Automated Certification of Logic Program Groundness Analysis

Thierry Marianne, Fred Mesnard, Étienne Payet

September 2025

UR | UNIVERSITÉ DE LA RÉUNION

Abstract interpretation and interactive theorem proving

- At the extremes of the verification automation spectrum : *Abstract Interpretation* and *Interactive Thereom Proving*
- Proofs assistants (for instance *Isabelle* with *Sledgehammer*) can rely on SMT solvers (like *cvc5*) to automate proof drafting without abstact interpretation.
- *Run-time checking* mechanism, *random test case* generator, and *unit-test framework* have been combined to test *static analyzers*
- We have considered *groundness analysis* to combine these two techniques in the context of logic programming :
  - abstract interpretation generates invariants automatically
  - interactive theorem proving allows us to certify these invariants.

Certification of Logic Programs Groundness Analysis

We apply two distinct techniques to automate the formal certification of invariants generated by abstract interpretation.

These techniques depend on

- *proving automatically* groundnesss properties from the theoretical framework of LPTP (*Logic Program Theorem Prover* [1]) and
- the *automated construction of derivations certified* by LPTP proof checker.

[1] natural deduction proof assistant designed by Robert F. Stärk

Groundness analysis by abstract interpretation

We consider logic programming with SLD resolution as the operational semantics.

We only consider *positive logic programs* and our reference semantics is the s-semantics, a non-ground semantics of logic programs.

A term t is ground when it is variable-free.

For example, any constant is a ground term, in particular the constant [] denoting the empty list.

Groundness analysis by abstract interpretation

Abstract interpretation is a formal method invented by Patrick and Radhia Cousot for designing **program semantics approximations**

- used to **collect information for data flow analysis** for concrete domains
- resulting in a **behavior abstraction of a program** during its execution
- **generating invariants** for analyzed programs automatically.

It was extended to logic program analysis.

Groundness analysis by abstract interpretation

In the context of using cTI (*constraint-based Termination Inference*), 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. Dependencies between variable groundness can be represented by boolean constraints.

Groundness analysis by abstract interpretation

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).

List concatenation example

Let us consider the logic program P composed of the single predicate *append/3*

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

Analyzing P by abstract interpretation provides the following arguments groundness dependency relation in boolean form :

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

List concatenation example

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

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.

LPTP theoretical framework

In the mid-1990s, Robert F. Stärk defines his Prolog programs verification theory and offers LPTP (*Logic Program Theorem Prover*).

At the heart of LPTP, IND(P) is a first-order axioms set associated with the logic program P including

- Clark's completion and
- Induction along its predicates definition.

LPTP theoretical framework

In LPTP format, groundness properties are expressed by calling the **unary predicate *gr*** predefined in an axiomatic way.

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.

LPTP allows to prove operational properties for a logic program. The declarative operator **S**ucceeds expresses the **success of the predicate call** to which it is applied.

Automated proving by theorem provers

Prior to this work, translating IND(P) axioms in FOF format (*First-Order Form*) allowed us to experiment with *automated theorem provers* for Prolog program verification (LPAR-25, ICLP 2025).

The requirement for proving a property are the corresponding logic program P and the invariant to be certified. We compile P into the FOF version of IND(P). The groundness property to prove is compiled as a FOF conjecture.

Automated proving by theorem provers

In particular, we translate the invariant inferred by abstract interpretation for our example :

```
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) ) ) ) )
).
```

We prove the invariant automatically by applying both provers *Vampire* and *E Theorem Prover* within a time limit.
As a result, 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.

Automated generation in natural deduction

append/3 predicate arguments groundness example

The *axiom VIII* schema from LPTP theoretical framework IND(P) allows us to generate an **induction axiom** for

- a user-defined directly recursive predicate and
- a formula to prove.

The **append3_gr** lemma is obtained by applying this axiom from the relation inferred by abstract interpretation for *append/3*

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

Groundness formula derivation

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 $\neg gr(x_j)$.

Groundness formula derivation

We generate an inductive proof inspired by a proof in *Logic in Computer Science* authored by Huth and Ryan from

- the groundness formula to show
- the *append/3* predicate inductive definition

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

Elements of the least fixpoint of the immediate consequences operator T_P are approximated by abstract compilation.
We will prove that these elements belong to the non-ground representation of the least term model M_P of P with LPTP.
As $M_P = lfp(T_P)$, these proofs certify that these elements belong to the least fixpoint of T_P for the s-semantics.

Recursive procedure to prove the groundness formula

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

Input:

- `Phi`: Groundness formula
- `TruthValue`: Truth value to show for `Phi`
- `Premises`: Groundness hypothesis for `Phi` variables

Output:

- `Deriv`: Proof term in LPTP format for `Phi` or ¬`Phi`

Proof by induction construction example

Base case

- Base case: $\forall x_1, x_2, x_3$ **S**append$(x_1, x_2, x_3) \leftrightarrow$ $x_1 = [] \wedge \exists x_4$ with $x_4 = x_2 \wedge x_4 = x_3$
- The law of excluded middle gives $gr(x_4) \vee \neg gr(x_4)$

Proof.

Induction$_0$: $\forall x_1, x_2, x_3$ (**S 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$_2$: $\neg gr(x_4)$. ...

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$_1$: $gr(x_4) \wedge gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge gr(x_4) \wedge \neg gr([]) \vee$
$\neg gr(x_4) \wedge \neg gr(x_4) \wedge gr([]) \vee \neg gr(x_4) \wedge \neg gr(x_4) \wedge \neg gr([])$. $\square$

Proof by induction construction example

Induction step

Let us prove the induction step from *append/3* second clause,
$\forall x_6, x_7, x_8\ (gr(x_6) \wedge gr(x_7)) \leftrightarrow gr(x_8) \wedge \mathbf{S}append(x_6, x_7, x_8) \implies$
$\exists x_5 (gr([x_5|x_6]) \wedge gr(x_7)) \leftrightarrow gr([x_5|x_8])$

We apply the same algorithm after enumerating the 16 groundness cases of variables x_5, x_6, x_7 et x_8.

Proof.
Hypothesis$_0$: $gr(x_8) \wedge gr(x_7) \wedge gr(x_6) \vee \neg gr(x_8) \wedge gr(x_7) \wedge \neg gr(x_6) \vee$
$\neg gr(x_8) \wedge \neg gr(x_7) \wedge gr(x_6) \vee \neg gr(x_8) \wedge \neg gr(x_7) \wedge \neg gr(x_6)$ and
S append(x_6, x_7, x_8).
Case$_1$: $gr(x_5)$.
Case$_2$: $gr(x_6)$.
Case$_3$: $gr(x_7)$.
Case$_4$: $gr(x_8)$. ...
Hence$_4$, in all cases :
$(gr(x_8) \wedge gr(x_7) \wedge gr(x_6)) \vee (\neg gr(x_8) \wedge gr(x_7) \wedge \neg gr(x_6)) \vee$
$(\neg gr(x_8) \wedge \neg gr(x_7) \wedge gr(x_6)) \vee (\neg gr(x_8) \wedge \neg gr(x_7) \wedge \neg gr(x_6)) \rightarrow$
$(gr([x_5|x_8]) \wedge gr(x_7) \wedge gr([x_5|x_6])) \vee (\neg gr([x_5|x_8]) \wedge gr(x_7) \wedge \neg gr([x_5|x_6])) \vee$
$(\neg gr([x_5|x_8]) \wedge \neg gr(x_7) \wedge gr([x_5|x_6])) \vee$
$(\neg gr([x_5|x_8]) \wedge \neg gr(x_7) \wedge \neg gr([x_5|x_6]))$. $\square$

Proof certification

We **certify the derivations validity** by checking with LPTP the proof term automatically generated.

```
:- initialize.
:- needs_thm($(examples)/axiom_2_5/axiom_2_5).
:- needs_gr($(examples)/filex/append3).

:- lemma(append3_gr,
 all [x1,x2,x3]:
  (succeeds append(?x1,?x2,?x3) ⇒ gr(?x3) & gr(?x2) & gr(?x1) \/
    ~ gr(?x3) & gr(?x2) & ~ gr(?x1) \/ ~ gr(?x3) & ~ gr(?x2) & gr(?x1) \/
    ~ gr(?x3) & ~ gr(?x2) & ~ gr(?x1)),
```

lptp

```
LPTP, Version 1.06, July 21, 1999.
Copyright (C) 1999 by Robert F. Staerk
GNU Prolog 1.5.0 (64 bits)
Compiled Jul  8 2021, 09:35:47 with clang
Copyright (C) 1999-2023 Daniel Diaz

| ?- exec('/Users/Shared/logic-program-theorem-prover-swipl/examples/filex/append3.pr').
! LPTP-Message: append3_gr o.k.

(20 ms) yes
| ?-
```

```
U:**-  *lptp*        All   L9    <I>   (Inferior LPTP:run Helm +2)
```

Experimental results

- Methodology applied to LPTP library and some additional programs[2]
- Complete proofs published on-line[3]

Prog.	Prop.	Vars	Inf. (ms)	Vamp./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

[2] from a MacBook Pro M2 with macOS Sonoma 14.6.1.

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

Key strengths and limitations

Key strengths

- All analysed programs invariants have been proved by the automated theorem provers
- All generated derivations have been automatically certified

Limitations

When applying automated theorem provers

- Absence of proof reconstruction in LPTP syntax
- No termination guarantee

When constructing derivation automatically

- Exponential complexity in the number of variables of the properties to be certified

Follow-ups

- More efficient alternative construction of property derivations to be certified for groundness analysis
- Application of the methodology to other abstract domains

Thank you

thierry.marianne@univ-reunion.fr