Formal Verification of Answer Set Programs Containing Advanced Language Constructs

Zachary Hansen[1]

[1]University of Nebraska Omaha (UNO), Omaha, NE, 68106, USA

Abstract

This research summary investigates strategies for the formal verification of a broad class of logic programs in the Answer Set Programming (ASP) paradigm. In particular, it extends translation-based semantics for ASP programs to the advanced language constructs known as aggregates and conditional literals. This allows ASP practitioners to write proofs of correctness in a modular fashion, and, in many cases, enables them to verify their programs automatically using resolution theorem provers. The theoretical contributions of this work are supported by a proof assistant for ASP called ANTHEM.

Keywords

Logic programming, formal methods, software verification, automated reasoning

1. Introduction

Answer Set Programming (ASP) is a declarative programming paradigm initially developed to address challenging problems in AI, such as the frame problem [1]. Since then, it has become an increasingly popular approach to modeling and solving combinatorial search and optimization problems [2, 3]. ASP is by now employed in numerous high-consequence and challenging domains such as explainable donor-patient matching [4], space shuttle decision support systems [5, 6], and train scheduling [7]. This underscores the need for *formal methods for ASP*; the research described here contributes to this topic with the goal of making (semi) automated verification of realistic ASP programs possible.

In this work, I explore the close relationship ASP (in particular, fragments of the language supported by the CLINGO solver) shares with other logical formalisms, such as the logic of here-and-there, as a means to rigorously verify answer set programs. I have worked on extending a research agenda focused on

1. transforming ASP programs into second-order formulas to facilitate writing proofs of correctness by hand; and
2. reducing these second-order representations into first-order ones to automate verification with theorem provers.

In particular, the focus is on developing translation-based semantics for the advanced language constructs known as *aggregates* and *conditional literals*. Supporting these widely used constructs allows ASP practitioners to verify a broader class of programs than the current state of the art.

Listing 1: A slightly modified program [8] solving the Frame problem.

```
in(P, R, 0) :- in0(P, R).
in(P, R, T+1) :- goto(P, R, T).
{in(P, R, T+1)} :- in(P, R, T), T = 0..h-1.
:- in(P, R1, T), in(P, R2, T), R1 != R2.
in_building(P, T) :- in(P, R, T), room(R).
:- not in_building(P, T), person(P), T = 0..h-1.
```

zachhansen@unomaha.edu (Z. Hansen)
0000-0002-8447-4048 (Z. Hansen)

2. Background

In this paper, we consider variations of a language of logic programs called MINI-GRINGO [9, 10, 11, 12, 13, 8] in which terms may contain arithmetic operations

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

and, consequently, can have 0 (as in the case of $1/0$), 1 (as in the case of $1 + 3$), or many (as in the case of $1..3$) values. This language supports *atoms* of the form $p(\mathbf{t})$ where p is a predicate symbol and $\mathbf{t}$ is a tuple of terms, (basic) *literals* of the form A or $not\ A$ or $not\ not\ A$ where A is an atom and not is the default negation operator, and *comparisons* ($<, \leq, >, \geq, =$) between terms. We often refer to basic literals and comparisons as *atomic formulas.* MINI-GRINGO supports normal and choice rules along with constraints, but not rules with disjunctive heads. In the following, we investigate the task of extending MINI-GRINGO and the associated results on formal verification with conditional literals and aggregates.

Aggregates and Conditional Literals These advanced language constructs improve the expressivity of ASP, but their behavior is more challenging to precisely define than basic atoms and literals. The semantics of aggregates in particular have been widely studied [14, 15, 16, 17, 18, 19, 20, 21, 22], and in the presence of certain forms of recursion through aggregates these characterizations can diverge. An *aggregate element* is an expression of the form

$$t_1, \ldots, t_k : l_1, \ldots, l_m \tag{1}$$

where each t_i is a term and each l_i is an atomic formula. An *aggregate atom* has the form $\#\mathrm{op}\{E\} \prec u$ where op is an operation name (typically `count` or `sum`), E is an aggregate element, $\prec$ is a comparison symbol and u is a term.

Conditional literals originated in the LPARSE grounder [23], but are also now supported by CLINGO [24]. They have the form

$$H \ : \ l_1, \ldots, l_m, \tag{2}$$

where H is either a literal, a comparison, or the symbol $\bot$ (denoting falsity) and $l_1, \ldots, l_m$ is a list of atomic formulas.

Traditionally, the semantics of these constructs rely on *grounding*, in which all variables are replaced by variable-free terms. The semantics of the resulting propositional program can then be defined, for example, in terms of reducts. However, this makes it difficult to verify a program modeling some problem in separation from a *grounding context* provided by a concrete instance of the problem. In the following, we explore some alternative semantics that bypass the need for grounding.

Here-and-there and Equilibrium Logic ASP shares a close relationship with other logical formalisms, in particular, the logic of here-and-there (HT) [25]. This connection has been widely studied in the context of strong equivalence [26], which is a useful property (in the context of program refactoring) that tells us two programs are universally interchangeable. Translating certain classes of ASP programs into their HT "formula representations" provides a way to establish strong equivalence: if the formula representations are HT-equivalent, then the programs are strongly equivalent. Furthermore, HT acts as a monotonic basis for equilibrium logic [27], which can be used to capture the semantics of a broad class of ASP programs. For example, the semantics of CLINGO are defined in terms of a translation τ, which produces infinitary propositional formulas [28] whose equilibrium models coincide with the program's stable models [24]. Having a mathematical specification of this solver's behavior is a powerful tool for formal verification, but it still has certain drawbacks. For instance, infinitary propositional theories cannot be processed by automated theorem provers.

The SM Operator Another translational approach to defining the semantics of ASP is the SM operator [29]. This transformation turns the HT formula representation of a logic program into a second-order theory, the Herbrand models of which capture the stable models of the original program. This approach is closely related to the equilibrium logic approach; in the SM characterization the equilibrium condition is enforced by predicate quantification minimizing belief in certain predicates (referred to as *intensional* predicates). This results in a flexible generalization of the stable model semantics: when the list of intensional predicates includes all predicates from the program (excluding comparisons) then the SM models capture the stable models of the program's formula representation. On the other hand, if the list of intensional predicates is empty, then the SM models capture the classical models of the program's formula representation. The ability to distinguish between intensional and extensional predicates is useful for verifying program modules in which some predicates are taken to be inputs and others are defined by the module in question [30]. The SM operator is also attractive for formal verification purposes because it bypasses a reference to grounding.

First-order Logic While not every ASP program can be reduced to a first-order theory whose classical models coincide with the program's stable models, there is a broad class of programs for which this is possible. The semantics of programs meeting the syntactic restriction of *tightness* can be captured by a process called Clark's Completion [31]. A program is tight if its predicate dependency graph lacks positive cycles [11, 32], intuitively, a predicate should not depend (directly or indirectly) on itself.

3. Goals

The long-term goal of my research is the development of a tool-assisted methodology for formally verifying the correctness of ASP programs. Following the example of successful verification tools from procedural imperative paradigms, such as Spec# [33], I believe the emphasis should be on making such tools accessible enough that verification can be a routine task conducted alongside program development. As such, most of my work is centered around a proof assistant called ANTHEM, developed in collaboration with researchers at the University of Nebraska Omaha (UNO), the University of Texas at Austin, and the University of Potsdam. A component of my eventual dissertation will be extending this system with conditional literals, while the extension with aggregates will be left as a top priority for future work.

4. Current Status and Preliminary Results

During my time in the UNO doctoral program, I have been developing pieces of a translation-based semantics for an extension of MINI-GRINGO that supports conditional literals and aggregates. An overview of our approach can be found in Figure 1. For a language of logic programs (such as MINI-GRINGO extended with conditional literals), we first define a translation (call it τ^*) such that the equilibrium models of $\tau^* P$ coincide with the stable models of program P as computed by CLINGO. We establish this connection by transforming $\tau^* P$ into infinitary propositional formulas and establishing their strong equivalence with the theory τP. For an HT-theory Γ, previously established results [28, 11] have shown the connection between the models of the second-order theory $SM[\Gamma]$ and the equilibrium models of $gr(\Gamma)$ - this connects our SM-based characterization to the behavior of CLINGO. This provides a characterization of the stable model semantics in terms of second-order theories, which is convenient for dividing programs into components and constructing a modular proof of correctness [30]. Finally, to support the task of automated verification with first-order theorem provers, we must establish valid reductions from HT-theories into first-order theories. The two reductions of interest to us are a modified form of Clark's Completion (COMP) [31, 9, 11, 8], and a transformation called γ which embeds the behavior of HT-satisfaction into classical satisfaction of the transformed theory [34, 35, 12]. COMP may be used in the verification of a type of equivalence called *external equivalence* [36, Definition 3], whereas γ is used within strong equivalence verification [12].

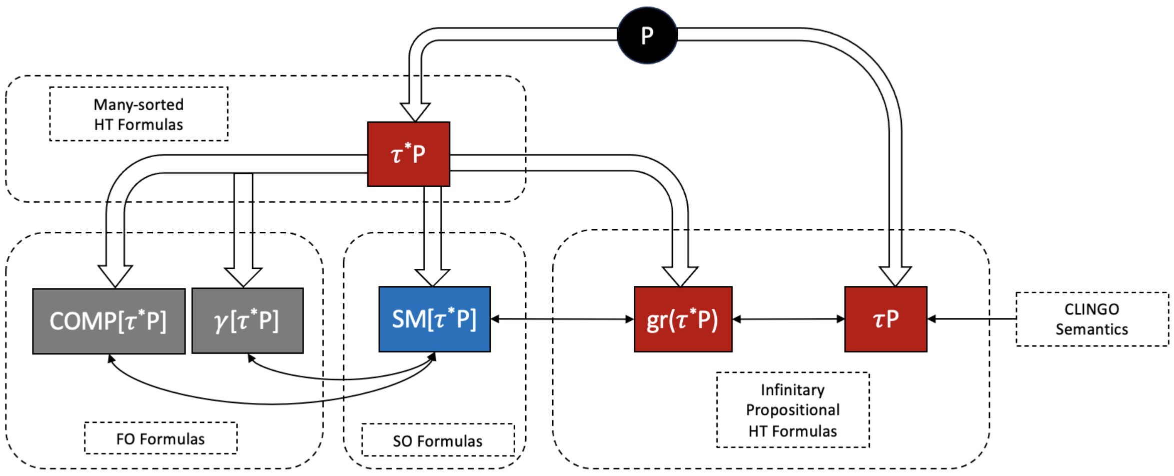

Figure 1: The semantics of a logic program P within 4 different formalisms.

4.1. Results Presented at Previous Doctoral Consortiums

Results on Conditional Literals While ASP rules typically exhibit a very "flat" structure that is more readable than, for example, first-order formulas with many levels of nested connectives, there are a few useful constructs that bring ASP rules closer to that level of expressivity. Conditional literals are one such construct - as demonstrated by our past work [37, 38], conditional literals behave like nested implications within ASP rule bodies. This allows ASP practitioners to concisely express knowledge that may be difficult to otherwise encode. In particular, they excel at expressing existential constraints, e.g. that a property must hold *for some* element of a domain. For example, within Listing 1 rules 5-6 require that, for every (person, time) pair (P, T), $in(P, R, T)$ must hold *for some* room R. However, this requires the introduction of an (otherwise unnecessary) auxiliary predicate, $in\_building/2$. From a formal verification perspective this is undesirable, as it can complicate arguments of strong equivalence. A more concise representation is possible with conditional literals; indeed, we can replace rules 5-6 with the constraint

```
:- person(P); T = 0..h-1; not in(P, R, T) : room(R).          (3)
```

In past work we have extended MINI-GRINGO with conditional literals and developed an SM-based semantics for the resulting language [38]. We have shown that these semantics capture the behavior of CLINGO, and demonstrated their utility in verifying programs in a modular way [37, Section 6].

Results on Aggregates In a similar approach as we employed for conditional literals, my collaborators and I have explored translation-based semantics for aggregates that bypass the need for grounding [39, 40, 41, 42]. In this characterization, aggregates are defined as intensional functions on sets of tuples. For an aggregate element of the form (1), the associated sets are constructed from tuples satisfying the conditions $l_1, \ldots, l_m$. In addition to designing a translation κ from ASP programs with aggregates into HT-theories, we developed second-order axioms to define the behavior of sets and aggregate function symbols. For a class of standard interpretations satisfying certain assumptions, models of $SM[\kappa\Pi]$ satisfying these aggregate axioms are in one-to-one correspondence with the stable models of Π. For tight programs with finite aggregates, we additionally showed that the second-order semantics can be replaced with a first-order characterization using completion and first-order axioms [41]. By treating the function symbols used to construct sets as *intensional functions* [43], we were able to characterize [42] the behavior of two divergent[1] aggregate semantics, namely the FLP-reduct [44, 19] of DLV and the FT-reduct [45] of CLINGO. An interesting finding from this work was that the FLP-reduct based semantics

[1]It should be noted that, to maintain an uncontroversial semantics, the ASP-Core-2 semantics forbids recursion through aggregates, and recent versions of DLV refuse to ground programs containing recursive aggregates.

can be captured by translating the default negation connective "not" into a new negation connective within the logic of here-and-there. Importantly, by developing semantics that accommodated recursion, we were able to define strong equivalence not just between programs written in the same language, but also between CLINGO and DLV programs.

As was the case with conditional literals, one advantage of the proposed semantics is that they enable us to argue the adherence of a program to a natural language specification by dividing the program into modules capturing aspects of the desired behavior. We applied this methodology to programs solving the Graph Coloring and Traveling Salesman problems; the latter example was particularly useful because the modular SM-based semantics enabled us to "recycle" a proof of correctness developed for a Hamiltonian Cycle program used within the Traveling Salesman program [46].

4.2. New Results

The ANTHEM Proof Assistant Recently, my collaborators and I have finished the development of ANTHEM 2.0.[2] This is a proof assistant for ASP that supports several types of verification for MINI-GRINGO programs. It operates by transforming questions of equivalence into tasks for the first-order theorem prover VAMPIRE [47]. The new version of ANTHEM can verify strong equivalence as well as equivalence of external behavior [36]; this type of equivalence is especially useful when we want to confirm that a program has been refactored correctly. The external behavior of a program Π has been preserved in its refactored version Π' if Π and Π' have the same outputs for every (valid/intended) input [48]. Thus, answer set (weak) equivalence is supported as a special case of external equivalence. Additionally, the tool supports specification adherence verification, that is, it can confirm that a tight MINI-GRINGO program implements a specification written in first-order logic.

ANTHEM 2.0 substantially improves upon its predecessors (ANTHEM-SE [35], ANTHEM-1 [11], ANTHEM-P2P [36, 49]) in addition to consolidating their functionalities. It adds powerful new features such as natural translation [50] for strong equivalence tasks, and proof outlines with definitions and inductive lemmas for external equivalence tasks. The extended features of proof outlines have enabled us to verify realistic problems that previous systems could not address in a reasonable amount of time. For example, verifying a refactoring of the Frame problem in Listing 1 can be accomplished with ANTHEM 2.0 using a single inductive lemma.

Strong Equivalence of Conditional Literals In past work [38] we demonstrated that an extension of MINI-GRINGO with conditional literals captured the behavior of CLINGO. That work did not show, however, that previously established results on the strong equivalence of MINI-GRINGO programs were preserved in our extension. In particular, we needed to confirm that the γ transformation as implemented by ANTHEM 2.0 was still valid in the presence of conditional literals. I have recently finished proving that these results are indeed preserved and implemented a prototypical extension of ANTHEM 2.0 with conditional literals (unpublished).

5. Ongoing Directions and Expected Achievements

External Equivalence of Conditional Literals The next step is to extend results on external equivalence of MINI-GRINGO programs to MINI-GRINGO programs with conditional literals. This requires adapting results on tightness to programs with conditional literals. Because the definition of tightness (for arbitrary first-order theories) depends on the number and nature of implications present in a theory [29], the extension of MINI-GRINGO with conditional literals will likely require new syntactic conditions defining tightness. For this we can likely use the definition of predicate dependency graphs for first-order formulas [29, Section 7.3].

[2]The system description can be found at https://arxiv.org/abs/2507.11704.

Semantics for EXTENDED MINI-GRINGO In the following, we refer to the language of MINI-GRINGO extended with aggregates and conditional literals as EXTENDED MINI-GRINGO. The central goal of my eventual dissertation is to provide a semantics for EXTENDED MINI-GRINGO based on the SM operator and demonstrate that these semantics conform to the behavior of CLINGO. Thus far, we have mostly considered the challenging features of this language (arithmetic, aggregates, and conditional literals) in isolation from each other. Integrating these results is a work in progress. Once completed, this will enable ASP practitioners to argue the correctness of EXTENDED MINI-GRINGO programs with respect to natural language specifications in the style of past work. When such programs do not contain aggregates, practitioners will additionally be able to automatically verify strong and external equivalence properties. Extending ANTHEM 2.0 with aggregates, on the other hand, is a serious engineering challenge as VAMPIRE does not natively support set theory in the same way that it supports integer arithmetic.

6. Conclusions and Future Directions

As a concise, human-readable, declarative programming paradigm, ASP is an attractive choice for writing transparent, demonstrably correct programs. Furthermore, past work has shown that relating ASP to other logical formalisms such as first-order and second-order logic is a powerful technique for rigorously verifying programs. The work described here focuses on extending these techniques to programs containing the advanced language constructs known as aggregates and conditional literals. In particular, my proposed dissertation will provide a semantics based on second-order logic for a broad class of CLINGO programs that supports arithmetic, aggregates, and conditional literals. Additionally, I plan to extend the ANTHEM proof assistant with support for conditional literals, enabling automated verification of strong and external equivalence properties for such programs. The most interesting option for future work after this is completed is an extension of ANTHEM with aggregates. This will likely require a partial axiomatization of set theory provided as part of the background theory ANTHEM supplies to VAMPIRE, or the identification of an appropriate higher-order theorem prover as a backend for ANTHEM.

Acknowledgments

I am very grateful to my supervisor, Yuliya Lierler, for 5 years of stellar mentorship. Deep thanks also belong to Jorge Fandinno, Vladimir Lifschitz, and Torsten Schaub for their valuable guidance on all things ANTHEM. Finally, I am grateful to Tobias Stolzmann for their irreplaceable contributions to the implementation of that system.

Declaration on Generative AI

The author has not employed any Generative AI tools.

References

[1] V. Lifschitz, T. Schaub, S. Woltran, Interview with vladimir lifschitz, KI - Künstliche Intelligenz 32 (2018) 213–218.
[2] V. W. Marek, M. Truszczyński, Stable Models and an Alternative Logic Programming Paradigm, Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 375–398. doi:10.1007/978-3-642-60085-2_17.
[3] I. Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, Annals of Mathematics and Artificial Intelligence 25 (1999) 241–273. doi:10.1023/A:1018930122475.

[4] P. Cabalar, B. Muñiz, G. Pérez, F. Suárez, Explainable machine learning for liver transplantation, 2021. URL: https://arxiv.org/abs/2109.13893.

[5] M. Balduccini, M. Gelfond, Model-based reasoning for complex flight systems, in: Proceedings of Infotech@Aerospace (American Institute of Aeronautics and Astronautics), 2005.

[6] M. Balduccini, M. Gelfond, M. Nogueira, R. Watson, M. Barry, An A-Prolog decision support system for the Space Shuttle, in: Working Notes of the AAAI Spring Symposium on Answer Set Programming, 2001.

[7] D. Abels, J. Jordi, M. Ostrowski, T. Schaub, A. Toletti, P. Wanko, Train scheduling with hybrid answer set programming, Theory and Practice of Logic Programming 21 (2021) 317–347. doi:10.1017/S1471068420000046.

[8] J. Fandinno, V. Lifschitz, N. Temple, Locally tight programs, Theory and Practice of Logic Programming 24 (2024) 942–972. doi:10.1017/S147106842300039X.

[9] A. Harrison, V. Lifschitz, D. Raju, Program completion in the input language of GRINGO, Theory and Practice of Logic Programming 17 (2017) 855–871. doi:10.1007/978-1-4684-3384-5_11.

[10] V. Lifschitz, P. Lühne, T. Schaub, Verifying strong equivalence of programs in the input language of GRINGO, in: M. Balduccini, Y. Lierler, S. Woltran (Eds.), Proceedings of the Fifteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'19), volume 11481 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2019, pp. 270–283.

[11] J. Fandinno, V. Lifschitz, P. Lühne, T. Schaub, Verifying tight logic programs with anthem and vampire, Theory and Practice of Logic Programming 5 (2020) 735–750. doi:10.1017/S1471068403001765.

[12] J. Fandinno, V. Lifschitz, On Heuer's procedure for verifying strong equivalence, in: S. Gaggl, M. Martinez, M. Ortiz (Eds.), Proceedings of the Eighteenth European Conference on Logics in Artificial Intelligence (JELIA'23), volume 14281 of *Lecture Notes in Computer Science*, Springer-Verlag, 2023. doi:10.1007/978-3-031-43619-2.

[13] J. Fandinno, V. Lifschitz, Omega-completeness of the logic of here-and-there and strong equivalence of logic programs, Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning 19 (2023) 240–251. doi:10.24963/kr.2023/24.

[14] P. Simons, I. Niemelä, T. Soininen, Extending and implementing the stable model semantics, Artificial Intelligence 138 (2002) 181–234.

[15] A. Dovier, E. Pontelli, G. Rossi, Intensional sets in CLP, in: C. Palamidessi (Ed.), Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings, volume 2916 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 284–299.

[16] N. Pelov, M. Denecker, M. Bruynooghe, Well-founded and stable semantics of logic programs with aggregates, Theory and Practice of Logic Programming 7 (2007) 301–353.

[17] T. C. Son, E. Pontelli, A constructive semantic characterization of aggregates in answer set programming, Theory and Practice of Logic Programming 7 (2007) 355–375.

[18] P. Ferraris, Logic programs with propositional connectives and aggregates, ACM Transactions on Computational Logic 12 (2011) 25:1–25:40.

[19] W. Faber, G. Pfeifer, N. Leone, Semantics and complexity of recursive aggregates in answer set programming, Artificial Intelligence 175 (2011) 278–298. doi:10.1016/j.artint.2010.04.002.

[20] M. Gelfond, Y. Zhang, Vicious circle principle and logic programs with aggregates, Theory and Practice of Logic Programming 14 (2014) 587–601.

[21] M. Gelfond, Y. Zhang, Vicious circle principle, aggregates, and formation of sets in ASP based languages, Artificial Intelligence 275 (2019) 28–77.

[22] P. Cabalar, J. Fandinno, T. Schaub, S. Schellhorn, Gelfond-zhang aggregates as propositional formulas, Artificial Intelligence 274 (2019) 26–43.

[23] T. Syrjänen, Cardinality constraint programs, in: J. J. Alferes, J. Leite (Eds.), Logics in Artificial Intelligence, Springer, Berlin, Heidelberg, 2004, p. 187–199. doi:10.1007/978-3-540-30227-8_18.

[24] M. Gebser, A. Harrison, R. Kaminski, V. Lifschitz, T. Schaub, Abstract gringo, Theory and Practice of Logic Programming 15 (2015) 449–463. doi:10.1017/S1471068415000150.

[25] A. Heyting, Die formalen Regeln der intuitionistischen Logik, in: Sitzungsberichte der Preussischen Akademie der Wissenschaften, Deutsche Akademie der Wissenschaften zu Berlin, 1930, pp. 42–56.

[26] V. Lifschitz, D. Pearce, A. Valverde, Strongly equivalent logic programs, ACM Transactions on Computational Logic 2 (2001) 526–541. doi:10.1145/383779.383783.

[27] D. Pearce, Equilibrium logic, Annals of Mathematics and Artificial Intelligence 47 (2006) 3–41. doi:10.1007/s10472-006-9028-z.

[28] M. Truszczynski, Connecting First-Order ASP and the Logic FO(ID) through Reducts, volume 7265 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2012. URL: http://link.springer.com/10.1007/978-3-642-30743-0_37. doi:10.1007/978-3-642-30743-0_37.

[29] P. Ferraris, J. Lee, V. Lifschitz, Stable models and circumscription, Artificial Intelligence 175 (2011) 236–263. doi:10.1016/j.artint.2010.04.011.

[30] P. Cabalar, J. Fandinno, Y. Lierler, Modular answer set programming as a formal specification language, Theory and Practice of Logic Programming (2020). doi:10.1007/978-1-4471-0043-0.

[31] K. L. Clark, Negation as Failure, Springer US, Boston, MA, 1978, pp. 293–322. doi:10.1007/978-1-4684-3384-5_11.

[32] F. Fages, Consistency of Clark's completion and the existence of stable models, Journal of Methods of Logic in Computer Science 1 (1994) 51–60.

[33] M. Barnett, M. Fähndrich, K. R. M. Leino, P. Müller, W. Schulte, H. Venter, Specification and verification: the spec# experience, Communications of the ACM 54 (2011) 81–91.

[34] D. Pearce, H. Tompits, S. Woltran, Encodings for equilibrium logic and logic programs with nested expressions, in: P. Brazdil, A. Jorge (Eds.), Proceedings of the Tenth Portuguese Conference on Artificial Intelligence (EPIA'01), volume 2258 of *Lecture Notes in Computer Science*, Springer-Verlag, 2001, pp. 306–320. doi:10.1007/3-540-45329-6_31.

[35] 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.

[36] 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. doi:10.1017/S1471068423000200.

[37] Z. Hansen, Y. Lierler, Semantics for conditional literals via the sm operator, in: G. Gottlob, D. Inclezan, M. Maratea (Eds.), Logic Programming and Nonmonotonic Reasoning, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2022, p. 259–272. doi:10.1007/978-3-031-15707-3_20.

[38] Z. Hansen, Y. Lierler, Sm-based semantics for answer set programs containing conditional literals and arithmetic, in: E. Erdem, G. Vidal (Eds.), Practical Aspects of Declarative Languages, Springer Nature Switzerland, Cham, 2025, p. 71–87.

[39] J. Fandinno, Z. Hansen, Y. Lierler, Axiomatization of aggregates in answer set programming, Proceedings of the AAAI Conference on Artificial Intelligence 36 (2022) 5634–5641. doi:10.1609/aaai.v36i5.20504.

[40] J. Fandinno, Z. Hansen, Recursive aggregates as intensional functions, in: Proceedings of Answer Set Programming and Other Computing Paradigms (ASPOCP 2023), 2023, pp. 1–22.

[41] 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. doi:10.1613/jair.1.15786.

[42] J. Fandinno, Z. Hansen, Recursive aggregates as intensional functions in answer set programming: Semantics and strong equivalence, Proceedings of the AAAI Conference on Artificial Intelligence 39 (2025) 14893–14901. doi:10.1609/aaai.v39i14.33633.

[43] M. Bartholomew, J. Lee, First-order stable model semantics with intensional functions, Artificial Intelligence 273 (2019) 56–93. doi:10.1016/j.artint.2019.01.001.

[44] W. Faber, N. Leone, G. Pfeifer, Recursive aggregates in disjunctive logic programs: Semantics and complexity, in: Logics in Artificial Intelligence, Springer, Berlin, Heidelberg, 2004, p. 200–212. URL: https://link.springer.com/chapter/10.1007/978-3-540-30227-8_19. doi:10.1007/978-3-540-30227-8_19.

[45] A. Harrison, V. Lifschitz, Relating two dialects of answer set programming, Theory and Practice of Logic Programming 19 (2019) 1006–1020.

[46] J. Fandinno, Z. Hansen, Y. Lierler, Arguing correctness of asp programs with aggregates, in: G. Gottlob, D. Inclezan, M. Maratea (Eds.), Logic Programming and Nonmonotonic Reasoning, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2022, p. 190–202. doi:10.1007/978-3-031-15707-3_15.

[47] L. Kovács, A. Voronkov, First-order theorem proving and vampire, in: N. Sharygina, H. Veith (Eds.), Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings, volume 8044 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 1–35. doi:10.1007/978-3-642-39799-8_1.

[48] W. F. Opdyke, Refactoring object-oriented frameworks, University of Illinois at Urbana-Champaign, 1992.

[49] Z. Hansen, Anthem-p2p: Automatically verifying the equivalent external behavior of asp programs, Electronic Proceedings in Theoretical Computer Science 385 (2023) 330–332. doi:10.4204/EPTCS.385.34.

[50] V. Lifschitz, Transforming gringo rules into formulas in a natural way, in: W. Faber, G. Friedrich, M. Gebser, M. Morak (Eds.), Proceedings of the Seventeenth European Conference on Logics in Artificial Intelligence (JELIA'21), volume 12678 of *Lecture Notes in Computer Science*, Springer-Verlag, 2021, pp. 421–434. doi:10.1007/978-3-030-75775-5_28.