On Heuer's Procedure for Verifying Strong Equivalence

Jorge Fandinno[1] and Vladimir Lifschitz[2]

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

Abstract. In answer set programming, two groups of rules are considered strongly equivalent if replacing one group by the other within any program does not affect the set of stable models. Jan Heuer has designed and implemented a system that verifies strong equivalence of programs in the ASP language mini-GRINGO. The design is based on the syntactic transformation τ^* that converts mini-GRINGO programs into first-order formulas. Heuer's assertion about τ^* that was supposed to justify this procedure turned out to be incorrect, and in this paper we propose an alternative justification for his algorithm. We show also that if τ^* is replaced by the simpler and more natural translation ν then the algorithm will still produce correct results.

1 Introduction

In answer set programming (ASP), two groups of rules are considered strongly equivalent if replacing one group by the other within any program does not affect the set of stable models [21]. This equivalence relation has been extensively studied in the literature because of its interesting theoretical properties and its usefulness for the practice of answer set programming [1–6, 8, 10, 11, 15, 16, 19, 20, 22–24, 27]. Jan Heuer designed and implemented a system that verifies strong equivalence of programs in the ASP language mini-GRINGO. The system is described in his Bachelor Thesis [12], presented to the University of Potsdam.

The design of the system is based on the syntactic transformation τ^* [20], which converts mini-GRINGO rules and programs into first-order formulas. Mini-GRINGO programs Π_1, Π_2 are strongly equivalent whenever the formulas $\tau^*\Pi_1$ and $\tau^*\Pi_2$ can be derived from each other in the deductive system *HTA* ("here-and-there with arithmetic") [17].

To use a resolution theorem prover as a proof engine for *HTA*, Heuer needed an additional translation that would relate *HTA* to a classical first-order theory. The translation that he implemented is a straightforward generalization of the process proposed by Pearce et al. [25] for propositional formulas.

Unfortunately, the claim that is supposed to justify this additional translation [12, Theorem 3] is incorrect as stated, because it disregards the existence of interpretations that treat arithmetical symbols in nonstandard ways.[3] For example, the facts $p(2+3)$ and $p(5)$ are strongly equivalent, but we cannot assert

[3] Lifschitz et al. [20] made the same mistake in their Proposition 6.

that they get the same truth value under any interpretation. Indeed, the expressions $2+3$ and 5 can have different values if the symbols 2, 3, 5 and + are not interpreted as usual in arithmetic.

In this paper we show that, in spite of this difficulty, Heuer's procedure is in fact correct. Second, we show that the procedure will produce correct results if we modify it by replacing $\tau^* R$ with the simpler translation νR when the rule R is regular [18]. The paper begins with a review of the mini-GRINGO language and of the target language of the translations τ^* and ν (Sections 2 and 3). The main results of this paper are stated in Section 4. In Section 5 we give examples of their use. In Section 6 we describe an extension HTA$^\omega$ of the deductive system HTA, which plays an important role in our proofs of the main results.

2 Review: Programs

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 $\overline{n}$. *Precomputed terms* are numerals and symbolic constants. We assume that a total order on precomputed terms is chosen such that for all integers m and n, $\overline{m} < \overline{n}$ iff $m < n$.

Terms allowed in a mini-GRINGO program are formed from precomputed terms and variables using the six operation names

$$+ \quad - \quad \times \quad / \quad \backslash \quad ..$$

(the last three serve to represent integer division, remainder and intervals). An *atom* is a symbolic constant optionally followed by a tuple of terms in parentheses. A *literal* is an atom possibly preceded by one or two occurrences of *not*. A *comparison* is an expression of the form $t_1 \prec t_2$, where t_1, t_2 are terms and $\prec$ is $=$ or one of the comparison symbols

$$\neq \quad < \quad > \quad \leq \quad \geq \tag{1}$$

A *rule* is an expression of the form $Head \leftarrow Body$, where

- *Body* is a conjunction (possibly empty) of literals and comparisons, and
- *Head* is either an atom (then the rule is *basic*), or an atom in braces (then this is a *choice rule*), or empty (then this is a *constraint*).

A *(mini-*GRINGO*) program* is a finite set of rules.

The semantics of ground terms is defined by assigning to every ground term t the finite set $[t]$ of its *values* [20, Section 3]. Values of a ground term are precomputed terms. For instance,

$$[\overline{2}/\overline{3}] = \{\overline{0}\}, \ [\overline{2}/\overline{0}] = \emptyset, \ [\overline{0}\,..\,\overline{2}] = \{\overline{0}, \overline{1}, \overline{2}\}.$$

A *predicate symbol* is a pair p/n, where p is a symbolic constant, and n is a nonnegative integer. About a predicate symbol p/n we say that it *occurs* in a program Π if a rule of Π contains an atom of the form $p(t_1, \ldots, t_n)$.

Stable models of a program are defined as stable models of the set of propositional formulas obtained from it by the syntactic transformation τ [20, Section 3]. Atomic parts of these formulas are *precomputed atoms*—atoms $p(\mathbf{t})$ such that the members of $\mathbf{t}$ are precomputed terms. For example, τ transforms the rule

$$\{q(X)\} \leftarrow p(X) \tag{2}$$

into the set of formulas $p(t) \rightarrow (q(t) \vee \neg q(t))$ for all precomputed terms t. The rule

$$q(\overline{0}\,..\,\overline{2}) \leftarrow p \tag{3}$$

is transformed into $p \rightarrow (q(\overline{0}) \wedge q(\overline{1}) \wedge q(\overline{2}))$. Thus stable models of mini-GRINGO programs are sets of precomputed atoms.

Mini-GRINGO programs Π_1 and Π_2 are *strongly equivalent* to each other if, for every set Ω of propositional combinations of precomputed atoms, $\tau\Pi_1 \cup \Omega$ has the same stable models as $\tau\Pi_2 \cup \Omega$.

3 Review: Two-sorted formulas

The target language of the translations τ^* [20, Section 6] and ν [18, Sections 4, 5] is a first-order language with the sort *general* and its subsort *integer*.[4] Variables of the first sort are meant to range over arbitrary precomputed terms, and we identify them with variables used in mini-GRINGO rules. Variables of the second sort are meant to range over numerals (or, equivalently, integers). This is made precise in the definition of a standard interpretation at the end of this section.

The signature σ_0 of the language includes

- all precomputed terms as object constants; an object constant is assigned the sort *integer* iff it is a numeral;
- the symbols $+$, $-$ and $\times$ as binary function constants; their arguments and values have the sort *integer*;[5]
- predicate symbols p/n as n-ary predicate constants; their arguments have the sort *general*;
- comparison symbols (1) as binary predicate constants; their arguments have the sort *general*.

An atomic formula $(p/n)(t_1, \ldots, t_n)$ can be abbreviated as $p(t_1, \ldots, t_n)$. An atomic formula of the form $\prec(t_1, t_2)$, where $\prec$ is a comparison symbol, can be written as $t_1 \prec t_2$. A conjunction of the form $t_1 \leq t_2 \wedge t_2 \leq t_3$ can be written as $t_1 \leq t_2 \leq t_3$, and similarly for other chains of inequalities.

[4] The need to use a language with two sorts is explained by the fact that function symbols in a first-order language are supposed to represent total functions, and arithmetic operations are not defined on symbolic constants.

[5] The symbols / and \ are not included because the corresponding functions are not total on the set of integers. The symbol .. is not included because intervals do not belong to the domain of precomputed terms.

For example, the translation ν converts rule (2) into

$$\forall X(p(X) \to (q(X) \vee \neg q(X))). \tag{4}$$

Rule (3) is transformed into

$$p \to \forall N(\overline{0} \leq N \leq \overline{2} \to q(N)),$$

where N is an integer variable. The result of applying ν to the rule

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

is

$$\forall XN(p(X, N) \to q(X, N + \overline{1})). \tag{6}$$

An interpretation of the signature σ_0 is *standard* if

- its domain of the sort *general* is the set of all precomputed terms;
- its domain of the sort *integer* is the set of all numerals;
- it interprets every precomputed term t as t;
- it interprets $\overline{m}+\overline{n}$ as $\overline{m+n}$, and similarly for subtraction and multiplication;
- it interprets every atomic sentence $t_1 \prec t_2$, where t_1 and t_2 are precomputed terms, as true iff the relation $\prec$ holds for the pair (t_1, t_2).

4 Translation γ and its properties

By σ_0' we denote the extension of the signature σ_0 obtained by adding, for every predicate symbol p/n, a new n-ary predicate constant $(p/n)'$. An atomic formula $(p/n)'(\mathbf{t})$ can be abbreviated as $p'(\mathbf{t})$. For the signature σ_0', the definition of a standard interpretation is the same as for the signature σ_0 above.

For any formula F over the signature σ_0, by F' we denote the formula over σ_0' obtained from F by replacing every occurrence of every predicate symbol p/n by $(p/n)'$.

The translation γ, which relates logic of here-and-there with arithmetic to classical logic, maps formulas over σ_0 to formulas over σ_0'.[6] It is defined recursively:

- $\gamma F = F$ if F is atomic,
- $\gamma(\neg F) = \neg F'$,
- $\gamma(F \wedge G) = \gamma F \wedge \gamma G$,
- $\gamma(F \vee G) = \gamma F \vee \gamma G$,
- $\gamma(F \to G) = (\gamma F \to \gamma G) \wedge (F' \to G')$,
- $\gamma(\forall X\, F) = \forall X\, \gamma F$,
- $\gamma(\exists X\, F) = \exists X\, \gamma F$.

[6] Heuer [12, Sections 2.2.3 and 3.3] denotes this translation by σ^*. We switched to γ to avoid confusion with the symbols denoting signatures.

Our justification of Heuer's procedure is given by Theorem 1 below. In the statement of the theorem, $\mathcal{A}(p/n)$ stands for the formula $\forall \mathbf{X}(p(\mathbf{X}) \to p'(\mathbf{X}))$, where $\mathbf{X}$ is an n-tuple of distinct general variables.

Theorem 1. *Mini-*GRINGO *programs Π_1, Π_2 are strongly equivalent iff all standard interpretations of σ'_0 satisfy the formula*

$$\left(\bigwedge_{p/n} \mathcal{A}(p/n)\right) \to (\gamma\tau^*\Pi_1 \leftrightarrow \gamma\tau^*\Pi_2), \tag{7}$$

where the conjunction extends over all predicate symbols p/n that occur in Π_1 or in Π_2.

This theorem differs from the incorrect assertion mentioned in the introduction [12, Theorem 3] by requiring the interpretations to be standard. It shows that strong equivalence between Π_1 and Π_2 can be established by proving formula (7) in a first-order theory such that its axioms are satisfied by all standard interpretations. This is how Heuer's procedure operates. It translates formula (7) into the TPTP language [26] using the algorithm implemented earlier as part of the proof assistant ANTHEM [7]. Then the theorem prover VAMPIRE [14] is invoked to find a proof.

Results similar to the theorem above are due to Lin [23] (his Theorem 1 is about strong equivalence of propositional programs), to Pearce et al. [25] (their Theorem 6(iii) is about strong equivalence of propositional formulas), and to Ferraris et al. [9] (their Theorem 9 is about strong equivalence of first-order formulas without arithmetic).

Theorem 2 below shows that the assertion of Theorem 1 will remain true if we replace τ^* by the simpler and more natural translation ν when a "regular" rule [18, Section 2] is translated. (The two main distinctive features of regular rules are that they do not use function symbols / and \, and do not apply arithmetical operations to intervals, as in $X \times (Y \mathinner{..} Z)$.)

For any mini-GRINGO program Π, by $\mu\Pi$ we denote the set consisting of

- the formulas νR for all rules R of Π that are regular [18, Section 2], and
- the formulas $\tau^* R$ for all rules R of Π that are not regular.

The assertion of Theorem 1 will remain true if we replace τ^* in its statement by μ:

Theorem 2. *Mini-*GRINGO *programs Π_1, Π_2 are strongly equivalent iff all standard interpretations of σ'_0 satisfy the formula*

$$\left(\bigwedge_{p/n} \mathcal{A}(p/n)\right) \to (\gamma\mu\Pi_1 \leftrightarrow \gamma\mu\Pi_2), \tag{8}$$

where the conjunction extends over all predicate symbols p/n that occur in Π_1 or in Π_2.

5 Examples

Example 1. Let Π_1 be rule (2), and let Π_2 be the rule

$$q(X) \leftarrow p(X) \wedge \mathit{not}\ \mathit{not}\ q(X). \tag{9}$$

Both rules are regular, so that $\mu\Pi_1$ is (4), and $\mu\Pi_2$ is

$$\forall X(p(X) \wedge \neg\neg q(X) \to q(X)).$$

Then $\gamma\mu\Pi_1$ is

$$\forall X((p(X) \to (q(X) \vee \neg q'(X))) \wedge (p'(X) \to (q'(X) \vee \neg q'(X)))),$$

which is (classically) equivalent to

$$\forall X(p(X) \to (q(X) \vee \neg q'(X))); \tag{10}$$

$\gamma\mu\Pi_2$ is

$$\forall X((p(X) \wedge \neg\neg q'(X) \to q(X)) \wedge (p'(X) \wedge \neg\neg q'(X) \to q'(X))),$$

which is equivalent to

$$\forall X(p(X) \wedge q'(X) \to q(X)))$$

and furthermore to (10). Thus the consequent of (8) is in this case logically valid, and the programs are strongly equivalent by Theorem 2.

Example 2. Let Π_1 be rule (5), and let Π_2 be the rule

$$q(X, Y) \leftarrow p(X, Y - \overline{1}). \tag{11}$$

Both rules are regular, so that $\mu\Pi_1$ is (6), and $\mu\Pi_2$ is

$$\forall XN(p(X, N - \overline{1}) \to q(X, N)).$$

Then $\gamma\mu\Pi_1$ is

$$\forall XN((p(X, N) \to q(X, N + \overline{1})) \wedge (p'(X, N) \to q'(X, N + \overline{1}))),$$

and $\gamma\mu\Pi_2$ is

$$\forall XN((p(X, N - \overline{1}) \to q(X, N)) \wedge (p'(X, N - \overline{1}) \to q'(X, N))).$$

The equivalence $\gamma\mu\Pi_1 \leftrightarrow \gamma\mu\Pi_2$ is a logical consequence of the formulas

$$\forall N((N - \overline{1}) + \overline{1} = N) \text{ and } \forall N((N + \overline{1}) - \overline{1} = N),$$

which are satisfied by all standard interpretations. The programs are strongly equivalent by Theorem 2.

In both examples above, we did not refer to the antecedent of implication (8); in each case, all standard interpretations satisfy the consequent. Strong equivalence between the program

$$\begin{array}{l} q \leftarrow p, \\ \leftarrow p \wedge \neg q \end{array}$$

and its first rule is a case when the presence of the antecedents in implications (7) and (8) is essential. This example is due to Lin [23, Section 2].

6 Logic of here-and-there with arithmetic

Proofs of Theorems 1 and 2 are derived from a lemma that refers to an extension of the deductive system *HTA*. This extension, denoted by HTA^ω, can be described as the result of adding a few axiom schemas and inference rules to intuitionistic logic with equality for the signature σ_0.

The list of additional axioms includes the Hosoi axiom schema [13]

$$F \vee (F \to G) \vee \neg G$$

and the schema SQHT [22]

$$\exists X(F(X) \to \forall X\, F(X)).$$

It includes also the formulas

$$t_1 \prec t_2$$

where $\prec$ is one of comparison symbols (1), and t_1, t_2 are precomputed terms that satisfy the condition $t_1 \prec t_2$;

$$\neg(t_1 \prec t_2)$$

where $\prec$ is $=$ or one of comparison symbols (1), and t_1, t_2 are precomputed terms that do not satisfy the condition $t_1 \prec t_2$; and

$$\overline{m+n} = \overline{m} + \overline{n}, \qquad \overline{m-n} = \overline{m} - \overline{n}, \qquad \overline{m \cdot n} = \overline{m} \times \overline{n}$$

for all integers m, n.

The additional inference rules are "omega-rules" with infinitely many premises:

$$\frac{F(t) \text{ for all precomputed terms } t}{\forall X\, F(X)}$$

where X is a general variable, and

$$\frac{F(\overline{n}) \text{ for all integers } n}{\forall N\, F(N)} \tag{12}$$

where N is an integer variable [6, Section 5.3].

Main Lemma. *Let Π_1, Π_2 be mini-*GRINGO *programs, and let F_i $(i = 1, 2)$ be a sentence over σ_0 that is equivalent to $\tau^* \Pi_i$ in HTA^ω. Programs Π_1, Π_2 are strongly equivalent iff every standard model of $\mathcal{A}$ satisfies $\gamma F_1 \leftrightarrow \gamma F_2$, where $\mathcal{A}$ denotes the set of formulas $\mathcal{A}(p/n)$ for all predicate symbols p/n.*

Conclusion

Theorem 2 shows that Heuer's procedure can be modified by replacing $\tau^* R$ with the simpler translation νR when the rule R is regular. We expect that this modification will make the system easier to use, and we plan to verify this conjecture in collaboration with researchers at the University of Potsdam.

References

1. Bochman, A., Lifschitz, V.: Yet another characterization of strong equivalence. In: Technical Communications of the 27th International Conference on Logic Programming (ICLP). pp. 11–15 (2011)
2. Cabalar, P., Ferraris, P.: Propositional theories are strongly equivalent to logic programs. Theory and Practice of Logic Programming 7 (2007)
3. Chen, Y., Lin, F., Li, L.: SELP — a system for studying strong equivalence between logic programs. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning. pp. 442–446 (2005)
4. De Jongh, D., Hendriks, L.: Characterization of strongly equivalent logic programs in intermediate logics. Theory and Practice of Logic Programming 3, 259–270 (2003)
5. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case. In: Proceedings of AAAI Conference on Artificial Intelligence (AAAI). pp. 695–700 (2005)
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 (to appear) (2023)
7. Fandinno, J., Lifschitz, V., Lühne, P., Schaub, T.: Verifying tight logic programs with Anthem and Vampire. Theory and Practice of Logic Programming 20 (2020)
8. Ferraris, P.: On modular translations and strong equivalence. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). pp. 79–91 (2005)
9. Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription. Artificial Intelligence 175, 236–263 (2011)
10. Harrison, A., Lifschitz, V., Pearce, D., Valverde, A.: Infinitary equilibrium logic and strong equivalence. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). pp. 398–410 (2015)
11. Harrison, A., Lifschitz, V., Pearce, D., Valverde, A.: Infinitary equilibrium logic and strongly equivalent logic programs. Artificial Intelligence 246, 22–33 (May 2017)
12. Heuer, J.: Automated verification of equivalence properties in advanced logic programs (2020), Bachelor Thesis, University of Potsdam
13. 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)
14. Kovaćs, L., Voronkov, A.: First-order theorem proving and Vampire. In: International Conference on Computer Aided Verification. pp. 1—35 (2013)
15. Lee, J., Palla, R.: Yet another proof of the strong equivalence between propositional theories and logic programs. In: Working Notes of the Workshop on Correspondence and Equivalence for Nonmonotonic Theories (2007)
16. Lierler, Y., Lifschitz, V.: Termination of grounding is not preserved by strongly equivalent transformations. In: Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR) (2011)
17. Lifschitz, V.: Here and there with arithmetic. Theory and Practice of Logic Programming (2021)
18. Lifschitz, V.: Transforming gringo rules into formulas in a natural way. In: Proceedings of European Conference on Artificial Intelligence (2021)

19. Lifschitz, V.: Strong equivalence of logic programs with counting. Theory and Practice of Logic Programming 22 (2022)
20. 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)
21. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic 2, 526–541 (2001)
22. 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)
23. Lin, F.: Reducing strong equivalence of logic programs to entailment in classical propositional logic. In: Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR). pp. 170–176 (2002)
24. Lin, F., Chen, Y.: Discovering classes of strongly equivalent logic programs. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI) (2005), to appear
25. Pearce, D., Tompits, H., Woltran, S.: Characterising equilibrium logic and nested logic programs: Reductions and complexity. Theory and Practice of Logic Programming 9, 565–616 (2009)
26. Sutcliffe, G.: The TPTP problem library and associated infrastructure. Journal of Automated Reasoning 59(4), 483–502 (2017)
27. Turner, H.: Strong equivalence made easy: nested expressions and weight constraints. Theory and Practice of Logic Programming 3(4,5), 609–622 (2003)