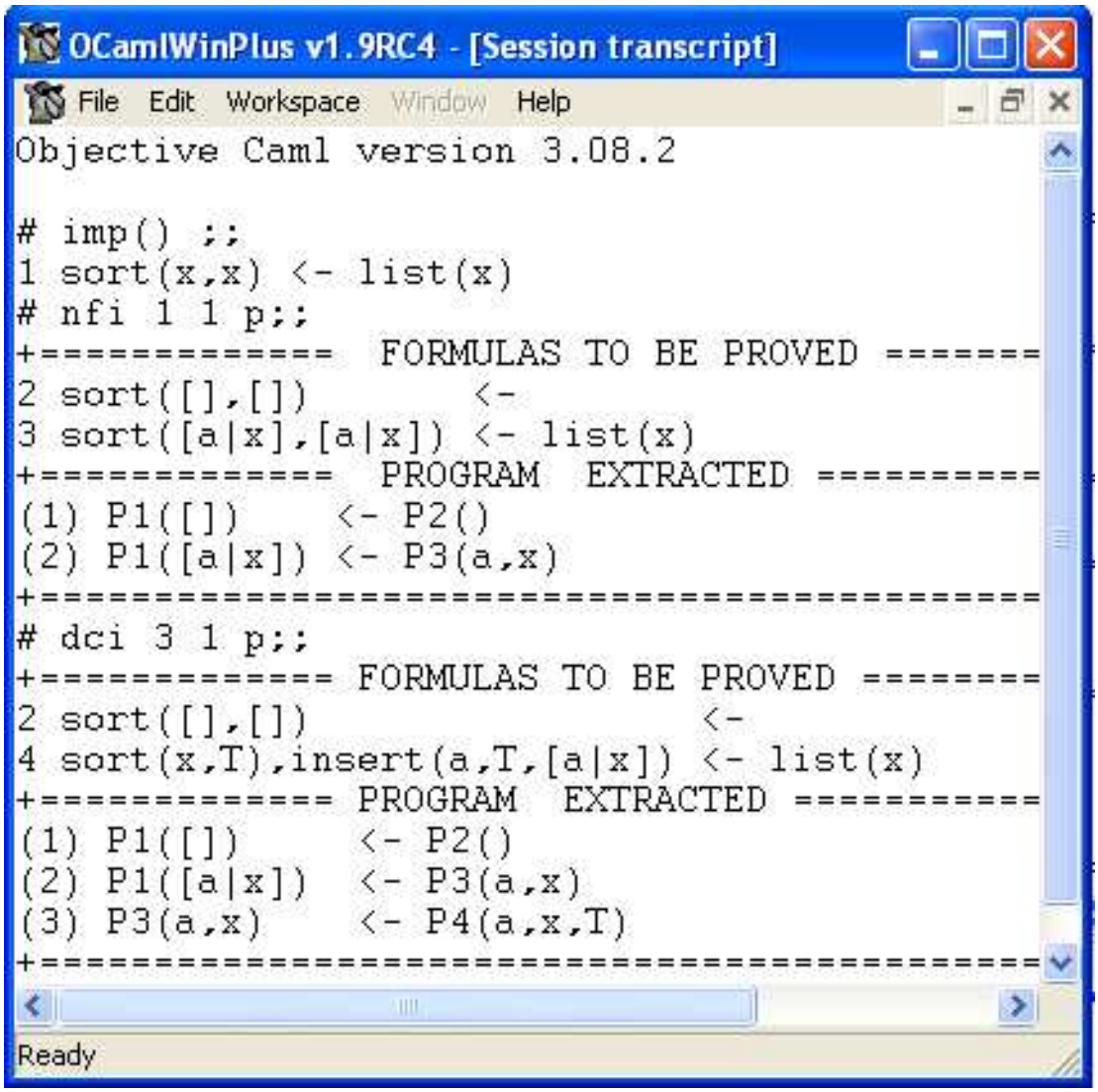

Figure 2: Proof session of $sort(x, x) \leftarrow list(x)$.