# Prouvabilité et correction des formules implicatives

Rim Zarrad, Moussa Demba, Khaled Bsaïes

# Prouvabilité et correction des formules implicatives

**Rim Zarrad** **Moussa Demba** **Khaled Bsaies**

Faculté des sciences de Tunis, Campus Universitaire el Manar DSI
2092 Tunis le Belvédaire-Tunisie
Unité de recherche URPAH
zarrad_rim@yahoo.fr {moussa.demba, khaled.bsaies}@fst.rnu.tn

**Résumé**

Dans ce papier, nous proposons une méthode permettant d'étudier la prouvabilité des formules implicatives par analyse statique. Elle consiste à caractériser les formules et les programmes par des (in)équations mathématiques permettant de vérifier si une formule est satisfiable ou non sans effectuer le processus de preuve.
Nous nous sommes aussi intéressés à la correction des formules fausses. En effet, nous proposons une méthode qui permet de synthétiser, pour une classe particulière de programmes et de formules implicatives, les prédicats de correction qui symbolisent les hypothèses manquantes à la formule pour qu'elle devienne vraie.

**Abstract**

The purpose of this work is to study, formally, the provability of implicative formulas by static analysis. To do that, we propose heuristics that allow us to know whether a given formula is satisfiable or not before attempting the proof. The method consists in characterizing the input formula and the program under consideration by mathematical (in)equations representing sufficient conditions for a proof to succeed, if it exists.
We are also interested in the analysis and correction of faulty conjectures, especially when correction involves finding a collection of hypotheses, together with a set of axioms, that turn faulty formulas into theorems. The method is based on corrective predicates invention.

## 1 Introduction

La vérification joue un rôle important dans le développement des logiciels. Une des solutions pour développer des systèmes de vérification efficaces est l'utilisation des méthodes formelles, en particulier la preuve des théorèmes. Cette méthode consiste à prouver qu'un programme donné $\mathcal{P}$ vérifie bien les propriétés pour lesquelles il a été conçu, elle se base sur le système de preuve par pliage/dépliage [7] qui n'est autre qu'un ensemble de règles de déduction.
En partant d'une formule totalement fausse, les méthodes existantes de [2, 1, 5, 3] ne peuvent détecter l'insatisfiabilité de cette formule qu'après avoir effectué le processus de preuve. Il serait plutôt intéressant de détecter que la formule est totalement fausse sans effectuer la preuve.
La problématique abordée dans cet article consiste à proposer une méthode permettant de savoir (sans utiliser un système de preuve) si une formule donnée est insatisfiable ou non. Une formule est satisfiable si elle admet une instance close (cas de base) pour laquelle la formule est vraie. Elle est insatisfiable dans le cas contraire. Cette méthode se base sur l'analyse statique de la formule et du programme associé qui n'est autre que l'extraction des informations pertinentes qui s'y trouvent.
Dans le cas où la formule n'est pas valide, on propose une approche permettant de localiser l'erreur et proposer une correction à cette formule. La correction se fait en générant un prédicat de correction qui caractérise les hypothèses manquante à la formule pour qu'elle devienne vraie.
Dans ce qui suit, nous allons tout d'abord introduire le système de preuve par pliage et dépliage, nous présentons par la suite la méthode élaborée permettant l'étude de la prouvabilité des formules implicatives. Les résultats trouvés seront exploités dans la dernière partie permettant ainsi de synthétiser un prédicat de correction à la formule.

## 2 Système de preuve

Etant donné un programme défini $\mathcal{P}$ et une formule implicative $\pi$, le système de preuve par pliage/dépliage [7]

tente de vérifier que $\pi$ est une propriété du programme $\mathcal{P}$. Il est basé sur l'application des règles de déduction.
Le schéma général d'une règle de déduction peut être vu comme suit :

$$\frac{< \pi >}{< \pi' >} \textit{(nom-règle)}$$

où $\pi$ est la formule à prouver et $\pi'$ est la conjonction ou la disjonction de formules implicatives obtenue à partir de $\pi$ en appliquant la règle de déduction *nom-règle*.
Soient $c_1, ..., c_k$ les clauses du programme $\mathcal{P}$ telles que $c_i$ : $B_i \leftarrow \Delta_i$, i = 1,..,k.
Le tableau 1 illustre les différentes règles de déduction et leurs conditions d'application.

| Nom de la règle | Schéma de la règle | Conditions |
|---|---|---|
| dépliage droit (nfi) | $\frac{\pi \ : \ \Gamma \leftarrow \Delta, A}{\wedge_{j=1}^{k}(\pi_j \ : \Gamma \leftarrow \Delta, \Delta_j)\theta_j}$ | $\theta_j = pgu(A, B_j)$ |
| dépliage gauche (dci) | $\frac{\pi \ : \Gamma, A \leftarrow \Delta}{\vee_{j=1}^{k}(\pi_j \ : (\Gamma, \Delta_j)\theta_j \leftarrow \Delta)}$ | $\theta_j = pgu(A, B_j)$ |
| pliage droit (cutr) | $\pi_0 \ : \ \Lambda \leftarrow \Sigma$<br>$\frac{\pi \ : \ \Gamma \leftarrow \Delta_1, \Delta_2}{\pi' \ : \ \Gamma \leftarrow \Lambda\theta, \Delta_2}$ | $\Sigma\theta = \Delta_1$ |
| pliage gauche (cutl) | $\pi_0 \ : \ \Lambda \leftarrow \Sigma$<br>$\frac{\pi \ : \ \Gamma_1, \Gamma_2 \leftarrow \Delta}{\pi' \ : \ \Sigma\theta, \Gamma_2 \leftarrow \Delta}$ | $\Lambda\theta = \Gamma_1$ |
| Simplification (simp) | $\frac{A, \Gamma \leftarrow B, \Delta}{\Gamma\theta \leftarrow \Delta}$ | $A\theta = B$ |

TAB. 1 – Les règles de déduction

Soient la formule $\pi_0$ : $pair(y) \leftarrow double(x, y)$ et $\mathcal{P}$ le programme suivant :

$$pair(0) \leftarrow \quad (1)$$
$$pair(s^2(x)) \leftarrow pair(x) \quad (2)$$
$$double(0, 0) \leftarrow \quad (3)$$
$$double(s(x), s^2(y)) \leftarrow double(x, y) \quad (4)$$

L'arbre de preuve associé à cette formule est représenté par la figure 1. Comme toutes les feuilles de l'arbre sont étiquetées par vrai, nous pouvons déduire que la formule $\pi_0$ est une propriété du programme $\mathcal{P}$.

## 3 Prouvabilité d'une formule

L'objectif de notre travail est d'étudier la prouvabilité d'une formule donnée. Pour cela, on va s'appuyer sur la notion de système de preuve déjà présenté, pour dégager la première instance close vraie de la formule si elle existe.

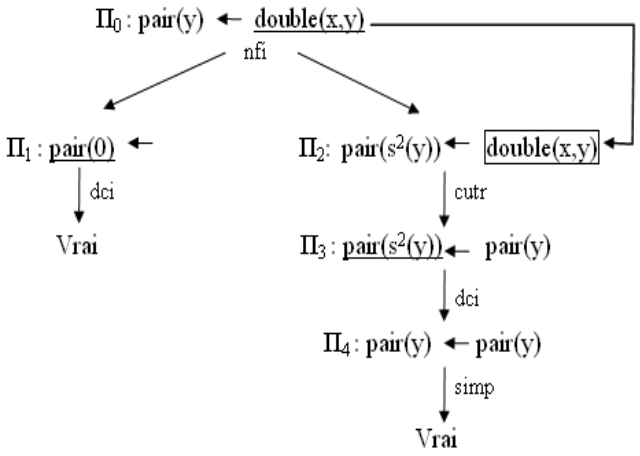


FIG. 1 – Arbre de preuve de $\pi_0$

### 3.1 Exemple introductif

Soient $\pi$ : sup(y,s(x))←double(x,y) une formule et $\mathcal{P}$ le programme logique suivant :

$$double(0, 0) \leftarrow \quad (5)$$
$$double(s(x), s^2(y)) \leftarrow double(x, y) \quad (6)$$
$$sup(s(x), 0) \leftarrow \quad (7)$$
$$sup(s(x), s(y)) \leftarrow sup(x, y) \quad (8)$$

Le processus de preuve de $\pi$ par pliage/dépliage donne l'arbre représenté par la figure 2.
A partir de cette figure, on constate que pour aboutir à la première instance clause vraie de la formule (i.e la première feuille de l'arbre étiquetée par vrai), il faut effectuer 2 nfi par rapport à 6 suivi d'un nfi par rapport à 5 puis de 3 dci par rapport à 8 et 1 dci par rapport à 7.
La méthode proposée permet de déterminer le nombre de dépliages (nfi et dci) permettant d'aboutir au premier cas vrai, s'il existe. Pour réaliser cet objectif, la formule et le programme associé sont caractérisés par des équations mathématiques dont la résolution détermine la validité de la formule dans ce programme. Lorsque la formule est totalement fausse (insatisfiable), le système d'(in)équations n'admettra pas de solutions entières.

### 3.2 Méthode proposée

L'objectif de la méthode proposée est de déterminer le nombre de dépliages nécessaires pour aboutir au premier cas de base vrai d'une formule donnée. Pour réaliser cet objectif, nous caractérisons mathématiquement les formules et les programmes considérés à travers une analyse statique afin d'extraire les informations pertinentes qui s'y trouvent. Cette analyse statique permettra l'étude de l'influence des dépliages sur les degrés d'instanciation des va-

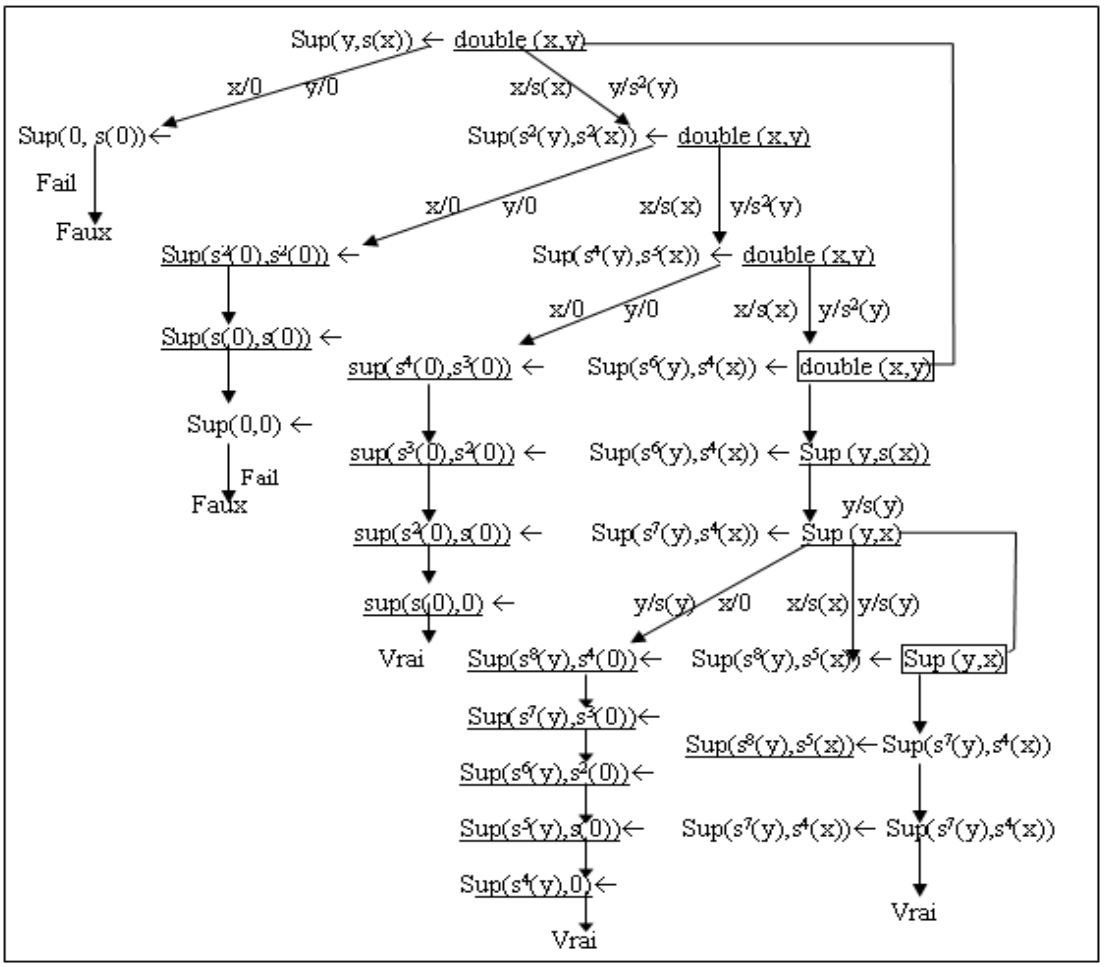


FIG. 2 – Arbre de preuve de $\pi$

riables de la formule. Elle permet de dégager un système d'(in)équations dont la résolution donne les valeurs de :

- $d_{rec}$ : nombre de nfi par rapport à la clause récursive
- $g_{rec}$ : nombre de dci par rapport à la clause récursive

Nous allons illustrer notre méthode sur la formule $\pi$ déjà présentée. Nous constatons que la recherche du premier cas de base vrai se fait en quatre étapes :

1. Application de $d_{rec}$ nfi sur double(x,y) en utilisant 6.
2. Application d'une nfi sur double(x,y) en utilisant 5.
3. Application de $g_{rec}$ dci sur sup(y,s(x)) en utilisant 8.
4. Application d'une dci sur sup(y,s(x)) en utilisant 7.

Dans ce qui suit, nous allons détailler ces étapes :

**Etape 1** L'application de $d_{rec}$ nfi récursifs influe sur le membre gauche en instanciant ses arguments. La formule obtenue aura alors cette forme :

$$\pi_1 : sup(s^{2*d_{rec}}(y), s^{d_{rec}+1}(x)) \leftarrow \text{double(x,y)}$$

**Etape 2** L'application de la clause (1) nécessite la substitution $\theta = \{x/0, y/0\}$. Nous obtenons alors la formule :

$$\pi_2 : sup(s^{2*d_{rec}}(0), s^{d_{rec}+1}(0)) \leftarrow$$

**Etape 3** L'application de $g_{rec}$ dci récursifs sur le membre gauche de $\pi_2$ donne :

$$\pi_3 : sup(s^{2*d_{rec}-g_{rec}}(0), s^{d_{rec}+1-g_{rec}}(0)) \leftarrow$$

**Etape 4** La dernière étape de la recherche du premier cas de base vrai consiste à appliquer la règle (3). Nous constatons que, pour que la formule $\pi_3$ soit vraie, il faut que :

$$\begin{cases} 2 * d_{rec} - g_{rec} \geq 1 & \Leftrightarrow \quad d_{rec} \geq 2 \\ d_{rec} - g_{rec} + 1 = 0 & \quad\quad\quad g_{rec} \geq 3 \end{cases}$$

Le système obtenu présente les conditions à vérifier par $d_{rec}$ et $g_{rec}$ pour avoir un cas de base vrai.

Pour trouver le premier cas de base vrai, il suffit d'ajouter la condition que $d_{rec}$ est minimum, ainsi nous obtenons le système linéaire (E) suivant :

$$\begin{cases} g_{rec} = d_{rec} + 1 \\ d_{rec} \geq 2 \\ d_{rec} = \mathcal{Z}(min) \end{cases}$$

La résolution de (E) donne comme solution $d_{rec}$=2 et $g_{rec}$=3. Donc pour trouver le premier cas de base vrai de $\pi$, il suffit de faire $d_{rec} + 1$ (trois) nfi suivi de $g_{rec} + 1$ (quatre) dci.

### 3.3 Généralisation

Nous nous sommes limités à une classe particulière de formules implicatives, il s'agit des formules avec un seul atome dans chacun des deux membres, et une classe particulière de programmes, ce sont les programmes récursifs linéaires décroissants à une seule clause récursive.
Soit $\pi$ une formule qui a la forme suivante :

$$P_2(s^{b_1}(x_1), ....., s^{b_n}(x_n)) \leftarrow P_1(s^{a_1}(x_1), ....., s^{a_p}(x_p))$$

Les programmes $\mathcal{P}_1$ et $\mathcal{P}_2$ définissant $P_1$ et $P_2$ sont les suivants :

$P_1(s^{\alpha_1}(d_1), ....., s^{\alpha_p}(d_p)) \leftarrow$
$P_1(s^{k_1+r_1}(z_1), .., s^{k_p+r_p}(z_p)) \leftarrow P_1(s^{k_1}(z_1), .., s^{k_p}(z_p))$
$P_2(s^{\beta_1}(c_1), ..., s^{\beta_n}(c_n)) \leftarrow$
$P_2(s^{k'_1+r'_1}(y_1), .., s^{k'_n+r'_n}(y_n)) \leftarrow P_2(s^{k'_1}(y_1), .., s^{k'_n}(y_n))$

A partir d'étude rigoureuse d'exemples, nous avons établi une approche permettant la recherche du premier cas de base vrai d'une formule. Il s'agit d'un processus formé par les quatres étapes suivantes :

1. $d_{rec}$ nfi en utilisant la clause récursive de $P_1$.
2. 1 nfi en utilisant le cas de base de $P_1$.
3. $g_{rec}$ dci en utilisant la clause récursive de $P_2$.
4. 1 dci en utilisant le cas de base de $P_2$.

Cette approche se base sur l'étude de l'influence des nfi et dci sur les degrés d'instanciation des variables de la formule. Elle consiste à caractériser les formules et les programmes par des (in)équations mathématiques permettant ainsi de vérifier si une formule est satisfiable ou non sans effectuer le processus de preuve.

## 4 Cas d'une formule insatisfiable

Notre approche permet de prouver l'insatisfiabilité d'une formule sans effectuer le processus de preuve.

**Propriété 1** *Dans le cas où la formule $\pi$ est totalement fausse, il n'existe pas de valeurs pour $d_{rec}$ et $g_{rec}$ pour lesquelles les équations sont vérifiées.*

**Exemple 1** *Soit la formule suivante :*

$$\pi : even(s(y)) \leftarrow plus(x, x, y)$$

*En appliquant notre approche sur cette formule, nous obtenons l'équation suivante : $2 * d_{rec} + 1 - 2 * g_{rec} = 0$, c'est à dire $2 * (d_{rec} - g_{rec}) = -1$. Cette équation n'a pas de solution, nous pouvons alors déduire que cette formule est totalement fausse. Ceci est vrai, en effet $\pi$ exprime le fait que si x+x=y alors y+1 est impair.*

## 5 Correction des formules fausses

Dans le domaine de la vérification, les preuves ne se terminent pas toujours avec succès. Ceci est dû soit au fait que la formule à prouver est incorrecte soit que que le programme contient des erreurs. Dans le cas où la formule est incorrecte, la plupart des prouveurs se limitent à signaler l'échec de la preuve et ne s'intéressent pas à détecter la cause de l'échec ou de proposer un moyen pour la corriger.

Certains travaux ont été élaborés dans le domaine de la correction des incohérences dans les programmes et les formules logiques. Une première classe de méthodes [6] s'intéresse à la génération des contre-exemples. La deuxième classe qui s'intéresse à la correction des conjectures est celle qui vise à synthétiser un prédicat de correction. En particulier, Frănová et Kodratoff [4] ont proposé une méthode appelée Pres qui permet de construire des prédicats de correction. Dans cette partie, nous proposons une méthode permettant de corriger des formules fausses. Etant donnés une formule $\pi$ et un programme $\mathcal{P}$ tels que $\mathcal{M}(\mathcal{P}) \not\models \pi$, notre but est d'inventer un prédicat de correction $P_0$ tel que $\mathcal{M}(\mathcal{P} \cup \mathcal{Q}) \models (\pi \leftarrow P_0)$, où $\mathcal{Q}$ est le programme définissant $P_0$.
Notre approche se base sur les résultats trouvés dans la section précédente, c'est à dire sur les valeurs de $d_{rec}$ et $g_{rec}$ représentant respectivement le nombre de nfi et de dci récursifs nécessaires pour atteindre un cas de base vrai.

### 5.1 Exemple introductif

L'étude effectuée pour la recherche du premier cas de base vrai de la formule $sup(y, x) \leftarrow double(x, y)$ donne comme résultat le système (E) suivant :

$$(E) \begin{cases} d_{rec} \geq 1 \\ g_{rec} = d_{rec} \end{cases}$$

Cela veut dire que pour trouver une substitution qui rend vraie cette formule, il faut appliquer $d_{rec} \geq 1$ dépliages récursifs du prédicat double suivi d'un dépliage du même prédicat par rapport à son cas de base, ensuite, il faut effectuer le même nombre de dépliages récursifs du prédicat sup suivi d'un dépliage du même prédicat par rapport à son cas de base.
La méthode que nous allons présenter permet de calculer le

prédicat de correction d'une formule donnée en se basant sur les valeurs de $d_{rec}$ et $g_{rec}$.
Pour dégager le cas de base de $P_0$, nous allons nous intéresser à la première substitution qui rend la formule $\pi$ vraie. La recherche de ce cas de base donne comme résultat $d_{rec} = g_{rec} = 1$. La substitution effectuée pour aboutir à ce cas de base est $\theta = \{x/s(0), y/s^2(0)\}$. Nous pouvons alors dégager la clause représentant le cas de base de $P_0$ :

$$P_0(s(0), s^2(0)) \leftarrow$$

A partir du système (E), nous pouvons constater que la différence entre deux valeurs successives de $d_{rec}$ est une constante, donc pour passer d'une instance vraie à une autre il suffit d'appliquer la même substitution $\theta = \{x/s(x), y/s^2(y)\}$. Nous pouvons ainsi conclure que si la formule $\pi$ est vraie pour x et y, alors $\pi$ est vraie pour $s(x)$ et $s^2(y)$. On obtient ainsi la clause représentant le cas récursif du prédicat de correction.

$$P_0(s(x), s^2(y)) \leftarrow P_0(x, y)$$

Le programme définissant $P_0$ est alors :

$$\begin{cases} P_0(s(0), s^2(0) \leftarrow \\ P_0(s(x), s^2(y) \leftarrow P_0(x, y) \end{cases}$$

### 5.2 Formalisation de la méthode proposée

Nous avons généralisé la démarche vue dans le paragraphe précédent en calculant le prédicat de correction d'une formule donnée à partir des valeurs de $d_{rec}$ et $g_{rec}$. Soit le système suivant :

$$E \begin{cases} d_{rec(min)} = \gamma \\ d_{rec(k+1)} - d_{rec(k)} = \delta \end{cases}$$

avec $d_{rec(min)}$ représente la valeur minimale de $d_{rec}$, $d_{rec(k)}$ et $d_{rec(k+1)}$ représentent deux valeurs consécutives de $d_{rec}$, nous avons ainsi supposé que la différence entre deux nfi récursifs menant à vrai est toujours une constante. A la formule $\pi$, on introduit le prédicat de correction $P_0(x_1, ..., x_m)$, où $x_1, ..., x_m$ sont les variables constituants les deux membres de $\pi$ qui seront dégagés à travers l'analyse statique de la formule et du programme associé.

## 6 Conclusion

L'objectif du travail effectué est d'étudier la prouvabilité d'une formule implicative donnée en détectant son premier cas de base vrai. Ce dernier est déterminé en calculant le nombre de nfi et dci nécessaires pour aboutir à vrai. La méthode proposée offre aussi la possibilité de détecter qu'une formule est insatisfiable sans effectuer le processus de preuve. Les résultats ainsi obtenus sont utilisés pour la correction des spécifications fausses. Etant donnés une formule $\Pi$ et un programme $\mathcal{P}$ tels que $\mathcal{M}(\mathcal{P}) \not\models \Pi$, nous avons présenté une méthode permettant de synthétiser un prédicat de correction $P_0$ telle que $\mathcal{M}(\mathcal{P} \cup \mathcal{Q}) \models (\Pi \leftarrow P_0)$, où $\mathcal{Q}$ est le programme définissant $P_0$.

Nous nous limitons aux formules implicatives à un seul atome dans chaque membre et aux programmes récursifs linéaires, décroissants et à une seule clause récursive. Il serait intéressant d'élargir cette classe de programmes sur lesquels nous pouvons appliquer nos méthodes. En effet, un programme $P$ dont le membre droit contient plus qu'un prédicat peut être étudié en appliquant une projection sur un seul prédicat du membre droit. Nous obtenons un programme $\breve{P}$ auquel nous pouvons appliquer les procédures déjà élaborées. Une heuristique peut alors être établie pour généraliser les résultats sur cette classe de programmes.

## Références


[1] M. Demba, F. Alexandre, and K. Bsaïes. Correction de conjectures fausses. Technical Report TR A02-R-492, LORIA, France, 2002.

[2] M. Demba, K. Bsaïes, and F. Alexandre. Proving theorems by folding/unfolding. In *Proc. of the International Conference :Sciences of Electronic, Technologies of Information and Telecommunications (SETIT'04)*, *Sousse Tunisia*, March 2004.

[3] L. Fribourg. Extracting Logic Programs from Proofs that Use Extended Prolog Execution and Induction. In D.H.D. Warren and P. Szeredi, editors, *7th International Conference on Logic Programming*, pages 685–699. MIT Press, 1990.

[4] M. Frănová and Y. Kodratoff. Predicate synthesis from formal specifications or using mathematical induction for finding preconditions of theorems. Rap. de Rech. No.646, L.R.I, Orsay, 1991.

[5] R. Monroy and A. Bundy. On the correction of faulty formulae. *Informatics Research Report EDI-INF-RR-0182*, 5(1) :165–173, Septembre 2001. Centre for intelligence systems and their applications.

[6] M. Protzen. Patching faulty conjectures. In *In M. A. McRobbie and J. K. Slaney, editors, 13th Conference on Automated Deduction*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 77–91. Springer, 1996.

[7] A. Sakurai and H. Motoda. Proving Definite Clauses without Explicit Use of Inductions. In K. Furukawa, H. Tanaka, and T Fujisaki, editors, *Proceedings of the 7th Conference, Logic Programming'88*, volume 383 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1988.