Automated Certification of Logic Program Groundness Analysis

Thierry Marianne[0009−0002−0661−1825], Fred Mesnard[0000−0002−8775−7430], and Etienne Payet[0000−0002−3519−025X]

LIM - University of Reunion, France
{thierry.marianne,frederic.mesnard,etienne.payet}@univ-reunion.fr

Abstract. In this paper, we are interested in the combination of abstract interpretation and interactive theorem proving, which stand at the opposite ends of the program verification automation spectrum. Our idea is to take advantage of the respective strengths of these two techniques. As a first case study, we focus on groundness analysis of logic programs. Using the proof assistant LPTP (*Logic Program Theorem Prover*), we automate the formal certification of groundness invariants generated by abstract interpretation. We present an experimental evaluation of our approach by applying it to a set of logic programs. Our experiments are twofold. On the one hand, to certify groundness invariants we use automated theorem provers drawing on the theoretical framework of LPTP. On the other hand, we generate and certify natural deduction proofs of groundness invariants with the proof checker of LPTP.

Keywords: Abstract interpretation · Certification · Groundness analysis

1 Introduction

Abstract interpretation [8] is a tool of choice for the static analysis of programming languages. In practice, abstract interpreters are complex programs and potentially contain defects, which raises the question of the validity of the computed information [5]. This article describes two possible answers in the context of groundness analysis of logic programs. Our goal is to automatically certify groundness properties inferred by bottom-up abstract interpretation. We rely on the theoretical framework of LPTP (*Logic Program Theorem Prover*) [17] and its implementation. On the one hand, we use automated demonstration to prove our target conjectures expressed in the specification language of LPTP axiomatized in first order logic. On the other hand, we first construct derivations of the groundness properties by running an implemented completeness theorem for propositional natural deduction, as natural deduction is the proof format of LPTP. Then we certify these derivations by using the proof checker integrated in LPTP.

This article is organized as follow. In Section 2, we introduce some preliminary notions. Section 3 highlights analysis techniques based on abstract interpretation applied to logic programs to infer inter-arguments relations as boolean relations. In Section 4, we leverage automated theorem proving to generate proofs of groundness properties expressed in FOF (*First Order Form*), a well-known logic language from TPTP (*Thousands of Problems for Theorem Provers*) [18] for expressing first-order logic axioms and conjectures. Section 5 describes how we formulate groundness properties in LPTP syntax directly from the invariants inferred by abstract interpretation. We present an algorithm to construct a derivation in natural deduction for these properties in the theoretical framework of LPTP. To do so, we explicit elements based on the proof by induction of a propositional formula. Finally, in Section 6, we present an experimental evaluation of the two approaches on several logic programs.

2 Preliminaries

We refer to [1,13] for the basics of logic programming including SLD resolution for the operational semantics. We only consider *positive logic programs* and our reference semantics is the s-semantics [3], a non-ground semantics of logic programs. Let P be a positive logic program. Elements of the least fixpoint of the immediate consequences operator T_P are approximated by abstract compilation, for instance using a boolean bottom-up implementation of T_P.

We will prove that these elements belong to the non-ground representation of the least term model M_P of P with LPTP. According to [3], as $M_P = lfp(T_P)$, these proofs certify that these elements belong to the least fixpoint of T_P for the s-semantics.

3 Groundness Analysis

Abstract interpretation [8] is a formal method applied to approximate program semantics by collecting information about data flow for concrete domains, which can be unbounded when the program is executed. The result of this static analysis is an abstraction of the program behaviour [7]. Abstract interpretation was extended to the analysis of logic programs [4,9].

In the context of cTI (*constraint-based Termination Inference*) [15], an abstract interpretation technique named *abstract compilation* provides approximations of the analysed program. These approximations are expressed as inter-arguments relations in the form of boolean relations. We use them to formulate groundness properties. Indeed, dependencies between variables groundness can be represented by boolean constraints [2]. For instance, a relation such as "if the variable Y is instantiated to a ground term, then the variable X is ground" can be denoted by the boolean constraint $y \Rightarrow x$. The boolean variable y (resp. x) represents the instantiation state (ground or possibly not ground) of the variable Y (resp. X).

Let us consider the logic program P defining the predicate `append/3`:

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

The analysis of P by abstract interpretation provides the inter-argument dependency relation $(x \wedge y) \Leftrightarrow z$. This formula expresses the fact that if the call $\texttt{append}(x, y, z)$ succeeds, then, after its proof (via SLD-resolution or bottom-up computation) the third argument z is ground if and only if the first argument x and the second argument y are ground.

4 Automatic proofs by Automated Theorem Proving

In the mid-90s, Robert F. Stärk showed that for any logic programs P, if $gr(t)$ is provable, ($gr/1$ is a predefined predicate of IND(P), the theoretical framework of LPTP), then t is ground [17]. So $gr(t)$ actually expresses the groundness of a term t. IND(P) consists of first-order logic axioms associated to P. This theory includes Clark's completion [6] and induction along the definition of the predicates. This guarantee allows us to formulate groundness properties about P in the specification language of LPTP, whose semantics is the first-order calculus of classical logic [17].

We follow the methodology introduced in [14] to prove groundness properties inferred by abstract interpretation. The requirements for proving a property are the corresponding logic program P (*append* in our running example) and the invariant to be certified. We compile P into the FOF version of IND(P). Moreover, the groundness property to prove and its induction axiom are compiled as a FOF conjecture. Both this conjecture and IND(P) are stored in a single file. For instance, for our running example, the conjecture is

```
fof('lemma-append3', conjecture,
    ! [Xx1,Xx2,Xx3] : ( append_succeeds(Xx1,Xx2,Xx3)
    => ( ( ( ( ( gr(Xx3) & gr(Xx2) ) & gr(Xx1) ) |
    ( ( ~ ( gr(Xx3) ) & gr(Xx2) ) & ~ ( gr(Xx1) ) ) ) |
    ( ( ~ ( gr(Xx3) ) & ~ ( gr(Xx2) ) ) & gr(Xx1) ) ) |
    ( ( ~ ( gr(Xx3) ) & ~ ( gr(Xx2) ) ) & ~ ( gr(Xx1) ) ) ) ) )
).
```

Then we apply both *Vampire* [12] and *E Theorem Prover* [16] to the FOF file to try to prove the conjecture within a time limit. So we get either a positive answer or a "don't know" answer. We also have a trace of the positive answer. This trace is not expressed as a LPTP derivation but in a FOF format. The results of our experiments on several logic programs are reported in Sect. 6.

5 Automatic Generation of LPTP Proofs

We consider a groundness property formula provided by cTI for a pure logic program P. In [14], we showed IND(P) (see [17]) allows us the generate an induction axiom for a directly recursive predicate and a formula to prove. For example, let us prove Lemma *append3_gr* below that is inferred by abstract interpretation, where the expression $\mathbf{S}\,\texttt{append}(x_1, x_2, x_3)$ expresses success of the goal

$\texttt{append}(x_1, x_2, x_3)$ and the unary predicate *gr* of LPTP expresses groundness of its argument.

Lemma [*append3_gr*] $\forall x_1, x_2, x_3\, (\mathbf{S}\,\texttt{append}(x_1, x_2, x_3) \rightarrow (gr(x_3) \wedge gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge gr(x_2) \wedge \neg gr(x_1)) \vee (\neg gr(x_3) \wedge \neg gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge \neg gr(x_2) \wedge \neg gr(x_1)))$.

Axiom II of IND(P), whose application is hard-coded in LPTP, allows one to introduce equivalences involving *gr* applied to compound terms:

Axiom II of IND(P) for *gr* [17]
4. $gr(c)$ [if c is a constant]
5. $gr(x_1) \wedge \ldots \wedge gr(x_m) \leftrightarrow gr(f(x_1, \ldots, x_m))$ [f is m-ary]

The automatic translation by LPTP of the code of append/3 into an inductive definition results in the formula:

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

The general form of the groundness property to prove is an implication whose conclusion is in clausal disjunctive normal form. It is derived from the groundness formula inferred by cTI:

$$\forall x_1, \ldots, x_n\ \mathbf{S}\, R(x_1, \ldots, x_n) \rightarrow \bigvee \bigwedge gr\_ngr(x_j)$$

where R is a n-ary user-defined predicate and $gr\_ngr(x_j)$ denotes either $gr(x_j)$ or $\neg gr(x_j)$.

We implement an algorithm inspired by the proof of the next proposition, which is used in [11] to show the completeness of natural deduction for propositional logic.

Proposition 1 (Prop. 1.38 of [11]). *Let ϕ be a formula such that $p_1, \ldots, p_n$ are its only propositional atoms. Let l be any line number in ϕ's truth table. For all $1 \leq i \leq n$, let $\hat{p}_i$ be p_i if the entry in line l of p_i is True, otherwise $\hat{p}_i$ is $\neg p_i$. Then we have*

1. *$\hat{p}_1, \ldots, \hat{p}_n \vdash \phi$ is provable if the entry for ϕ in line l is True.*
2. *$\hat{p}_1, \ldots, \hat{p}_n \vdash \neg\phi$ is provable if the entry for ϕ in line l is False.*

This result is proved by structural induction on ϕ. Taking our inspiration from it, we replace the propositional variables in the groundness formula to prove by calls to *gr*. As LPTP is implemented in Prolog, we reuse its syntax, its declarative operator **S** for predicate success and its tactics to mechanize the generation of a complete proof term.

First of all, we enumerate the variables of the target groundness formula ϕ. The idea is to generate a proof matching the lines of the truth table of ϕ. We note that the number of lines of the table is 2^n, where n is the number of variables. We implemented the procedure `prove_with_premises`, given in pseudo-code below. It returns a derivation (`Deriv`) from a groundness property (`Phi`), some premises (`Premises`) and a truth value (`TruthValue`). It considers all possible relations (groundness `gr`, negation `~`, conjunction `&`, disjunction `\/` and implication `=>`) and contains recursive calls on the sub-formulas of `Phi` (see `Form`, `Phi1` and `Phi2` below). The procedure `assemble_derivation` transforms the groundness sub-formulas by application of Axiom II of IND(P) before assembling the groundness formula derivation in LPTP syntax from the sole `Premises` passed as argument for the formula truth value (`TruthValue`) to be shown.

```
prove_with_premises(Phi, TruthValue, Premises)
  switch Phi
    case gr(Form) :
      Deriv := assemble_derivation(gr(Form), TruthValue, Premises)
    case ~ Form :
      Deriv := prove_with_premises(Form, ~ TruthValue, Premises)
    case Phi1 & Phi2 :
      D1 := prove_with_premises(Phi1, TruthValue, Premises)
      D2 := prove_with_premises(Phi2, TruthValue, Premises)
      Deriv := concat(D1, D2)
    case Phi1 \/ Phi2 :
      D1 := prove_with_premises(
        Phi1, TruthValue, concat(~ Phi2, Premises)
      )
      D2 := prove_with_premises(
        Phi2, TruthValue, concat(~ Phi1, Premises)
      )
      Deriv := concat(D1, D2)
    case Phi1 => Phi2 :
      Deriv := prove_with_premises(
        Phi2, TruthValue, concat(Phi1, Premises)
      )
  return Deriv
```

We apply the disjunction elimination rule in a systematic way from the 2^n cases we get. This can be translated directly in LPTP by the recursive application of the *cases* tactic for disjunctions expressing that a variable is ground or free. The target groundness formula ϕ is reduced by syntactic decomposition to produce a derivation. Hence, the sequents we get depend solely on the premises originating from the successive application of the law of excluded middle for each variable. As a result, we cover all 2^n groundness cases for all variables at stake by grouping these sequents altogether.

Back to our running example, the induction axiom schema for `append`/3 and the groundness formula provided by abstract interpretation allow us to generate a proof of Lemma *append3_gr* with two gaps (**GAP**), one in the base case and one in the induction step. We apply the algorithm described above to fill these gaps. In both cases, we aim at proving a tautology (for all possible variables groundness) in disjunctive normal form. Beginning with the base case (between the two horizontal lines below), we observe that $gr([])$ is true by application of Axiom II.4 of IND(P). Moreover, we find the unique variable x_4, so two cases must be considered depending on its groundness. The algorithm previously described provides the following excerpt after filling the gap in the base case.

Lemma [*append3_gr*] $\forall x_1, x_2, x_3\ (\mathsf{S}\,\mathtt{append}(x_1, x_2, x_3) \rightarrow$
$(gr(x_3) \wedge gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge gr(x_2) \wedge \neg gr(x_1)) \vee$
$(\neg gr(x_3) \wedge \neg gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge \neg gr(x_2) \wedge \neg gr(x_1)))$.

Proof.

Induction$_0$: $\forall x_1, x_2, x_3\ (\mathsf{S}\,\mathtt{append}(x_1, x_2, x_3) \rightarrow$
$(gr(x_3) \wedge gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge gr(x_2) \wedge \neg gr(x_1)) \vee$
$(\neg gr(x_3) \wedge \neg gr(x_2) \wedge gr(x_1)) \vee (\neg gr(x_3) \wedge \neg gr(x_2) \wedge \neg gr(x_1)))$.

Hypothesis$_1$: none.

Case$_2$: $gr(x_4)$. $gr(x_4) \wedge gr(x_4)$. $gr(x_4) \wedge gr(x_4) \wedge gr([])$.
$(gr(x_4) \wedge gr(x_4) \wedge gr([])) \vee (\neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]))$.
$(gr(x_4) \wedge gr(x_4) \wedge gr([])) \vee (\neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([])) \vee$
$(\neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([]))$.
$(gr(x_4) \wedge gr(x_4) \wedge gr([])) \vee (\neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([])) \vee$
$(\neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([])) \vee (\neg gr(x_4) \wedge \neg gr(x_4) \wedge \neg gr([]))$.

Case$_2$: $\neg gr(x_4)$. $\neg gr(x_4) \wedge \neg gr(x_4)$.
$\neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([])$. $(gr(x_4) \wedge gr(x_4) \wedge gr([])) \vee$
$(\neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([])) \vee (\neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([]))$.
$(gr(x_4) \wedge gr(x_4) \wedge gr([])) \vee (\neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([])) \vee$
$(\neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([])) \vee (\neg gr(x_4) \wedge \neg gr(x_4) \wedge \neg gr([]))$.

Hence$_2$, 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$_1$: $(gr(x_4) \wedge gr(x_4) \wedge gr([])) \vee (\neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([])) \vee$
$(\neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([])) \vee (\neg gr(x_4) \wedge \neg gr(x_4) \wedge \neg gr([]))$.

Hypothesis$_1$: $\forall x_5, x_6, x_7, x_8\ (gr(x_8) \wedge gr(x_7) \wedge gr(x_6) \vee$
$\neg gr(x_8) \wedge gr(x_7) \wedge \neg gr(x_6) \vee \neg gr(x_8) \wedge \neg gr(x_7) \wedge gr(x_6) \vee$
$\neg gr(x_8) \wedge \neg gr(x_7) \wedge \neg gr(x_6)$ and $\mathsf{S}\,\mathtt{append}(x_6, x_7, x_8))$. $\perp$ by **GAP**.

Conclusion$_1$: $(gr([x_5|x_8]) \wedge gr(x_7) \wedge gr([x_5|x_6])) \vee (\neg gr([x_5|x_8]) \wedge gr(x_7) \wedge$
$\neg gr([x_5|x_6])) \vee (\neg gr([x_5|x_8]) \wedge \neg gr(x_7) \wedge gr([x_5|x_6])) \vee (\neg gr([x_5|x_8]) \wedge \neg gr(x_7) \wedge$
$\neg gr([x_5|x_6]))$. □

We apply the same technique to the induction step after enumerating its variables (x_5, x_6, x_7, x_8). The complete derivations are publicly available online[1]. Finally, we check the generated proof term with LPTP to certify the derivations that we have built.

[1] github.com/atp-lptp/automated-certification-of-logic-program-groundness-analysis

6 Experimentation

We experimented our methodology using programs from the LPTP library, to which other programs were added, see Table 1, sorted in ascending order of number of lines of code in LPTP format (column LOC).

Table 1. Experimental results

Prog.	Prop.	Vars	Inf. (ms)	V/E (ms)	FOF LOC	Deriv. (ms)	Cert. (ms)	LOC
member.pl	1	3	3	3	66	6	7	269
for.pl	2	4	4	8	243	8	6	273
addmul.pl	2	6	4	18	1412	9	16	911
ackermann.pl	1	3	4	17	1972	8	16	949
fib.pl	2	5	5	19	1768	10	14	970
nat.pl	4	8	6	166	2383	11	16	975
int.pl	6	11	10	55	6386	16	19	1049
split.pl	1	3	9	10	974	14	28	1123
suffix.pl	4	10	12	17	554	18	22	1203
list.pl	5	12	19	2456	6760	26	67	2873
derivDLS.pl	1	3	20	12306	4705	263	146	3931
reverse.pl	4	10	20	493	3451	43	93	3958
average1.pl	3	7	7	22	752	17	116	8846
permutation.pl	7	19	22	215	2864	36	216	9335
transitiveclosure.pl	6	18	43	105	10220	541	654	14213
sort.pl	9	22	72	12755	867612	319	12187	71067

We used SWI-Prolog version 9.2.2 and an Apple MacBook Pro M2 running macOS Sonoma 14.6.1. For each program (column Prog.), we measured the number of groundness properties inferred by cTI (column Prop.), the total number of variables that these properties contain (column Vars) and the time it takes for cTI to compute them (column Inf.). Using the methodology described in Sect. 4 (without proof reconstruction in LPTP format), we successfully certified all these properties automatically using *Vampire* (version 4.9) and *E Theorem Prover* (version 3.1), with a time limit of 1 min. The shortest certification time performed by either *Vampire* or *E Theorem Prover* has been reported in column V/E. The number of lines of the proofs in FOF format is provided in column FOF LOC. We also applied the technique presented in Sect. 5. We measured the time required to construct the groundness properties proof (column Deriv.) and the time to certify their derivations with LPTP (column Cert.).

We didn't find any wrong invariant from this limited experiment. Such a case would result in a timeout in the V/E column and a failure to construct the LPTP derivation for the wrong invariant.

We observe that the number of lines in the FOF format proofs (column FOF LOC) ranges from comparable to up to an order of magnitude greater than the number of lines of code in LPTP format (column LOC). Moreover, the space complexity to generate the derivation cases is exponential in the number of variables occurring in each groundness property. This complexity is one of the main limitations of our implementation. It originates from the choice of Huth and Ryan's algorithm to construct a tautology in natural deduction.

7 Conclusion

Proof assistants like Isabelle with *Sledgehammer* use tools like *cvc5*[2] without abstract interpretation. Using such tools participates in saving time when proving facts by suggesting lemmas and tactics. In order to keep the possibility for users to check entire proofs, a proof reconstruction mechanism based on certificates has been introduced [10].

This article illustrated this idea in the context of logic programming. We combine the advantage of invariants algorithmic generation by abstract interpretation and formal verification offered by an interactive proof assistant. We have considered two techniques to certify groundness properties. First, we leveraged automated theorem provers without guaranteed termination and no proof in LPTP format. Second, we have used the proof checker integrated within LPTP by automatically constructing proofs for each groundness property by applying Huth and Ryan's algorithm, which always terminates, with a complexity exponential in the number of propositional variables.

More in-depth work is needed to better compare these two techniques. We would like to take into account negation by failure as well. We aim at reducing computing time complexity exposed above. Other inter-arguments relations types, matching other abstract domains and proof procedures, belong to paths yet to be explored to show the genericity of our approach.

Acknowledgments. We are thankful to Manuel Hermenegildo, Pedro Lopez and José Morales for all the discussions we had on this topic. We also thank the anonymous reviewers for their insightful and constructive criticisms.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Apt, K.R.: Logic programming. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, pp. 493–574. Elsevier and MIT Press (1990)
2. Armstrong, T., Marriott, K., Schachte, P., Søndergaard, H.: Two classes of boolean functions for dependency analysis. Science of Computer Programming **31**(1), 3–45 (1998). https://doi.org/10.1016/S0167-6423(96)00039-1
3. Bossi, A., Gabbrielli, M., Levi, G., Martelli, M.: The s-semantics approach: Theory and applications. J. Log. Program. **19/20**, 149–197 (1994). https://doi.org/10.1016/0743-1066(94)90026-4

[2] https://cvc5.github.io/blog/2024/03/15/isabelle-reconstruction.html

4. Bruynooghe, M.: A framework for the abstract interpretation of logic programs. Department of Computer Science, K.U.Leuven, Leuven, Belgium (1987-10-01). https://doi.org/10.1016/0743-1066(91)80001-T
5. Casso, I., Morales, J.F., López-García, P., Hermenegildo, M.V.: Testing Your (Static Analysis) Truths. In: Fernández, M. (ed.) Logic-Based Program Synthesis and Transformation. pp. 271–292. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-68446-4_14
6. Clark, K.L.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977. pp. 293–322. Advances in Data Base Theory, Plemum Press, New York (1977). https://doi.org/10.1007/978-1-4684-3384-5_11
7. Cousot, P., Cousot, R.: Abstract interpretation frameworks. Journal of Logic and Computation **2**(4), 511–547 (Aug 1992). https://doi.org/10.1093/logcom/2.4.511
8. Cousot, P., Cousot, R.: 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. p. 238–252 (1977). https://doi.org/10.1145/512950.512973
9. Cousot, P., Cousot, R.: Abstract interpretation and application to logic programs. J. Log. Program. **13**(2&3), 103–179 (1992). https://doi.org/10.1016/0743-1066(92)90030-7
10. Fleury, M., Schurr, H.J.: Reconstructing veriT Proofs in Isabelle/HOL. Electronic Proceedings in Theoretical Computer Science **301**, 36–50 (Aug 2019). https://doi.org/10.4204/EPTCS.301.6
11. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems. Logic in Computer Science: Modelling and Reasoning about Systems, Cambridge University Press (2000)
12. Kovács, L., Voronkov, A.: First-order Theorem Proving and Vampire. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 1–35. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_1
13. Lloyd, J.W.: Foundations of Logic Programming, 2nd Edition. Springer (1987)
14. Mesnard, F., Marianne, T., Payet, E.: Automated theorem proving for Prolog verification. In: Bjørner, N., Heule, M., Voronkov, A. (eds.) LPAR 2024 Complementary Volume. Kalpa Publications in Computing, vol. 18, pp. 137–151. EasyChair (2024). https://doi.org/10.29007/c25r
15. Mesnard, F., Neumerkel, Ulrich et Payet, E.: cTI : un outil pour l'inférence de conditions optimales de terminaison pour Prolog. In: 10eme Journées francophones de programmation logique et programmation par contraintes (JFPLC'2001). pp. 271–286. Association Française pour la Programmation en Logique et la programmation par Contraintes (AFPLC), Hermes Science Publications, Paris, France (Apr 2001)
16. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, higher, stronger: E 2.3. In: Fontaine, P. (ed.) Proc. of the 27th CADE, Natal, Brasil. pp. 495–507. No. 11716 in LNAI, Springer (2019). https://doi.org/10.1007/978-3-030-29436-6_29
17. Stärk, R.F.: The theoretical foundations of LPTP (a logic program theorem prover). The Journal of Logic Programming **36**(3), 241–269 (1998). https://doi.org/10.1016/S0743-1066(97)10013-9
18. Sutcliffe, G.: The logic languages of the TPTP world. Logic Journal of the IGPL **31**(6), 1153–1169 (Nov 2023). https://doi.org/https://doi.org/10.1093/jigpal/jzac068