

# Vers une automatisation de la certification des propriétés de clôture pour Prolog

Thierry Marianne   Fred Mesnard   Étienne Payet

Laboratoire d'Informatique et de Mathématiques (LIM)  
Université de La Réunion

JFLA 2025



UR | UNIVERSITÉ  
DE LA RÉUNION

# Plan

- 1 Introduction
- 2 Analyse de clôture par interprétation abstraite
- 3 Certification de propriétés de clôture avec LPTP
- 4 Évaluation expérimentale
- 5 Contact et remerciements

# Introduction

## Combiner interprétation abstraite et preuve interactive

Aux extrêmes du spectre de l'automatisation de la vérification logicielle

- *Interprétation abstraite* et
- *Preuve interactive*

Choix de l'*analyse de clôture*<sup>1</sup> de la programmation logique afin de combiner ces deux techniques :

- Calcul de propriétés de clôture par *interprétation abstraite*
- Construction de dérivations dans la syntaxe de LPTP (*Logic Program Theorem Prover*)<sup>2</sup>
- Certification des dérivations par le vérificateur de preuve de LPTP
- Évaluation expérimentale de la méthodologie

---

1. *Groundness analysis* en anglais

2. Assistant de preuve en déduction naturelle conçu par Robert F. Stärk

# Introduction

## Définition (Terme)

Les termes sont définis de manière inductive tel que suit :

- une variable est un terme
- si  $f$  est un symbole de fonction  $m$ -aire et  $t_1, \dots, t_m$  des termes alors  $f(t_1, \dots, t_m)$  est un terme. En particulier, toute constante est un terme pour un symbole de fonction d'arité 0.

## Définition (Atome)

Si  $p$  est un symbole de relation  $n$ -aire et  $t_1, \dots, t_n$  des termes alors  $p(t_1, \dots, t_n)$  est un atome.

## Définition (Terme clos)

Un terme  $t$  est clos lorsqu'il ne contient aucune variable. Par exemple, toute constante est un terme clos, en particulier la constante  $[]$  dénotant la liste vide.

# Analyse de clôture par interprétation abstraite

*L'interprétation abstraite*, méthode formelle de conception d'**approximations de la sémantique d'un programme**

- servant à **collecter des informations pour l'analyse du flot des données** pour des domaines concrets [CC77]
- résultant en une **abstraction du comportement du programme** durant son exécution [CC92].

L'interprétation abstraite étendue et formalisée

- à l'analyse de flot des programmes logiques et,
- dans le cadre de **l'inférence de modes** en particulier [Bru01] [CC92].

En **analyse de clôture**, les modes des prédicats appelés sont des approximations du domaine concret de l'exécution d'un but [Mel86].

# Analyse de clôture par interprétation abstraite

## Exemple d'application de l'analyse de clôture

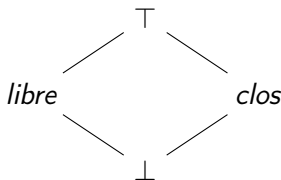
L'**analyse de clôture** est mise à contribution lors de l'**optimisation de la compilation de code Prolog**.

En particulier, le code Prolog peut être spécialisé afin de simplifier l'algorithme d'unification des variables au moyen de filtrage par motif.

## Définition (Mode d'un prédicat)

Un mode d'un prédicat dans un programme logique indique comment ses **arguments** seront **instanciés à l'appel de ce prédicat**. Les arguments sont associés aux classes d'équivalence du treillis ci-après

- *vide*, l'ensemble vide,  $\perp$ ,
- *clos*, l'ensemble des termes clos,
- *libre*, l'ensemble des variables libres et
- *inconnu*, l'ensemble de tous les termes,  $\top$ .



## Définition ( $Bfun_n$ , Fonctions booléennes)

Pour tout entier naturel  $n$ , une fonction booléenne  $n$ -aire est une fonction  $Bool^n \rightarrow Bool$ , où  $Bool$  est un domaine booléen. L'ensemble de toutes les fonctions booléennes  $n$ -aire est dénoté  $Bfun_n$ .

Fonctions booléennes comme **approximations représentant les dépendances entre la clôture des variables**  $Bfun_n$  [AMSS98] [MS93].

Exemple :

Soient  $y$  et  $x$  des **variables propositionnelles**,  
la fonction booléenne  $y \rightarrow x$  représente **les états d'instanciations respectifs des variables  $Y$  et  $X$**  d'un programme logique :

« Lorsque la variable  $Y$  du programme est instanciée à un terme clos, alors la variable  $X$  devient un terme clos à son tour »

## Exemple de la concaténation de listes

Considérons le programme logique  $P$  constitué du prédicat *append*/3

```
append([], Xs, Xs).  
append([X|Xs], Ys, [X|Zs]) :-  
    append(Xs, Ys, Zs).
```

L'analyse de  $P$  par interprétation abstraite donne la relation de dépendance de clôture des arguments sous forme booléenne suivante :

$$(x_{append} \wedge y_{append}) \leftrightarrow z_{append} \quad (1)$$

D'après cette formule, le dernier argument  $z_{append}$  du prédicat *append*/3 est clos si et seulement si son premier argument  $x_{append}$  et son deuxième argument  $y_{append}$  sont clos.

## Succès de l'appel à *append/3*

La relation (1) donne les états d'instanciations des arguments **au succès de l'appel du prédicat *append/3***.

$$(x_{append} \wedge y_{append}) \leftrightarrow z_{append} \quad (1)$$

```
append(clos, clos, clos).  
append(libre, clos, libre).  
append(clos, libre, libre).  
append(libre, libre, libre).
```

Dans les années 90, Robert F. Stärk définit sa théorie de vérification de programmes Prolog [Sta98].

$\text{IND}(P)$  est un ensemble d'axiomes du premier ordre associé au programme logique  $P$  incluant

- la complétion de Clark [Cla78] et
- l'induction d'après la définition de ses prédicats

# Formulation de propriétés de clôture avec LPTP

Dans  $\text{IND}(P)$ , la preuve de clôture d'un terme  $t$  implique que ce **terme est clos par l'évaluation d'un but** [Sta98].

Cette garantie permet de formuler des propriétés de clôture de  $P$  dans le langage de spécification de LPTP (*Logic Program Theorem Prover*) dont la sémantique est celle du calcul des prédicats de la logique classique [Sta98].

Nous **dérivons automatiquement** ces formules en déduction naturelle avant de les **certifier** avec LPTP.

# Certification de propriétés de clôture avec LPTP

## Axiome II de $\text{IND}(P)$

Dans le format de LPTP, les propriétés de clôture sont exprimées par appel au **prédicat unaire** *gr* exprimant qu'un terme est clos.

L'application de l'*axiome II* de  $\text{IND}(P)$  [Sta98]<sup>3</sup> permet d'introduire des équivalences quand le prédicat *gr* est appliqué aux termes composés.

### Axiome II de $\text{IND}(P)$ pour *gr*

4.  $gr(c)$  [si  $c$  est une constante]

5.  $gr(x_1) \wedge \dots \wedge gr(x_m) \Leftrightarrow gr(f(x_1, \dots, x_m))$  [si  $f$  est  $m$ -aire]

---

3. faisant partie intégrante de LPTP

# Formule de clôture au format LPTP

Opérateur déclaratif **Succeeds**

LPTP permet de prouver des propriétés opérationnelles d'un programme logique. L'opérateur déclaratif **Succeeds** exprime le **succès de l'appel du prédicat** auquel il est appliqué [Sta98].

La traduction automatique à partir du code Prolog du prédicat *append/3* en une définition inductive par LPTP donne l'équivalence :

$$\forall x, y, z (\mathbf{S\ append}(x, y, z) \leftrightarrow (x = [] \wedge z = y) \vee (\exists v_0, xs, zs (x = [v_0|xs] \wedge z = [v_0|zs] \wedge \mathbf{S\ append}(xs, y, zs))))$$

# Formule de clôture au format LPTP

Le schéma d'*axiome VIII* du cadre théorique  $\text{IND}(P)$  de LPTP [Sta98] nous autorise à générer un **axiome d'induction pour un prédicat directement récursif** défini par l'utilisateur et une formule à prouver [MMP24].

Forme générale de la formule de clôture à **certifier avec LPTP** dont la conclusion en forme normale disjonctive est **obtenue par interprétation abstraite** :

$$\forall x_1, \dots, x_i \text{ SR}(t_1, \dots, t_j) \implies \bigvee_{k=1}^{Dis} \bigwedge_{x_m \in Con_k} gr(x_m)$$

- où  $i, j, k, m$  sont des entiers naturels,
- $R$  est un prédicat non-prédéfini  $n$ -aire,
- $t_1, \dots, t_j$  les arguments passés à  $R$  lorsqu'il réussit (termes comprenant des variables  $x_1, \dots, x_i$ ),
- $Dis$  le nombre de disjonctions et
- $Con_k$  l'ensemble des variables dont les propriétés de clôture sont exprimées par le  $k$ -ème disjunct.

# Formule de clôture au format LPTP

Exemple de la clôture des arguments du prédicat *append*/3

Démontrons le lemme **append3\_gr** obtenu par application de l'*axiome VIII* de  $\text{IND}(P)$  à partir de la relation (1) en Forme Normale Disjonctive obtenue par interprétation abstraite pour *append*/3 :

**Lemma [append3\_gr]**  $\forall x_1, x_2, x_3 (\mathbf{S} \text{ append}(x_1, x_2, x_3) \rightarrow$   
 $(gr(x_3) \wedge gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge gr(x_2) \wedge \neg gr(x_1)) \vee$   
 $(\neg gr(x_3) \wedge \neg gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge \neg gr(x_2) \wedge \neg gr(x_1))).$

# Dérivation de formule de clôture

Nous générons un schéma d'induction à partir de

- la définition inductive du prédicat *append*/3
- la Forme Normale Disjonctive de la formule à prouver
- la syntaxe de LPTP (ISO-Prolog) et ses tactiques de preuves

en nous inspirant de la preuve de la proposition 1.38 issue de l'ouvrage *Logic in Computer Science* de Huth et Ryan [HR00].

**Lemma 1** [*append3-gr*]  $\forall x_1, x_2, x_3 (\mathbf{S} \text{append}(x_1, x_2, x_3) \rightarrow gr(x_3) \wedge gr(x_2) \wedge gr(x_1) \vee \neg gr(x_3) \wedge gr(x_2) \wedge \neg gr(x_1) \vee \neg gr(x_3) \wedge \neg gr(x_2) \wedge gr(x_1) \vee \neg gr(x_3) \wedge \neg gr(x_2) \wedge \neg gr(x_1)).$

**Proof.**

Induction<sub>0</sub>:  $\forall x_1, x_2, x_3 (\mathbf{S} \text{append}(x_1, x_2, x_3) \rightarrow gr(x_3) \wedge gr(x_2) \wedge gr(x_1) \vee \neg gr(x_3) \wedge gr(x_2) \wedge \neg gr(x_1) \vee \neg gr(x_3) \wedge \neg gr(x_2) \wedge gr(x_1) \vee \neg gr(x_3) \wedge \neg gr(x_2) \wedge \neg gr(x_1)).$

Hypothesis<sub>1</sub>: none.  $\perp$  by **GAP**.

Conclusion<sub>1</sub>:  $gr(x_4) \wedge gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]) \vee \neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge \neg gr(x_4) \wedge \neg gr([]).$

Hypothesis<sub>1</sub>:  $gr(x_8) \wedge gr(x_7) \wedge gr(x_6) \vee \neg gr(x_8) \wedge gr(x_7) \wedge \neg gr(x_6) \vee \neg gr(x_8) \wedge \neg gr(x_7) \wedge gr(x_6) \vee \neg gr(x_8) \wedge \neg gr(x_7) \wedge \neg gr(x_6)$  and  $\mathbf{S} \text{append}(x_6, x_7, x_8)$ .  $\perp$  by **GAP**.

Conclusion<sub>1</sub>:  $gr([x_5|x_8]) \wedge gr(x_7) \wedge gr([x_5|x_6]) \vee \neg gr([x_5|x_8]) \wedge gr(x_7) \wedge \neg gr([x_5|x_6]) \vee \neg gr([x_5|x_8]) \wedge \neg gr(x_7) \wedge gr([x_5|x_6]) \vee \neg gr([x_5|x_8]) \wedge \neg gr(x_7) \wedge \neg gr([x_5|x_6]). \quad \square$

où  $x_4, x_5, x_6, x_7, x_8$  sont des variables fraîches introduites afin de prouver le cas de base et l'étape d'induction.

## Proposition (Proposition 1.38 de [HR00])

*Soit une formule  $\phi$  avec  $p_1, p_2, \dots, p_n$  ses atomes propositionnels. Soit  $l$  toute ligne du tableau de vérité de  $\phi$ . Pour tout  $1 \leq i \leq n$ , soit  $\hat{p}_i$  égal à  $p_i$  si l'entrée de  $p_i$  à la ligne  $l$  est  $T$ , autrement  $\hat{p}_i$  est  $\neg p_i$ . Nous avons alors*

- ❶  *$\hat{p}_1, \hat{p}_2, \dots, \hat{p}_n \vdash \phi$  peut être démontrée si l'entrée pour  $\phi$  à la ligne  $l$  est  $T$ .*
- ❷  *$\hat{p}_1, \hat{p}_2, \dots, \hat{p}_n \vdash \neg\phi$  peut être démontrée si l'entrée pour  $\phi$  à la ligne  $l$  est  $F$ .*

Une **preuve de la formule de clôture cible existe en déduction naturelle** par complétude du calcul des propositions (Théorème 1.37 de Huth et Ryan [HR00]) comme il s'agit d'un **invariant de notre programme** inféré par interprétation abstraite et exprimé sous forme booléenne.

# Procédure récursive de dérivation de la formule de clôture

```
prove_with_premises(Phi, TruthValue, Premises)
  switch Phi
    case gr(Form) : ... return Deriv
    case ~ Form   : ... return Deriv
    case Phi1 & Phi2 : ... return Deriv
    case Phi1 \ / Phi2 : ... return Deriv
    case Phi1 => Phi2 : ... return Deriv
```

Entrée :

- $\Phi$  : Formule de clôture
- TruthValue : Valeur de vérité à démontrer pour  $\Phi$
- Premises : Hypothèses de clôture des variables de  $\Phi$

Sortie :

- Deriv : Terme de preuve au format LPTP de  $\Phi$  ou de  $\neg\Phi$

# Exemple de construction de preuve

- Cas de base :  $\forall x_1, x_2, x_3 \text{ Sappend}(x_1, x_2, x_3) \leftrightarrow x_1 = [] \wedge \exists x_4 \text{ telle que } x_4 = x_2 \wedge x_4 = x_3$
- Principe du tiers exclu donne  $gr(x_4) \vee \neg gr(x_4)$

**Proof.**

Induction<sub>0</sub> :  $\forall x_1, x_2, x_3 (\text{Sappend}(x_1, x_2, x_3) \rightarrow (gr(x_3) \wedge gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge gr(x_2) \wedge \neg gr(x_1)) \vee (\neg gr(x_3) \wedge \neg gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge \neg gr(x_2) \wedge \neg gr(x_1)))$ .

Hypothesis<sub>1</sub> : none.

Case<sub>2</sub> :  $gr(x_4)$ .  $gr(x_4) \wedge gr(x_4)$ .  $gr(x_4) \wedge gr(x_4) \wedge gr([])$ .  
 $(gr(x_4) \wedge gr(x_4) \wedge gr([])) \vee \neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([])$ .  
 $gr(x_4) \wedge gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]) \vee$   
 $\neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([])$ .  
 $gr(x_4) \wedge gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]) \vee$   
 $\neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge \neg gr(x_4) \wedge \neg gr([])$ .

Case<sub>2</sub> :  $\neg gr(x_4)$ . by **GAP**

Hence<sub>2</sub>, in all cases :  $gr(x_4) \wedge gr(x_4) \wedge gr([]) \vee$   
 $\neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]) \vee \neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([]) \vee$   
 $\neg gr(x_4) \wedge \neg gr(x_4) \wedge \neg gr([])$ .

Conclusion<sub>1</sub> :  $gr(x_4) \wedge gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]) \vee$   
 $\neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge \neg gr(x_4) \wedge \neg gr([])$ .  $\square$

# Exemple de construction de preuve

## Cas de base de la formule de clôture

$$\forall x_4 \text{ Sappend}([], x_4, x_4) \implies (gr(x_4) \wedge gr([])) \vee (\neg gr(x_4) \wedge gr([]))$$

**Proof.**

Induction<sub>0</sub> :  $\forall x_1, x_2, x_3 (\text{Sappend}(x_1, x_2, x_3) \rightarrow$

$$(gr(x_3) \wedge gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge gr(x_2) \wedge \neg gr(x_1)) \vee \\ (\neg gr(x_3) \wedge \neg gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge \neg gr(x_2) \wedge \neg gr(x_1))).$$

Hypothesis<sub>1</sub> : none.

$$\text{Case}_2 : gr(x_4). \quad gr(x_4) \wedge gr(x_4). \quad gr(x_4) \wedge gr(x_4) \wedge gr([]). \\ (gr(x_4) \wedge gr(x_4) \wedge gr([])) \vee \neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]). \\ gr(x_4) \wedge gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]) \vee \\ \neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([]). \\ gr(x_4) \wedge gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]) \vee \\ \neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge \neg gr(x_4) \wedge \neg gr([]).$$

Case<sub>2</sub> :  $\neg gr(x_4)$ . by **GAP**

$$\text{Hence}_2, \text{ in all cases : } gr(x_4) \wedge gr(x_4) \wedge gr([]) \vee \\ \neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]) \vee \neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([]) \vee \\ \neg gr(x_4) \wedge \neg gr(x_4) \wedge \neg gr([]).$$

$$\text{Conclusion}_1 : gr(x_4) \wedge gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]) \vee \\ \neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge \neg gr(x_4) \wedge \neg gr([]). \quad \square$$

# Exemple de construction de preuve

Démontrons l'étape d'induction à partir de la seconde clause de *append*/3,

$$\forall x_6, x_7, x_8 (gr(x_6) \wedge gr(x_7)) \leftrightarrow gr(x_8) \wedge \mathbf{Sappend}(x_6, x_7, x_8) \implies$$

$$\exists x_5 (gr([x_5|x_6]) \wedge gr(x_7)) \leftrightarrow gr([x_5|x_8])$$

Nous appliquons le même algorithme après énumération des 16 cas de clôture des variables  $x_5$ ,  $x_6$ ,  $x_7$  et  $x_8$ <sup>4</sup>.

**Proof.**

Hypothesis<sub>0</sub> :  $gr(x_8) \wedge gr(x_7) \wedge gr(x_6) \vee \neg gr(x_8) \wedge gr(x_7) \wedge \neg gr(x_6) \vee$   
 $\neg gr(x_8) \wedge \neg gr(x_7) \wedge gr(x_6) \vee \neg gr(x_8) \wedge \neg gr(x_7) \wedge \neg gr(x_6)$  and

**Sappend**( $x_6, x_7, x_8$ ).

Case<sub>1</sub> :  $gr(x_5)$ .

Case<sub>2</sub> :  $gr(x_6)$ .

Case<sub>3</sub> :  $gr(x_7)$ .

Case<sub>4</sub> :  $gr(x_8)$ .  $\perp$  by **GAP**.

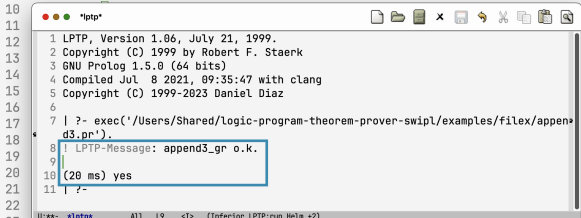
Hence<sub>4</sub>, in all cases :

$$\begin{aligned} & (gr(x_8) \wedge gr(x_7) \wedge gr(x_6)) \vee (\neg gr(x_8) \wedge gr(x_7) \wedge \neg gr(x_6)) \vee \\ & (\neg gr(x_8) \wedge \neg gr(x_7) \wedge gr(x_6)) \vee (\neg gr(x_8) \wedge \neg gr(x_7) \wedge \neg gr(x_6)) \rightarrow \\ & (gr([x_5|x_8]) \wedge gr(x_7) \wedge gr([x_5|x_6])) \vee (\neg gr([x_5|x_8]) \wedge gr(x_7) \wedge \neg gr([x_5|x_6])) \vee \\ & (\neg gr([x_5|x_8]) \wedge \neg gr(x_7) \wedge gr([x_5|x_6])) \vee \\ & (\neg gr([x_5|x_8]) \wedge \neg gr(x_7) \wedge \neg gr([x_5|x_6])). \quad \square \end{aligned}$$

# Certification de la preuve

Nous certifions la validité des dérivations obtenues en vérifiant avec LPTP le terme de preuve généré de manière automatique.

```
1 :- initialize.
2 :- needs_thm($(examples)/axiom_2_5/axiom_2_5).
3 :- needs_gr($(examples)/filex/append3).
4
5 :- lemma(append3_gr,
6   all [x1,x2,x3]:
7     (succeeds append(?x1,?x2,?x3) => gr(?x3) & gr(?x2) & gr(?x1) \
8      ~ gr(?x3) & gr(?x2) & ~ gr(?x1) \ / ~ gr(?x3) & ~ gr(?x2) & gr(?x1) \
9      ~ gr(?x3) & ~ gr(?x2) & ~ gr(?x1)),
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```



# Cadre expérimental

## Programmes issus de la bibliothèque de LPTP

| Programme            | Prop. | Vars | Inf. <sup>5</sup> | Deriv. <sup>5</sup> | Cert. <sup>5</sup> | LOC   |
|----------------------|-------|------|-------------------|---------------------|--------------------|-------|
| member.pl            | 1     | 3    | 2.5               | 5.2                 | 6.9                | 269   |
| addmul.pl            | 2     | 6    | 3.0               | 8.5                 | 16.2               | 911   |
| fib.pl               | 2     | 5    | 3.2               | 9.1                 | 14.2               | 970   |
| split.pl             | 1     | 3    | 6.9               | 14.0                | 28.9               | 1123  |
| suffix.pl            | 4     | 10   | 7.9               | 18.9                | 23.2               | 1203  |
| list.pl              | 5     | 12   | 12.4              | 28.5                | 66.6               | 2873  |
| permutation.pl       | 7     | 19   | 13.0              | 40.7                | 241.6              | 9335  |
| reverse.pl           | 4     | 10   | 11.8              | 48.4                | 95.3               | 3958  |
| sort.pl              | 9     | 22   | 41.2              | 351.7               | 12413.5            | 71067 |
| transitiveclosure.pl | 6     | 18   | 24.1              | 648.2               | 676.8              | 14213 |
| mergesort.pl         | 4     | 10   | 31.1              | 41366.8             | 1463.2             | 48400 |

5. Temps d'**Inf**érence d'invariants par interprétation abstraite, de **Dérivation**, et de **Certification** de preuves en ms

- *Complexité de génération des cas de dérivation d'ordre exponentiel* en le nombre de variables des formules de clôture à prouver.  
Un démonstrateur tel que *Vampire*<sup>6</sup> pourrait nous aider à **générer une preuve plus courte**, avant traduction au format LPTP
- Cette méthodologie pourra être étendue aux propriétés de programmes comportant des clauses *mutuellement récursives*.

---

6. <https://vprover.github.io/>

Des questions ?

Emails :

*{thierry.marianne,frederic.mesnard,etienne.payet}@univ-reunion.fr*



Nous remercions Manuel Hermenegildo, Pedro Lopez et José Morales pour les nombreuses discussions autour ce thème.



Tania Armstrong, Kim Marriott, Peter Schachte et Harald Søndergaard :

Two classes of boolean functions for dependency analysis.

*Science of Computer Programming*, 31(1):3–45, 1998.

Selected Papers of the First International Static Analysis Symposium.



Maurice Bruynooghe :

*A framework for the abstract interpretation of logic programs.*

Department of Computer Science, K.U.Leuven, Leuven, Belgium,  
1987-10-01.



Patrick Cousot et Radhia Cousot :

Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints.

*In Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL '77, page 238–252, New York, NY, USA, 1977. Association for Computing Machinery.*



Patrick Cousot et Radhia Cousot :

Abstract interpretation frameworks.

*Journal of Logic and Computation, 2(4):511–547, août 1992.*

Funding Information : We would like to thank Alan Mycroft for numerous judicial and useful comments on a first draft of this paper as well as the anonymous referees for their constructive suggestions for improvement. This work was supported in part by Esprit BRA 3124 Semantique and CNRS PRC C3.



Keith L. Clark :

*Negation as Failure*, pages 293–322.

Springer US, Boston, MA, 1978.



M. Huth et M. Ryan :

*Logic in Computer Science : Modelling and Reasoning about Systems.*

Logic in Computer Science : Modelling and Reasoning about Systems.

Cambridge University Press, 2000.



C. S. Mellish :

Abstract interpretation of prolog programs.

In Ehud Shapiro, éditeur : *Third International Conference on Logic*

*Programming*, pages 463–474, Berlin, Heidelberg, 1986. Springer

Berlin Heidelberg.



Fred Mesnard, Thierry Marianne et Etienne Payet :  
Automated theorem proving for prolog verification.  
*In* Nikolaj Bjørner, Marijn Heule et Andrei Voronkov, éditeurs : *LPAR 2024 Complementary Volume*, volume 18 de *Kalpa Publications in Computing*, pages 137–151. EasyChair, 2024.



Kim Marriott et Harald Søndergaard :  
Precise and efficient groundness analysis for logic programs.  
*ACM Lett. Program. Lang. Syst.*, 2(1–4):181–196, mars 1993.



Robert F. Staerk :  
The theoretical foundations of lptp (a logic program theorem prover).  
*The Journal of Logic Programming*, 36(3):241–269, 1998.