Strong Equivalence of Logic Programs Relative to a User Guide

Jorge Fandinno[1], Zachary Hansen[1], Yuliya Lierler[1], and Vladimir Lifschitz[2]

[1] University of Omaha Nebraska
[2] University of Texas at Austin

Abstract. Studying strong equivalence is useful for the practice of answer set programming because of the fact that replacing a group of rules within a program by a strongly equivalent group of rules does not change the meaning of the program. But this behavior is not guaranteed when some symbols are used in the program as placeholders. We show how to remedy this issue by generalizing the definition of strong equivalence. The new definition makes this relation dependent on a user guide—a formal expression describing how programs are expected to be used.

1 Introduction

In answer set programming (ASP), two groups of rules are called strongly equivalent if replacing one by the other within any program does not affect the set of stable models. The precise meaning of this condition depends on the class of rules under consideration. Strong equivalence was defined and studied first for propositional programs [16], then for sets of propositional formulas [8], for infinitary propositional formulas [10], for programs with variables [17], for programs in the subset of the input language of the grounder GRINGO called mini-GRINGO [15] and for other languages based on the stable model semantics. The mini-GRINGO language is of particular interest because it is used by the proof assistant ANTHEM [4] to automatically verify certain types of program correspondence, such as strong equivalence.

Studying strong equivalence is useful for the practice of answer set programming because of the fact that replacing a group of rules within an ASP program by a strongly equivalent group of rules does not change the meaning of the program. If, for instance, we improved a program by replacing one of its rules by a simpler rule, and we checked that two rules are strongly equivalent, then we can conclude that the stable models of the program remained the same without even looking at the other rules.

But this behavior is not guaranteed when some symbols are used the programs as placeholders [9, Section 3.15]. Consider, for instance, the rule

$$p \leftarrow a < b. \tag{1}$$

In the language of GRINGO, rule (1) is strongly equivalent to $p \leftarrow$ because symbolic constants are ordered lexicographically: a is less than b. But if a and b are placeholders then replacing (1) by $p \leftarrow$ does affect the meaning of the program.

In this paper we show how to generalize strong equivalence by making it dependent on a *user guide* [5, Section 3]—a formal expression describing how programs are expected to be used. A user guide can specify, in particular, that certain symbols are supposed to serve as placeholders, and tell us which combinations of values assigned to placeholders are allowed. For example, rule (1) is not strongly equivalent to $p \leftarrow$ with respect to the user guide that classifies a and b as placeholders. But these two rules will become strongly equivalent if we add to the user guide the assumptions that the value assigned to a is supposed to be a negative integer, and the value assigned to b should be positive.

After reviewing the syntax and semantics of mini-GRINGO in the next two sections, we define strong equivalence relative to a user guide in Section 4. In the second half of the paper we discuss the relationship between generalized strong equivalence and the logic of here-and-there.

2 Rules and programs

Dialects of mini-GRINGO [7, 14] differ by the choice of arithmetic operations included in their vocabularies. Strong equivalence relative to a user guide can be defined for an arbitrary dialect, but in this paper it is discussed in the framework of a specific dialect that includes only addition and multiplication.

We assume that three countably infinite sets of symbols are selected: *numerals*, *symbolic constants*, and *variables*. We assume that a 1-1 correspondence between numerals and integers is chosen; the numeral corresponding to an integer n is denoted by $\overline{n}$. Numerals and symbolic constants are collectively called *precomputed terms*. We assume that a total order on precomputed terms is chosen such that numerals are contiguous (that is, there are no symbolic constants between numerals), and, for any integers m and n, $\overline{m} < \overline{n}$ iff $m < n$.

Terms are formed from numerals, symbolic constants and variables using the binary function symbols $+$ and $\times$. An *atom* is an expression of the form $c(\mathbf{t})$, where c is a symbolic constant, and $\mathbf{t}$ is tuple of terms separated by commas. If $\mathbf{t}$ is empty then the parentheses can be dropped. *Literals* are expressions of the forms $c(\mathbf{t})$, $not\ c(\mathbf{t})$ and $not\ not\ c(\mathbf{t})$. *Comparisons* are expressions of the form $t_1 \prec t_2$ where t_1 and t_2 are terms, and $\prec$ is $=$ or one of the symbols

$$\neq \quad < \quad > \quad \leq \quad \geq \tag{2}$$

A *rule* is an expression of the form $Head \leftarrow Body$, where *Head* is either an atom or empty, and *Body* is a conjunction of literals and comparisons, possibly empty. A *constraint* is a rule with empty head. A *choice rule* $\{c(\mathbf{t})\} \leftarrow Body$ is stands for

$$c(\mathbf{t}) \leftarrow Body \wedge not\ not\ c(\mathbf{t}).$$

A *program* is a finite set of rules.

3 Stable models and strong equivalence

An atom or another syntactic expression is *ground* if it does not contain variables. A ground atom is *precomputed* if it does not contain arithmetic operations $+$, $\times$.

The semantics of mini-GRINGO defines when a set of precomputed atoms is a stable model of a program Π. The definition, reviewed in this section, refers to the operator τ that transforms Π into a set of propositional formulas formed from precomputed atoms.

For a ground term t that does not contain symbolic constants in the scope of arithmetic operations, its *value* $[t]$ is the precomputed term defined as follows:

- if t is precomputed then $[t]$ is t;
- if $[t_1]$ is $\overline{m}$ and $[t_2]$ is $\overline{n}$ then $[t_1 + t_2]$ is $\overline{m+n}$, and $[t_1 \times t_2]$ is $\overline{m \times n}$.

For example, the value of $\overline{2} + \overline{3}$ is $\overline{5}$.

Rules and programs that do not contain symbolic constants in the scope of arithmetic operations are said to be in *numeric normal form (NNF)*.[3] The operator τ, which rewrites programs and expressions that may occur in them in the syntax of propositional formulas, will be defined first for ground rules in numeric normal form, and then extended to arbitrary rules and programs.

The result of applying τ to a ground atom $c(t_1, \dots, t_k)$ that does not contain symbolic constants in the scope of arithmetic operations is defined as the precomputed atom $c([t_1], \dots, [t_k])$. A literal *not* $c(\mathbf{t})$ is transformed by τ into $\neg\tau(c(\mathbf{t}))$, and *not not* $c(\mathbf{t})$ is transformed into $\neg\neg\tau(c(\mathbf{t}))$. A comparison $t_1 \prec t_2$ that does not contain symbolic constants in the scope of arithmetic operations is transformed into $\top$ ("true") if the relation $\prec$ holds for the pair of values $[t_1]$, $[t_2]$, and into $\bot$ ("false") otherwise. The result of applying τ to a ground NNF rule

$$c(\mathbf{t}) \leftarrow B_1 \wedge \cdots \wedge B_n \tag{3}$$

is the formula

$$\tau(B_1) \wedge \cdots \wedge \tau(B_n) \rightarrow \tau(c(\mathbf{t})).$$

A ground NNF constraint

$$\leftarrow B_1 \wedge \cdots \wedge B_n \tag{4}$$

is transformed by τ into

$$\neg(\tau(B_1) \wedge \cdots \wedge \tau(B_n)).$$

To extend the definition of τ to arbitrary rules, we define an *instance* of a rule R as a ground NNF rule that can be obtained from R by substituting precomputed terms for variables.[4] The result of applying τ to R is defined as the set of formulas obtained by applying τ to instances of R. For example, instances of the rule

$$q(X, Y + \overline{1}) \leftarrow p(X, Y) \tag{5}$$

[3] The use of the expression "normal form" here is motivated by the fact that any rule can be rewritten as an NNF rule using additional variables [7, Section 4]. For example, the rule $q(X + a) \leftarrow p(X)$ can be rewritten as $q(X + Y) \leftarrow p(X) \wedge Y = a$.

[4] The requirement that the result of substitution be in numeric normal form is similar to the use of well-formed substitutions in the ASP-Core language [1, Section 3].

are rules of the form

$$q(t, \bar{i} + \overline{1}) \leftarrow p(t, \bar{i})$$

where t is a precomputed term, and i is an integer. The result of applying τ to R is the set of formulas of the form

$$p(t, \bar{i}) \to q(t, \overline{i+1}).$$

For any program Π, $\tau(\Pi)$ is defined as the union of the sets $\tau(R)$ over all rules R of Π. Stable models of Π are defined as stable models (answer sets) of $\tau(\Pi)$ in the sense of Ferraris [8, Section 2.1].

Sets Ω_1 and Ω_2 of propositional formulas are *strongly equivalent* if, for every set Ω of propositional formulas, $\Omega_1 \cup \Omega$ and $\Omega_2 \cup \Omega$ have the same stable models [8, Section 3]. Strong equivalence of programs Π_1, Π_2 is defined as strong equivalence of the sets $\tau(\Pi_1)$ and $\tau(\Pi_2)$. If Π_1 is strongly equivalent to Π_2 then, for any program Π, the programs $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same stable models, because $\tau(\Pi_1 \cup \Pi) = \tau(\Pi_1) \cup \tau(\Pi)$ and $\tau(\Pi_2 \cup \Pi) = \tau(\Pi_2) \cup \tau(\Pi)$.

4 Valuations and user guides

A *valuation* of a set PH of symbolic constants (that are meant to be used as placeholders) is a function that maps elements of PH to precomputed terms. For any term t and any valuation v, $v(t)$ stands for the term obtained from t by simultaneously substituting the terms $v(c)$ for all symbolic constants c that belong to the domain of v. For any rule R and any valuation v, $v(R)$ stands for the rule obtained from R by replacing, in each atom $c(t_1, \dots, t_k)$ (including those in the scope of *not*) and in each comparison $t_1 \prec t_2$, every term t_i by $v(t_i)$. For any program Π, $v(\Pi)$ stands for the set of rules $v(R)$ for all rules R of Π.

The condition on a pair of programs Π_1, Π_2 that we are interested in is strong equivalence of $v(\Pi_1)$ to $v(\Pi_2)$ for all valuations v satisfying a given user guide. To define the concept of a user guide, we need a formal language that can serve for expressing assumptions about valuations, such as "a is mapped to an integer," or "a is mapped to a negative integer." For any set PH of symbolic constants, let $\sigma(PH)$ be the two-sorted signature that consists of

(i) the sort *general* and its subsort *integer*,
(ii) all numerals as object constants of the sort *integer*,
(iii) symbolic constants from PH as object constants of the sort *general*,
(iv) the symbols $+$ and $\times$ as function constants with arguments and values of the sort *integer*,
(v) comparison symbols (2) as binary predicate constant with arguments of the sort *general*.[5]

For any valuation v of PH, let $S(v)$ be the interpretation of $\sigma(PH)$ such that

[5] There is no need to include $=$ because equalities are considered part of every first-order language.

(a) its universe of the sort *general* is the set of precomputed terms,
(b) its universe of the sort *integers* is the set of numerals,
(c) each numeral is interpreted as itself,
(d) each symbolic constant c is interpreted as $v(c)$,
(e) the function symbols $+$ and $\times$ are interpreted as usual in arithmetic, and
(f) the comparison symbols are interpreted as in the definition of mini-GRINGO (Section 2).

The definitions of $\sigma(PH)$ and $S(v)$ allow us to use first-order sentences to express assumptions about valuations. For instance, a valuation v maps a to a negative integer iff $S(v)$ satisfies sentence $\exists I(a = I) \wedge a < \overline{0}$, where I is an integer variable.

A *simple user guide*[6] is a pair (PH, Asm), where PH is a finite set of symbolic constants, and Asm is a finite set of sentences over $\sigma(PH)$. Elements of PH are *placeholders* of the user guide, and elements of Asm are its assumptions.

Programs Π_1 and Π_2 are *strongly equivalent* relative to a simple user guide UG if, for every valuation v of the placeholders of UG that satisfies its assumptions, $v(\Pi_1)$ is strongly equivalent to $v(\Pi_2)$. For example, $(\{a, b\}, \emptyset)$ and

$$(\{a, b\}, \{\exists I(a = I) \wedge a < \overline{0}, \exists I(b = I) \wedge b > \overline{0}\}) \tag{6}$$

are simple user guides. Rules (1) and $p \leftarrow$ are strongly equivalent relative to the latter, but they are not strongly equivalent relative to the former.

To give another example, consider the rules

$$q(X + \overline{1}) \leftarrow p(X) \wedge X < a \tag{7}$$

and

$$q(X) \leftarrow p(X - \overline{1}) \wedge X \leq a. \tag{8}$$

They are strongly equivalent relative to the user guide $(\{a\}, \emptyset)$.

5 Review: representing rules by first-order sentences

The theorem stated in the next section shows how first-order reasoning can be sometimes used for establishing strong equivalence of mini-GRINGO programs relative to a user guide. The theorem is restricted to programs in numeric normal form. Its statement refers to the translation ν [7, Section 5], which transforms NNF rules into two-sorted first-order sentences. In this section we reproduce the definition of ν for the special case of the dialect described above.

By σ we denote the two-sorted signature that consists of the sorts and symbols listed in clauses (i), (ii), (iv), (v) above and the following additional symbols:

(iii)′ all symbolic constants as object constants of the sort *general*,
(vi) expressions c/k, where c is a symbolic constant and k is a nonnegative integer, as k-ary predicate constants with arguments of the sort *general*.

[6] The word "simple" indicates that this concept is less general than user guides defined by Fandinno et al. [5].

Note that the set of terms over the signature σ and the set of mini-GRINGO terms partially overlap. On the one hand, a term over σ may include integer variables, which are not part of mini-GRINGO. In a mini-GRINGO term, on the other hand, symbolic constants and variables may occur in the scope of an arithmetic operation; this is not allowed in terms over σ, because both symbolic constants and variables are terms of the sort *general*.

Consider a rule R in numeric normal form. Applying the translation ν to R involves substituting an integer variable for each variable that occurs in R at least once in the scope of an arithmetic operation. Make the list $V_1, \dots, V_m$ of all variables satisfying this condition, and choose m distinct integer variables $I_1, \dots, I_m$.

Consider a term t that occurs in R. Since R is in numeric normal form, symbolic constants in t are not in the scope of arithmetic operations. It follows that the result of substituting $I_1, \dots, I_m$ for $V_1, \dots, V_m$ in t is a term over the signature σ. The operator that performs this transformation will be denoted by *p2f* ("programs-to-formulas"). For example, if R is rule (5) then $m = 1$, V_1 is Y, $p2f(X)$ is X, and $p2f(Y + \overline{1})$ is $I + \overline{1}$.

The result of applying ν to an atom $c(t_1, \dots, t_k)$ occurring in R is defined as the atomic formula $(c/k)(p2f(t_1), \dots, p2f(t_k))$. A literal $not\ c(\mathbf{t})$ that occurs in R is transformed by ν into $\neg\nu(c(\mathbf{t}))$, and $not\ not\ c(\mathbf{t})$ is transformed into $\neg\neg\nu(c(\mathbf{t}))$. A comparison $t_1 \prec t_2$ occurring in R is transformed into $p2f(t_1) \prec p2f(t_2)$. If R is (3) then $\nu(R)$ is defined as the universal closure of the formula

$$\nu(B_1) \wedge \cdots \wedge \nu(B_n) \to \nu(c(\mathbf{t})).$$

If R is a constraint (4) then $\nu(R)$ is the universal closure of

$$\neg(\nu(B_1) \wedge \cdots \wedge \nu(B_n)).$$

For example, the result of applying ν to rule (5) is the sentence

$$\forall X I_1((p/2)(X, I_1) \to (q/2)(X, I_1 + 1)).$$

6 Verifying strong equivalence

In this section, we introduce a deductive system that can be used for proving strong equivalence of NNF programs relative to a user guide. Deductive systems that allow reasoning about strong equivalence of logic programs have been developed for several classes of programs, including propositional programs [16], first-order programs [17], mini-GRINGO programs [6, 12], and mini-GRINGO with counting aggregates [2, 3, 13]. The theorem below is a generalization of the results for mini-GRINGO to the more general case of strong equivalence relative to a user guide.

An interpretation of the signature σ is *standard* if it satisfies conditions (a)–(c), (e) and (f) above. Thus standard interpretations of σ may differ from each other by how they interpret symbols of two kinds: symbolic constants and predicate constants c/k. By *Std* we denote the set of sentences over σ that do not

contain symbols of these two kinds and are satisfied by standard interpretations (by at least one, and consequently by all of them).

For any set PH of symbolic constants, $HT(PH)$ ("the logic of here-and-there for placeholders PH") is the deductive system obtained from intuitionistic predicate calculus with equality for the signature σ by adding the following axioms:

- the Hosoi axiom schema [11]: $F \vee (F \to G) \vee \neg G$,
- the static domain schema [17]: $\exists X(F \to \forall X\, F)$, where X is a general variable,
- all sentences from Std,
- the induction schema

$$F(\overline{0}) \wedge \forall I(I \geq \overline{0} \wedge F(I) \to F(I + \overline{1})) \to \forall I(I \geq \overline{0} \to F(I)),$$

  where I is an integer variable.

Theorem. *For any NNF programs Π_1, Π_2 and any simple user guide (PH, Asm), if the formula $\nu(\Pi_1) \leftrightarrow \nu(\Pi_2)$ is derivable from Asm in the deductive system $HT(PH)$ then Π_1 is strongly equivalent to Π_2 relative to (PH, Asm).*

As an example, consider proving the strong equivalence of rules (1) and $p \leftarrow$ relative to user guide (6). According to the theorem, this can be accomplished by showing that the formula

$$(a < b \to p) \leftrightarrow (\top \to p))$$

is derivable in $HT(\{a, b\})$ from $\exists I(a = I) \wedge a < \overline{0}$ and $\exists I(b = I) \wedge b > \overline{0}$. To verify this claim, observe that $a < b$ is an intuitionistic consequence of $a < \overline{0}$, $b > \overline{0}$ and the following sentence from Std:

$$\forall XYZ(X < Y \wedge Z > Y \to X < Z),$$

To prove that rule (7) is strongly equivalent to rule (8) relative to the user guide $(\{a\}, \emptyset)$, we will show that the formula

$$\forall I(p(I) \wedge I < a \to q(I + \overline{1})) \leftrightarrow \forall I(p(I - \overline{1}) \wedge I \leq a \to q(I)), \tag{9}$$

where I is an integer variable, is provable in $HT(\{a\})$. Note first that the formula

$$\forall I(I - \overline{1} + \overline{1} = I) \tag{10}$$

belongs to Std. The formula

$$\forall IX(I - \overline{1} < X \leftrightarrow I \leq X), \tag{11}$$

where X is a general variable, belongs to Std as well, because numerals are contiguous (Section 2). From the left-hand side of (9) we derive

$$p(I - \overline{1}) \wedge I - \overline{1} < a \to q(I - \overline{1} + \overline{1}).$$

Now the right-hand side follows by (10) and (11). The other direction is similar.

Note that the theorem provides only a sufficient condition for strong equivalence relative to a user guide. Similar results for strong equivalence of mini-GRINGO programs without placeholders [6, 12] only provide sufficient conditions as well. We do not know whether the condition in these theorems are also necessary, but we conjecture that it is not the case. Yet, the system can be made complete by adding deductive rules with infinitely many premises called ω-rules [6].

7 Conclusion

The generalization of strong equivalence defined in this paper can be useful in the process of developing ASP programs with placeholders. In some cases, verifying generalized strong equivalence can be accomplished by reasoning in a first-order theory based on the logic of here-and-there. In the future, we plan to relate ideas of this paper to the definition of external equivalence with respect to a user guide [5] and to incorporate a process of verifying generalized strong equivalence in a new version of the proof assistant ANTHEM [4].

Acknowledgments. We would like to thank anonymous reviewers for their valuable feedback that allowed us to improve presentation. This work was partially supported by the National Science Foundation CAREER award 2338635, USA. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

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

References

1. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., Schaub, T.: ASP-Core-2 input language format. Theory and Practice of Logic Programming **20**, 294–309 (2020)
2. Fandinno, J., Lifschitz, V.: Deductive systems for logic programs with counting: Preliminary report. In: Dodaro, C., Gupta, G., Martinez, M. (eds.) Proceedings of the Seventeenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'24). Lecture Notes in Artificial Intelligence, vol. 15245, pp. 1–14. Springer-Verlag (2024). https://doi.org/10.1007/978-3-031-74209-5
3. Fandinno, J., Lifschitz, V.: Deductive systems for logic programs with counting. Theory and Practice of Logic Programming **25**(5), 924–964 (2025). https://doi.org/10.1017/S1471068425100379
4. Fandinno, J., Glinzer, C., Hansen, Z., Heuer, J., Lierler, Y., Lifschitz, V., Schaub, T., Stolzmann, T.: ANTHEM: answer set programming and automated theorem proving. Theory and Practice of Logic Programming (2025)

5. Fandinno, J., Hansen, Z., Lierler, Y., Lifschitz, V., Temple, N.: External behavior of a logic program and verification of refactoring. Theory and Practice of Logic Programming (2023)
6. Fandinno, J., Lifschitz, V.: Omega-completeness of the logic of here-and-there and strong equivalence of logic programs. In: Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (2023)
7. Fandinno, J., Lifschitz, V.: A normal form for rules containing arithmetic operations. In: Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (2026), to appear, available at https://www.cs.utexas.edu/ vl/papers/normalform_krr.pdf
8. Ferraris, P.: Answer sets for propositional theories. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). pp. 119–131 (2005)
9. Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T., Thiele, S.: Potassco User Guide (2019), available at https://github.com/potassco/guide/releases/
10. Harrison, A., Lifschitz, V., Pearce, D., Valverde, A.: Infinitary equilibrium logic and strongly equivalent logic programs. Artificial Intelligence **246**, 22–33 (May 2017)
11. Hosoi, T.: The axiomatization of the intermediate propositional systems S_n of Gödel. Journal of the Faculty of Science of the University of Tokyo **13**, 183–187 (1966)
12. Lifschitz, V.: Here and there with arithmetic. Theory and Practice of Logic Programming (2021)
13. Lifschitz, V.: Strong equivalence of logic programs with counting. Theory and Practice of Logic Programming **22** (2022)
14. Lifschitz, V.: Generalizing the syntax of terms in mini-gringo. In: Proceedings of European Conference on Logics in Artificial Intelligence) (2025)
15. Lifschitz, V., Lühne, P., Schaub, T.: Verifying strong equivalence of programs in the input language of gringo. In: Proceedings of the 15th International Conference on Logic Programming and Non-monotonic Reasoning (2019)
16. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic **2**, 526–541 (2001)
17. 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). pp. 188–200 (2007)