Formal Methods for Answer Set Programming

Jan Heuer[1]

[1]University of Potsdam, Germany

Abstract

The increased use of Answer Set Programming (ASP) in industrial applications motivates the need for formal verification methods for ASP. This research builds on the ANTHEM system, which verifies correctness and equivalence of logic programs written in a subset of the CLINGO input language. A first limitation is the restriction on the supported input language of ANTHEM. A second limitation concerns the proof search, which in general is undecidable (as programs may contain arithmetic) and thus may (practically) not terminate. In such cases it is unclear whether this is due to the complexity of the proof task or due to non-equivalence. This paper discusses ongoing work addressing these limitations as well as outlines possible future research directions.

1. Introduction

Answer Set Programming (ASP; [1]) is increasingly being used in industrial applications such as planning, scheduling and configuration [2]. This motivates the need for methods and tools for the formal verification of ASP. The ANTHEM [3] system has recently been developed to address this requirement. The system supports two kinds of verification: verification against a given specification and verification of equivalence of two programs. To do so ANTHEM takes as input logic programs written in a subset of the input language of CLINGO [4]. The verification reduces the equivalence claim to a problem in first-order logic. ANTHEM then employs an automated theorem prover to verify the problem.

While ANTHEM significantly exceeds the capability of earlier systems for verification it still faces limitations. A first limitation are the restrictions on the input language. Important features such as non-tight rules and aggregates that are widely used in applications are not yet supported. A second limitation concerns the proof search. The underlying problem is undecidable in general, as MINI-GRINGO programs may contain arithmetic. Consequently, proof search may not terminate in practice. While the system is able to verify some problems quickly, it does not terminate for more complex equivalences in reasonable time. Furthermore, it is not possible to distinguish a complex but provable equivalence from a non-equivalence, as in both cases proof search fails to terminate.

This paper first provides further background on the ANTHEM system and its underlying theory. We then outline ongoing research addressing its limitations and discuss future research avenues.

2. Background

As mentioned in Section 1, recent work on verification in ASP is focused on the ANTHEM system [3]. While other systems have been proposed in the past [5, 6, 7, 8], ANTHEM supersedes these systems in terms of the supported input language. The input language of ANTHEM, MINI-GRINGO, is a first-order language supporting a large subset of the input language of CLINGO [4], including simple choices and arithmetic. The formal semantics of CLINGO is given in [9]. It is defined in terms of a transformation of first-order programs to infinitary propositional formulas. The definition of stable models for such formulas is due to [10], which extends the definition for (standard) propositional formulas [11].

As the basis for ANTHEM, an alternative to the transformation to infinitary propositional formulas is used. This alternative instead transforms first-order programs into first-order formulas [12]. The first-order logic targeted by ANTHEM is a two-sorted logic containing arithmetic. As the verification tasks

Joint Proceedings of the Workshops and Doctoral Consortium of the 42nd International Conference on Logic Programming (ICLP 2026) co-located with the Federated Logic Conference (FLoC 2026).

jan.heuer@uni-potsdam.de (J. Heuer)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-03

CEUR-WS.org/Vol-4311/DC_short_07.pdf

ANTHEM considers two equivalence relations: strong equivalence [12] and external equivalence [13, 14]. The general approach is to encode these problems into an equivalent representation in (classical) first-order logic. In this way the task of verifying an equivalence between logic programs is reduced to the task of proving that some first-order formula holds. Such a proof can then be found in a (semi-) automated way using automated theorem provers. Specifically, ANTHEM uses the typed form of the TPTP language [15] as its output format and then integrates with the VAMPIRE theorem prover [16].

Two programs are strongly equivalent if they have the same stable models in conjunction with any program [17]. In this way strong equivalence ensures that one program can be replaced by another in any arbitrary context. The method employed by ANTHEM to verify strong equivalence relies on the characterization that two programs are strongly equivalent iff they are equivalent in the logic of here-and-there [17]. In order to make use of provers for classical logic, ANTHEM then uses a transformation that encodes the HT semantics into classical logic [18, 19, 20].

In many cases strong equivalence is a too strict relation and fails to take into account how programs are meant to be used. This is addressed by external equivalence by taking into account the intended inputs of a program as well as what part of a program is considered its output. It corresponds to relativized uniform equivalence with projection [21]. The specification of output predicates allows programs to have auxiliary predicates that are not taken into account in the equivalence. The verification of external equivalence is based on completion. Completion [22] characterizes the stable models of a program by a set of classical formulas. It only accurately captures the stable models of tight programs. For non-tight programs additions such as loop formulas [23, 24] are necessary to forbid circular derivations. For the MINI-GRINGO language completion was defined in [25]. Using completion alone therefore limits the verification to tight programs. Recent work showed that this limitation can be weakened to so-called locally tight programs [26]. Intuitively, while tightness forbids any positive recursion, local tightness allows terminating recursion. The approach for verifying external equivalence also makes it possible to verify the equivalence of a program to a specification written in the language of first-order logic.

The current ANTHEM system has two main limitations. First, while its input language supersedes prior systems, it still lacks features common in many applications. Second, while ANTHEM can successfully verify some equivalences, proof search frequently does not terminate in a practical amount of time. As the underlying problem is undecidable in general, proof search may not terminate at all. In this case it is unclear whether this is due to the complexity of the proof task or non-equivalence of the programs.

For the first limitation, work has already been done to support conditionals [27]. Extending the language by aggregate has also been considered in several works [28, 29, 30]. An extension to support function symbols is proposed in [31]. Lifting the limitation of (local) tightness for verifying external equivalence is part of my ongoing work discussed later.

For the second limitation, some recent work aims to reduce the proof complexity. The transformation from programs to first-order formulas from [12] often produces formulas that are much more complex than expected. An alternative transformation [32] only covers MINI-GRINGO programs conforming to certain syntactic structures at the benefit of producing simpler formulas. Furthermore, formula simplifications have been implemented in ANTHEM. Another addition is the feature of proof outlines [3]. A proof outline is a series of intermediate lemmas that the user provides to ANTHEM to help in the proof search. Instead of proving the target equivalence directly, ANTHEM instead proves these lemmas first as intermediate steps. This enabled the verification of some more complex problems [33], but requires (significant) manual effort to formulate proof outlines. For distinguishing non-equivalent cases many of the earlier approaches [5, 6, 7] to verify propositional programs were able to generate counterexamples for non-equivalent cases. Adapting one such method to generate counterexamples in the setting of ANTHEM is part of my ongoing work.

3. Research Outline

In my research I plan to address the current limitations discussed above. My current contributions are: (i) extending ordered completion [34] to MINI-GRINGO including an approach to capture infinite

stable models [35], (ii) generalizing tightening [36] to MINI-GRINGO, and (iii) developing a method for the automated generation of counterexamples for external equivalence supporting programs with aggregates. The following subsections discuss each in detail and outline planned future work.

3.1. Language Extensions

Tightness. As discussed, currently the verification of external equivalence is limited to programs that are (locally) tight. Several methods to extend or modify completion to accommodate non-tight programs have been considered in the literature. They can be divided into two categories. First, adaptions of completion by extension with loop formulas [23, 37, 24] or by addition of order formulas [38, 39, 34]. Second, reduction of arbitrary program to classes of programs whose stable models are captured by completion such as locally tight programs [36], inherently tight programs [40], or tight programs [41, 42, 43]. As part of ongoing work I am adapting two of the approaches to the input language of ANTHEM.

The first approach, *ordered completion* [34], extends the usual completion by introducing auxiliary predicates encoding the derivation order of atoms. In this way it is possible to ensure non-circular derivations even in the presence of recursive (i.e., non-tight) rules. In my ongoing work I have already extended ordered completion to the MINI-GRINGO language [35]. While the definition of ordered completion from [34] only captures finite stable models, I have also developed an alternative encoding of the derivation order to capture infinite stable models.

The second approach, *tightening* [36], transforms programs into locally tight programs such that the stable models under projection to the original vocabulary are equal. I have also generalised the definition of this transformation to the MINI-GRINGO language.

While these two approaches make it possible to capture the semantics of non-tight programs in first-order logic, there are still limitations for capturing equivalences. Both approaches make the derivation order of atoms in the stable model explicit. Thus, equivalences of programs deriving the atoms of their stable models in different orders are not captured. The idea of private predicates from external equivalence offers a solution to this by considering the derivation order as private to each program. However, in general the characterization of external equivalence is a second-order formula. It can only be reduced to first-order logic under certain conditions that are not fulfilled here. It may thus be of interest to investigate the use of higher-order theorem provers [44, 45].

Other Language Extensions. As mentioned in Section 2, ANTHEM only supports a subset of the input language of CLINGO. The main language feature of interest currently not supported are aggregates. Theoretical work extending MINI-GRINGO with aggregates [29, 28, 30] relies on features not directly available in the target language of ANTHEM (functions operating on sets) or require the addition of an infinite number of predicates. Besides aggregates (and conditionals which have already been implemented), there are several smaller features missing such as pools, disjunctions, negative heads and function symbols. The latter has been considered in theoretical work [31]. I have already considered pools (a syntactic shortcut similar to interval terms) in past work [19] and plan to integrate them in future work.

Another interesting direction for language extensions are hybrid ASP systems [46, 47]. Such systems combine ASP with external theories in order to improve performance (by avoiding the usual grounding bottleneck). Supporting theory extensions would make it possible to compare a program using such extensions to a version using plain ASP only.

3.2. Non-Termination of Proof Search

Non-Equivalence. In my current work I have been developing a method to automatically generate counterexamples for external equivalence problems. Such a method can help to distinguish the case where proof search does not terminate (in reasonable time) because the programs are non-equivalent from the case where they are equivalent but the proof is complex. The method we are developing is based on an approach for verifying/generating counterexamples for propositional programs [48, 49].

We have modified their approach to consider programs with variables and aggregates in the context of external equivalence. The approach relies on a transformation to ASP that generates possible inputs and checks whether the considered programs have differing output. As this approach relies on ASP itself, usual difficulties such as tightness or aggregates are much easier to handle. Previous work on such ASP-based methods that considers private predicates [49] only allowed restricted use of them. We have developed a modification that allows for unrestricted use of private predicates in the programs.

First experiments using an implementation of this approach have shown that we can successfully generate counterexamples for non-equivalent problems. This method provides a complementary approach to run alongside ANTHEM's verification. Whichever procedure concludes first then determines the result: a counterexample witnesses non-equivalence, while a proof witnesses equivalence. In this way the combination gives definitive results for many problems for which proof search alone does not terminate. Besides extending the language supported by this method, future work is concerned with supporting the generation of counterexamples for strong equivalence based on [7].

Proof Complexity. For the case of equivalent programs that require complex proofs several directions have already been considered in the literature. As mentioned in Section 2, some formula simplifications are successfully used in ANTHEM to improve performance. Extending and formalizing the included simplifications, particularly those exploiting ANTHEM's two-sorted logic, is a subject of future work.

While the aforementioned feature of proof outlines in ANTHEM has made it possible to verify certain problems, developing such proof outlines is a highly complex task. Integrating ANTHEM with interactive theorem provers could be a way to make this task easier by taking advantage of existing tooling.

Another option is to investigate the integration of ANTHEM with different automated theorem provers. I have already carried out such an analysis in past work [19] which revealed CVC5 [50] as a promising addition. As noted in Section 3.1 the use of higher-order theorem provers may also be of interest. The recent development of an automated prover for the logic of here-and-there [51] may be beneficial for verifying strong equivalence.

Declaration on Generative AI

During the preparation of this work, the author used Claude in order to: Abstract drafting, Paraphrase and Reword. After using these tools, the author reviewed and edited the content as needed and take full responsibility for the publications's content.

References

[1] V. Lifschitz, Answer Set Programming, Springer, 2019.

[2] A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, E. Teppan, Industrial applications of answer set programming, Künstliche Intelligenz 32 (2018) 165–176.

[3] J. Fandinno, Z. Hansen, Y. Lierler, C. Glinzer, J. Heuer, T. Schaub, T. Stolzmann, V. Lifschitz, ANTHEM 2.0: Automated reasoning for answer set programming, Theory and Practice of Logic Programming 25 (2025) 668–684.

[4] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, Theory and Practice of Logic Programming 19 (2019) 27–82.

[5] Y. Chen, F. Lin, L. Li, SELP—a system for studying strong equivalence between logic programs, in: [52], 2005, pp. 442–446.

[6] J. Oetsch, M. Seidl, H. Tompits, S. Woltran, cct: A correspondence-checking tool for logic programs under the answer-set semantics, Proceedings of the Tenth European Conference on Logics in Artificial Intelligence (JELIA'06), Springer, 2006, pp. 502–505.

[7] T. Janhunen, E. Oikarinen, LPEQ and DLPEQ — translators for automated equivalence testing of logic programs, Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04), Springer, 2004, pp. 336–340.

[8] T. Eiter, W. Faber, P. Traxler, Testing strong equivalence of datalog programs - implementation and examples, in: [52], 2005, pp. 437–441.
[9] M. Gebser, A. Harrison, R. Kaminski, V. Lifschitz, T. Schaub, Abstract Gringo, Theory and Practice of Logic Programming 15 (2015) 449–463.
[10] M. Truszczyński, Connecting first-order ASP and the logic FO(ID) through reducts, Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz, Springer, 2012, pp. 543–559.
[11] P. Ferraris, Answer sets for propositional theories, in: [52], 2005, pp. 119–131.
[12] V. Lifschitz, P. Lühne, T. Schaub, Verifying strong equivalence of programs in the input language of GRINGO, Proceedings of the Fifteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'19), Springer, 2019, pp. 270–283.
[13] J. Fandinno, V. Lifschitz, P. Lühne, T. Schaub, Verifying tight logic programs with anthem and vampire, Theory and Practice of Logic Programming 20 (2020) 735–750.
[14] J. Fandinno, Z. Hansen, Y. Lierler, V. Lifschitz, N. Temple, External behavior of a logic program and verification of refactoring, Theory and Practice of Logic Programming 23 (2023) 933–947.
[15] G. Sutcliffe, The TPTP problem library and associated infrastructure: The FOF and CNF parts, v3.5.0, Journal of Automated Reasoning 43 (2009) 337–362.
[16] F. Bártek, A. Bhayat, R. Coutelier, M. Hajdu, M. Hetzenberger, P. Hozzová, L. Kovács, J. Rath, M. Rawson, G. Reger, M. Suda, J. Schoisswohl, A. Voronkov, The Vampire diary, Computer Aided Verification, Springer, 2025, pp. 57–71.
[17] V. Lifschitz, D. Pearce, A. Valverde, Strongly equivalent logic programs, ACM Transactions on Computational Logic 2 (2001) 526–541.
[18] D. Pearce, H. Tompits, S. Woltran, Encodings for equilibrium logic and logic programs with nested expressions, Proceedings of the Tenth Portuguese Conference on Artificial Intelligence (EPIA'01), Springer, 2001, pp. 306–320.
[19] J. Heuer, Automated Verification of Equivalence Properties in Advanced Logic Programs, Bachelor's thesis, University of Potsdam, 2020. URL: https://arxiv.org/abs/2310.19806.
[20] J. Fandinno, V. Lifschitz, On Heuer's procedure for verifying strong equivalence, Proceedings of the Eighteenth European Conference on Logics in Artificial Intelligence (JELIA'23), Springer, 2023.
[21] J. Oetsch, H. Tompits, Program correspondence under the answer-set semantics: The non-ground case, Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08), Springer, 2008, pp. 591–605.
[22] K. Clark, Negation as failure, Logic and Data Bases, Plenum Press, 1978, pp. 293–322.
[23] F. Lin, Y. Zhao, ASSAT: computing answer sets of a logic program by SAT solvers., Artificial Intelligence 157 (2004) 115–137.
[24] J. Lee, Y. Meng, First-order stable model semantics and first-order loop formulas, Journal of Artificial Intelligence Research 42 (2011) 125–180.
[25] A. Harrison, V. Lifschitz, D. Raju, Program completion in the input language of GRINGO, Theory and Practice of Logic Programming 17 (2017) 855–871.
[26] J. Fandinno, V. Lifschitz, N. Temple, Locally tight programs, Theory and Practice of Logic Programming 24 (2024) 942–972.
[27] Z. Hansen, Y. Lierler, SM-based semantics for answer set programs containing conditional literals and arithmetic, Proceedings of the Twenty-seventh International Symposium on Practical Aspects of Declarative Languages (PADL'25), Springer, 2025, pp. 71–87.
[28] J. Fandinno, Z. Hansen, Recursive aggregates as intensional functions in answer set programming: Semantics and strong equivalence, 2025, pp. 14893–14901.
[29] J. Fandinno, Z. Hansen, Y. Lierler, Axiomatization of non-recursive aggregates in first-order answer set programming, Journal of Artificial Intelligence Research 80 (2024) 977–1031.
[30] J. Fandinno, V. Lifschitz, Deductive systems for logic programs with counting, Theory and Practice of Logic Programming 25 (2025) 924–964.
[31] J. Fandinno, Y. Lierler, V. Lifschitz, A normal form for rules containing arithmetic operations, in: Proceedings of the Twenty-third International Conference on Principles of Knowledge Representation and Reasoning (KR'26), 2026.

[32] V. Lifschitz, Transforming gringo rules into formulas in a natural way, Proceedings of the Seventeenth European Conference on Logics in Artificial Intelligence (JELIA'21), Springer, 2021, pp. 421–434.
[33] V. Lifschitz, An experiment with Anthem: Semantic equivalence of tiling programs, Proceedings of the Nineteenth European Conference on Logics in Artificial Intelligence (JELIA'25), Springer, 2025, pp. 357–363.
[34] V. Asuncion, F. Lin, Y. Zhang, Y. Zhou, Ordered completion for first-order logic programs on finite structures, Artificial Intelligence 177-179 (2012) 1–24.
[35] J. Heuer, Ordered Completion for Non-Locally Tight Mini-Gringo Programs, Master's thesis, University of Potsdam, 2025. URL: https://arxiv.org/abs/2504.14252.
[36] M. Wallace, Tight, consistent, and computable completions for unrestricted logic programs, Journal of Logic Programming 15 (1993) 243–273.
[37] Y. Chen, F. Lin, Y. Wang, M. Zhang, First-order loop formulas for normal logic programs, Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06), AAAI Press, 2006, pp. 298–307.
[38] R. Ben-Eliyahu, R. Dechter, Propositional semantics for disjunctive logic programs, Annals of Mathematics and Artificial Intelligence 12 (1994) 53–87.
[39] I. Niemelä, Stable models and difference logic, Annals of Mathematics and Artificial Intelligence 53 (2008) 313–329.
[40] F. Lin, J. Zhao, On tight logic programs and yet another translation from normal logic programs to propositional logic, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03), Morgan Kaufmann Publishers Inc., 2003, pp. 853–858.
[41] T. Janhunen, Representing normal programs with clauses, Proceedings of the Sixteenth Eureopean Conference on Artificial Intelligence (ECAI'04), IOS Press, 2004, pp. 358–362.
[42] T. Janhunen, I. Niemelä, Compact translations of non-disjunctive answer set programs to propositional clauses, Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning, Springer, 2011, pp. 111–130.
[43] T. Janhunen, I. Niemelä, M. Sevalnev, Computing stable models via reductions to difference logic, Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09), Springer, 2009, pp. 142–154.
[44] C. Benzmüller, D. Miller, Automation of higher-order logic, Computational Logic, North-Holland, 2014, pp. 215–254.
[45] G. Sutcliffe, The TPTP problem library and associated infrastructure, Journal of Automated Reasoning 59 (2017) 483–502.
[46] M. Banbara, B. Kaufmann, M. Ostrowski, T. Schaub, Clingcon: The next generation, Theory and Practice of Logic Programming 17 (2017) 408–461.
[47] T. Janhunen, R. Kaminski, M. Ostrowski, S. Schellhorn, P. Wanko, T. Schaub, Clingo goes linear constraints over reals and integers, Theory and Practice of Logic Programming 17 (2017) 872–888.
[48] T. Janhunen, E. Oikarinen, Testing the equivalence of logic programs under stable model semantics, Proceedings of the Eighth European Conference on Logics in Artificial Intelligence (JELIA'02), Springer, 2002, pp. 493–504.
[49] E. Oikarinen, T. Janhunen, A translation-based approach to the verification of modular equivalence, Journal of Logic and Computation 19 (2009) 591–613.
[50] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, Y. Zohar, cvc5: A versatile and industrial-strength SMT solver, Proceedings of the Twenty-eighth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'22), Springer, 2022, pp. 415–442.
[51] J. Otten, T. Schaub, Implementing the first-order logic of here and there, Proceedings of the Forty-first International Conference on Logic Programming (ICLP'25), 2025, pp. 453–468.
[52] C. Baral, G. Greco, N. Leone, G. Terracina (Eds.), Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05), Springer, 2005.