# Termination of Grounding is Not Preserved by Strongly Equivalent Transformations

Yuliya Lierler[1] and Vladimir Lifschitz[2]

[1] University of Kentucky (yuliya@cs.uky.edu)
[2] University of Texas at Austin (vl@cs.utexas.edu)

**Abstract.** The operation of a typical answer set solver begins with grounding—replacing the given program with a program without variables that has the same answer sets. When the given program contains function symbols, the process of grounding may not terminate. In this note we give an example of a pair of consistent, strongly equivalent programs such that one of them can be grounded by LPARSE, DLV, and GRINGO, and the other cannot.

## 1 Introduction

The operation of a typical answer set solver, such as SMODELS,[3] DLV[4], or CLINGO[5], begins with "intelligent instantiation," or grounding—replacing the given program with a program without variables that has the same answer sets. When the given program contains function symbols, the process of grounding may not terminate. The grounder employed in the last (2010-10-14) release of DLV terminates when the input program is finitely ground in the sense of [1]. According to Theorem 5 from that paper, the class of finitely ground programs is undecidable. Before attempting to ground a program, DLV verifies a decidable condition that guarantees the termination of grounding. (Conditions of this kind are known, for instance, from [1, Section 5] and [2].) A command-line option can be used to override this "finite domain check"; ensuring termination becomes then the responsibility of the user.

In the course of a public discussion at a recent meeting[6], Michael Gelfond observed that the behavior of the current version of DLV may not be fully declarative, because of the possibility of nontermination, in the same sense in which the behavior of standard Prolog systems is not fully declarative: an "inessential" modification of a Prolog program may affect not only its runtime but even its termination, even the possibility of getting an output in principle. It is well known that termination of Prolog can be affected by very minor changes, such as changing the order of subgoals in the body of a rule, changing the order

[3] `http://www.tcs.hut.fi/Software/smodels/`
[4] `http://www.dlvsystem.com/`
[5] `http://potassco/sourceforge.net/`
[6] NonMon@30: Thirty Years of Nonmonotonic Reasoning, Lexington, KY, October 22–25, 2010.

of rules, or inserting trivial rules of the form $A \leftarrow A$. In the case of DLV, such modifications cannot affect termination. But isn't it possible that a slightly more complex transformation that has no effect on the meaning of the program would make the DLV grounder unusable?

Our goal in this note is to investigate to what degree this suspicion is justified. To make Gelfond's question precise, we need to explain what we mean by a transformation that has no effect on the meaning of the program. One possibility is consider transformations that are strongly equivalent in the sense of [3, 4]. Recall that logic programs $\Pi_1$ and $\Pi_2$ are said to be *strongly equivalent* to each other if, for every logic program $\Pi$, programs $\Pi \cup \Pi_1$ and $\Pi \cup \Pi_2$ have the same answer sets. For instance, changing the order of subgoals in the body of a rule produces a strongly equivalent program. The same can be said about changing the order of rules and about inserting a rule of the form $A \leftarrow A$. Further examples of strongly equivalent transformations are provided by removing rules that are "subsumed" by other rules of the program. For instance, a program of the form

$$\begin{array}{l} A \leftarrow B \\ A \leftarrow B, C \end{array}$$

is strongly equivalent to its first rule $A \leftarrow B$.

In this note, we give an example of a pair of consistent, strongly equivalent programs $\Pi_1$ and $\Pi_2$ such that $\Pi_1$ is finitely ground, and $\Pi_2$ is not. Thus one of these two "essentially identical" programs can be grounded by DLV, and the other cannot. The behavior of LPARSE (the grounder of SMODELS) and GRINGO 2.0.3 (the grounder of the latest version of CLINGO) is similar: they terminate on $\Pi_1$, but not on $\Pi_2$.

## 2 The Example

Program $\Pi_1$ consists of 4 rules:

$$\begin{array}{l} p(a) \\ q(X) \leftarrow p(X) \\ \leftarrow p(f(X)), q(X) \\ r(f(X)) \leftarrow q(X),\ \mathit{not}\ p(f(X)). \end{array}$$

According to the answer set semantics [5], $\Pi_1$ is shorthand for the set of ground instances of its rules:

$$\begin{array}{l} p(a) \\ q(f^i(a)) \leftarrow p(f^i(a)) \\ \leftarrow p(f^{i+1}(a)), q(f^i(a)) \\ r(f^{i+1}(a)) \leftarrow q(f^i(a)),\ \mathit{not}\ p(f^{i+1}(a)) \end{array}$$

$(i = 0, 1, \dots)$. It is easy to see that

$$\{p(a), q(a), r(f(a))\} \tag{1}$$

is an answer set of $\Pi_1$. Indeed, the reduct of $\Pi_1$ relative to this set is

$$\begin{array}{l} p(a) \\ q(f^i(a)) \leftarrow p(f^i(a)) \\ \leftarrow p(f^{i+1}(a)), q(f^i(a)) \\ r(f^{i+1}(a)) \leftarrow q(f^i(a)) \end{array}$$

$(i = 0, 1, \dots)$, and (1) is a minimal set of ground atoms satisfying these rules. As a matter of fact, each of the three grounders discussed in this note turns $\Pi_1$ into the set of facts (1) and tells us in this way that (1) is the *only* answer set of $\Pi_1$.

Program $\Pi_2$ is obtained from $\Pi_1$ by adding the rule

$$p(f(X)) \leftarrow q(X),\ \mathit{not}\ r(f(X)). \tag{2}$$

We will show that programs $\Pi_1$ and $\Pi_2$ are strongly equivalent to each other. In fact, this claim will remain true even if we drop the first two rules from each of the programs:

**Proposition 1** *The program*

$$\begin{array}{l} \leftarrow p(f(X)), q(X) \\ r(f(X)) \leftarrow q(X),\ \mathit{not}\ p(f(X)) \end{array} \tag{3}$$

*is strongly equivalent to*

$$\begin{array}{l} \leftarrow p(f(X)), q(X) \\ r(f(X)) \leftarrow q(X),\ \mathit{not}\ p(f(X)) \\ p(f(X)) \leftarrow q(X),\ \mathit{not}\ r(f(X)). \end{array} \tag{4}$$

In other words, if we take any program containing rules (3) and add to it the last of rules (4) then the answer sets of the program will remain the same.

Second, we will show that $\Pi_1$ is finitely ground, and $\Pi_2$ is not. In fact, $\Pi_1$ belongs to the class finite domain programs—the decidable set of finitely ground programs introduced in [1].

**Proposition 2** *$\Pi_1$ is a finite domain program.*

**Proposition 3** *Program $\Pi_2$ is not finitely ground.*

## 3 Proofs

### 3.1 Proof of Proposition 1

In view of the main theorem of [4], it is sufficient to prove the following fact:

**Lemma** *The formula*

$$q(x) \wedge \neg r(f(x)) \rightarrow p(f(x)) \tag{5}$$

*can be derived from the formulas*

$$\begin{array}{l}\neg(p(f(x)) \wedge q(x)),\\ q(x) \wedge \neg p(f(x)) \rightarrow r(f(x))\end{array} \tag{6}$$

*in propositional intuitionistic logic.*

**Proof of the Lemma.** The formula

$$\neg(q(x) \wedge \neg r(f(x))) \tag{7}$$

can be derived from (6) in classical propositional logic. By Glivenko's theorem,[7] it follows that it can be derived from (6) intuitionistically as well. It remains to observe that (5) is an intuitionistic consequence of (7).

### 3.2 Proof of Proposition 2

In this section, and in the proof of Proposition 3 below as well, we assume that the reader is familiar with the terminology and notation introduced in [1].

To show that $\Pi_1$ is a finite domain program we need to check that the arguments $p[1]$, $q[1]$, $r[1]$ of the predicates of $\Pi_1$ are finite-domain arguments [1, Definition 10]. Consider the argument $p[1]$. The only rule of $\Pi_1$ with $p$ in the head is $p(a)$. This rule satisfies Condition 1 from Definition 10. Consider the argument $q[1]$. The only rule with $q$ in the head is

$$q(X) \leftarrow p(X).$$

This rule satisfies Condition 2. Consider the argument $r[1]$. The only rule with $r$ in the head is

$$r(f(X)) \leftarrow q(X),\ \mathit{not}\ p(f(X)).$$

This rule satisfies Condition 3.

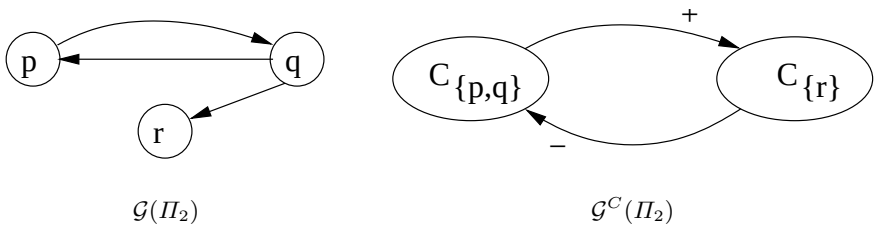
### 3.3 Proof of Proposition 3

To prove that program $\Pi_2$ is not finitely ground we need to find a component ordering $\nu$ for $\Pi_2$ such that the intelligent instantiation of $\Pi_2$ for $\nu$ is infinite [1, Definition 9]. The only component ordering for $\Pi_2$ is

$$\langle C_{\{p,q\}}, C_{\{r\}} \rangle,$$

as can be seen from the dependency graph $\mathcal{G}(\Pi_2)$ and the component graph $\mathcal{G}^C(\Pi_2)$ of this program [1, Definitions 1–4]; these graphs are shown in Figure 1. According to [1, Definition 8], the intelligent instantiation of $\Pi_2$ for

[7] This theorem [6], [7, Theorem 3.1] asserts that if a formula beginning with negation can be derived from a set $\Gamma$ of formulas in classical propositional logic then it can be derived from $\Gamma$ in intuitionistic propositional logic as well.




$\mathcal{G}(\Pi_2)$ $\mathcal{G}^C(\Pi_2)$

**Fig. 1.** The dependency graph and the component graph of program $\Pi_2$.

this component ordering is the set $S_2$ of ground rules defined by the formulas

$$S_1 = \Phi^{\infty}_{\Pi_2(C_{\{p,q\}}),S_0}(\emptyset),$$
$$S_2 = S_1 \cup \Phi^{\infty}_{\Pi_2(C_{\{r\}}),S_1}(\emptyset).$$

The module $\Pi_2(C_{\{p,q\}})$ is the program

$$p(a)$$
$$q(X) \leftarrow p(X)$$
$$p(f(X)) \leftarrow q(X),\ not\ r(f(X)),$$

and the $k$-th iteration of the operator $\Phi_{\Pi_2(C_{\{p,q\}}),S_0}$ on the empty set, for $k \geq 1$, consists of the rules

$$p(a),$$

$$q(f^j(a)) \leftarrow p(f^j(a)) \qquad (0 \leq j \leq \tfrac{k}{2} - 1),$$

$$p(f^{j+1}(a)) \leftarrow q(f^j(a)),\ not\ r(f^{j+1}(a)) \qquad (0 \leq j \leq \tfrac{k-3}{2}).$$

It is clear that the union $S_1$ of these iterations is infinite, and so is $S_2$.

## 4 Conclusion

Our goal was to find two nontrivial programs that have essentially the same meaning (which we chose to understand as "consistent programs that are strongly equivalent to each other") such that one of them can be grounded, and the other cannot. The pair $\Pi_1$, $\Pi_2$ is the simplest example that we could come up with, and the claim that these programs have essentially the same meaning is far from obvious (recall the proof of the lemma in Section 3.1). So the view that the possibility of nontermination makes the behavior of answer set solvers nondeclarative may not be justified, after all.

The fact that the termination of grounding is not preserved by strongly equivalent transformations shows, on the other hand, that such a transformation may serve as a useful preprocessing step before an attempt to ground a program.

## Acknowledgements

Thanks to Michael Gelfond and Wolfgang Faber for useful discussions. Yuliya Lierler was supported by a 2010 Computing Innovation Fellowship. Vladimir Lifschitz was partially supported by the National Science Foundation under Grant IIS-0712113.

## References

1. Calimeri, F., Cozza, S., Ianni, G., Leone, N.: Computable functions in ASP: theory and implementation. In: Proceedings of International Conference on Logic Programming (ICLP). (2008) 407–424
2. Lierler, Y., Lifschitz, V.: One more decidable class of finitely ground programs. In: Proceedings of International Conference on Logic Programming (ICLP). (2009)
3. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic **2** (2001) 526–541
4. Lifschitz, V., Pearce, D., Valverde, A.: A characterization of strong equivalence for logic programs with variables. In: Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). (2007)
5. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–385
6. Glivenko, V.: Sur quelques points de la logique de M. Brouwer. Académie Royale de Belgique. Bulletins de la Classe des Sciences, se'rie 5 **15** (1929) 183–188
7. Mints, G.: A Short Introduction to Intuitionistic Logic. Kluwer (2000)