Finite Groundings for ASP with Functions: A Journey through Consistency (Extended Abstract)

Lukas Gerlach, David Carral, Markus Hecher

Finite Groundings for ASP with Functions: A Journey through Consistency (Extended Abstract)

Lukas Gerlach[1,*], David Carral[2] and Markus Hecher[3]

[1]Knowledge-Based Systems Group, TU Dresden, Nöthnitzer Straße 46, 01062 Dresden, Germany
[2]LIRMM, Inria, University of Montpellier, CNRS, Montpellier, France
[3]MIT Computer Science & Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 32 Vassar St, Cambridge, MA 02139, USA

Abstract

Answer set programming (ASP) is a logic programming formalism used in various areas of artificial intelligence like combinatorial problem solving and knowledge representation and reasoning. It is known that enhancing ASP with function symbols makes basic reasoning problems highly undecidable. However, even in simple cases, state of the art reasoners, specifically those relying on a ground-and-solve approach, fail to produce a result. Therefore, we reconsider consistency as a basic reasoning problem for ASP. We show reductions that give an intuition for the high level of undecidability. These insights allow for a more fine-grained analysis where we characterize ASP programs as "frugal" and "non-proliferous". For such programs, we are not only able to semi-decide consistency but we also propose a grounding procedure that yields finite groundings on more ASP programs with the concept of "forbidden" facts.

Keywords

Answer Set Programming, Rule-Based Reasoning, Knowledge Representation, Computability Theory

1. Introduction

Answer set programming (ASP) [1, 2] is an established nonmonotonic reasoning formalism in the fields of knowledge-representation and reasoning as well as combinatorial problem solving. State-of-the-art ASP systems such as clasp [3] or wasp [4] utilize a ground-and-solve approach to compute answer sets of a given program. In the stage of *grounding*, the given program is instantiated with all relevant terms. Afterwards, the ground program can be efficiently *solved* usually with a SAT solver extended by unfounded set propagation, excluding sets of atoms that lack foundation (i.e. unfounded sets), thereby efficiently computing answer sets.

When function symbols occur in ASP programs, the grounding step will often not terminate and approach an infinite ground program. This is also a problem when using numbers, which we consider to merely be syntactic sugar around a successor function. While this behavior is in principle not incorrect as there are indeed programs that have infinite answer sets or infinitely many answer sets, for many programs it is clear that they only admit finite answer sets and only finitely many of them. In this case, we should always be able to give a large enough but finite ground program. There is a lot of related work discussing function symbols in ASP or logic programming in general [5, 6, 7], also classifying programs according to conditions that guarantee finite groundings [8, 9], and different works on ASP grounding and solving techniques [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]. Still, these approaches do not have support for function symbols as their primary goal. In practice, the problem of infinite ground programs is often counteracted by using auxiliary predicates to artificially limit the size of the grounding.

Figure 1: Wolf, Goat, Cabbage Puzzle

Example 1. *We consider the famous puzzle of a farmer who needs to cross a river with a wolf, a goat, and a cabbage. They may only take one item at a time and must not leave the wolf and the goat or the goat and the cabbage alone since then the former will eat the latter (see Figure 1). One essential part of the considered modeling is a rule as the following together with enough generated atoms, e.g.* $steps(0...100)$.

$$position(X, C, N+1) \leftarrow transport(X, N),$$
$$position(X, B, N), opposite(B, C), steps(N+1).$$

That is, if we guess that item X *is transported in step* N*, then its position is updated to the opposite river bank if we are not out of steps yet. Additional rules are introduced to detect and avoid redundant positions. However, despite the redundancy check, we need to bound (guard) the term* $N+1$*, as otherwise the grounding is infinite.*

The goal of our work is to make the artificial limit on the number of steps obsolete in the above example. We aim to define a grounding procedure that is guaranteed to terminate on programs that can only have finitely many finite answer sets. It turns out that such a procedure must be uncomputable but we also outline a computable relaxation still reflecting the key idea. In this extended abstract, we give a brief overview of the main ideas introduced in our previously published paper [20], where we make the following contributions. (1) We classify ASP programs as *frugal*, if they only have finite answer sets, and *non-proliferous*, if they only have finitely many finite answer sets. (2) We show that both properties are highly undecidable using novel reductions based on reconsiderations of the problem of *consistency*, i.e. checking if an ASP program has an answer set. (3) We propose a novel grounding procedure that ignores *forbidden atoms*. While deciding if an atom is forbidden is also undecidable, we give a sufficient condition able to capture cases like the redundancy check in Example 1.

22nd International Workshop on Nonmonotonic Reasoning, November 2-4, 2024, Hanoi, Vietnam
*Corresponding author.
lukas.gerlach@tu-dresden.de (L. Gerlach); david.carral@inria.fr (D. Carral); hecher@mit.edu (M. Hecher)
https://kbs.inf.tu-dresden.de/lug (L. Gerlach); https://www-sop.inria.fr/members/David.Carral/ (D. Carral); https://www.dbai.tuwien.ac.at/staff/hecher/ (M. Hecher)
0000-0003-4566-0224 (L. Gerlach); 0000-0001-7287-4709 (D. Carral); 0000-0003-0131-6771 (M. Hecher)

2. Preliminaries

We define $\texttt{Preds}$, $\texttt{Funs}$, $\texttt{Cons}$, and $\texttt{Vars}$ to be mutually disjoint and countably infinite sets of predicates, function symbols, constants, and variables, respectively. Every $s \in \texttt{Preds} \cup \texttt{Funs}$ is associated with some *arity* $\mathsf{ar}(s) \geq 0$. For every $i \geq 0$, both $\texttt{Preds}_i = \{P \in \texttt{Preds} \mid \mathsf{ar}(P) = i\}$ and $\texttt{Funs}_i = \{f \in \texttt{Funs} \mid \mathsf{ar}(f) = i\}$ are countably infinite. The set $\texttt{Terms}$ of terms includes $\texttt{Cons}$ and $\texttt{Vars}$; and contains $f(t_1, \ldots, t_i)$ for every $i \geq 1$, every $f \in \texttt{Funs}_i$, and every $t_1, \ldots, t_i \in \texttt{Terms}$. A term $t \notin \texttt{Vars} \cup \texttt{Cons}$ is *functional*. An ASP *program* P is a set of *(non-ground) rules* of the form $H \leftarrow p_1(T_1), \ldots, p_m(T_m), \neg p_{m+1}(T_{m+1}), \ldots, \neg p_n(T_n)$ where $H = q_1(S_1), \ldots, q_\ell(S_\ell)$ with $0 \leq \ell \leq 1$, $|S_i| = \mathsf{ar}(q_i)$ for every $1 \leq i \leq \ell$, and $|T_i| = \mathsf{ar}(p_i)$ for every $1 \leq i \leq n$. In the above, $q_1, \ldots, q_\ell, p_1, \ldots, p_n \in \texttt{Preds}$ are predicates and $S_1, \ldots, S_\ell, T_1, \ldots, T_n$ are vectors over terms, i.e. variables, constants or functional terms (featuring variables or constants). We assume that rules are *safe*; that is, every variable in a rule occurs in some T_i with $1 \leq i \leq m$. For such a rule r, we define $H_r = H$, B_r^+ as the set of all positive atoms, and B_r^- as the set of all negative atoms in the antecedent. We call a program *ground* if it does not feature variables. The ground program $\mathsf{Ground}(P)$ results from a (non-ground) program P by creating all possible instantiations of rules where variables are replaced by terms from the herbrand universe of P. An *interpretation* I, i.e. a set of atoms, is a model for a ground program P if it satisfies all rules in P. That is, $(H_r \cup B_r^-) \cap I \neq \emptyset$ or $B_r^+ \setminus I \neq \emptyset$. Furthermore, I is an *answer set* of P, if additionally, every atom a in I is *proven*, meaning that $\{a\} = H_r$ for some rule $r \in P$ such that I contains B_r^+ but no atom in B_r^- and there is an ordering of atoms over I such that the order of atoms in B_r^+ is strictly smaller than the order of a.

3. Characterization of Programs with Infinite Groundings

There are essentially two high level reasons why a given ASP program does not admit a finite ground program that would allow to solve it reliably. They can have (1) at least one infinite answer set or (2) infinitely many finite answer sets. The intuition here is that a valid ground program needs to overestimate all possible answer sets of a program. Formally, a ground program P_g is a *valid grounding* for a program P if P_g and P have the same answer sets. To be able to obtain valid groundings that are finite, we limit ourselves to programs, which are *frugal* and *non-proliferous*.

Definition 1. *A program is* frugal *if it only admits finite answer sets; it is* non-proliferous *if it only admits finitely many finite answer sets (but arbitrarily many infinite ones).*

Both conditions are independent of each other. For example, a program that has no finite answer sets but at least one infinite one is non-proliferous but not frugal. The existence of program that is frugal and proliferous is less obvious.

Example 2. *The following ASP program admits infinitely many finite answer sets but no infinite one.*

$$next(Y, f(Y)) \leftarrow next(X, Y), \neg last(Y).$$
$$last(Y) \leftarrow next(X, Y), \neg next(Y, f(Y)).$$
$$done \leftarrow last(Y). \quad \leftarrow \neg done. \quad next(c, d).$$

Clearly, $\{next(c, d), last(d), done\}$ is an answer set. Also, any finite chain of $next$ relations terminated by $last$ is an answer set. However, an infinite $next$-chain is not an answer set as it cannot contain any $last$ atom, hence does not feature $done$, and therefore violates the constraint.

Unfortunately, checking if a program is frugal or non-proliferous is highly undecidable. Both checks are complete for respective levels in the arithmetical or even analytical hierarchy (see [21] for an introduction). The specific definitions of the classes are not vital as we show reductions from and to problems already known to be complete. Checking if a program is non-proliferous is comparably easy.

Theorem 1 ([22, Theorem 2])**.** *Deciding if a program is non-proliferous is Σ_2^0-complete.*

Membership is achieved as follows. Note that we can semi-decide whether a given ASP program has at least n answer sets for a given n. We can now semi-decide if the program is non-proliferous by using an oracle for the previous problem: we enumerate all natural numbers i and check with the oracle if the program has at least i answer sets; if the oracle rejects, we accept, otherwise we continue.

Hardness follows by a reduction from the complement of the universal halting problem of Turing machines (TM), i.e. the question whether a given TM halts on all inputs, which is known to be Π_2^0-complete. The main idea of the proof is to generate all arbitrarily long but finite inputs to a TM with a construction similar to Example 2. We then mount a standard TM simulation on top. Another important ingredient is the realization that we can reduce universal halting to checking if a TM halts on infinitely many inputs. For details, see [22].

Checking if a program is frugal is much harder, namely Π_1^1-complete. Luckily, we can reuse reductions that we can also use to (re-)prove that consistency of ASP programs is Σ_1^1-complete (for other existing proofs see Corollary 5.12 in [5] and Theorem 5.9 in [6]).

Theorem 2 ([22, Theorem 1])**.** *Deciding program consistency is Σ_1^1-complete.*

In this extended abstract we only briefly present the reductions in both directions. For correctness arguments, we refer the interested reader to our technical report [22]. We obtain membership by reducing to the following problem.

Proposition 1 ([23, Corollary 6.2][1])**.** *Checking if some run of a non-deterministic Turing machine on the empty word visits the start state infinitely many times is in Σ_1^1.*

For a program P and an interpretation I, let $\mathsf{Active}_I(P)$ be the set of all rules in $\mathsf{Ground}(P)$ that are not satisfied by I. If I is finite, then so is $\mathsf{Active}_I(P)$ and Active_I is computable.

Definition 2. *For a program P, let M_P be the non-deterministic TM that, regardless of the input, executes the following instructions:*

1. *Initialize an empty set L_0 of literals, and some counters $i := 0$ and $j := 0$.*
2. *If L_i^+ and L_i^- are not disjoint,* halt.
3. *If L_i^+ is an answer set of P,* loop *on the start state.*

[1] The original result shows Π_1^1-completeness for the complement.

4. *Initialize* $L_{i+1} := L_i \cup H_r \cup \{\neg a \mid a \in B_r^-\}$ *where* r *is some non-deterministically chosen rule in* $\mathit{Active}_{L_i^+}(P)$.

5. *If* L_i *satisfies all of the rules in* $\mathit{Active}_{L_j^+}(P)$*, then increment* $j := j + 1$ *and visit the start state once.*

6. *Increment* $i := i + 1$ *and go to Step 2.*

Now the program P is consistent if and only if M_P admits a run on the empty word that visits its start state infinitely many times. For hardness, we reduce to the following.

Definition 3. *A* tiling system *is a tuple* $\langle T, HI, VI, t_0\rangle$ *where* T *is a finite set of tiles,* HI *and* VI *are subsets of* $T \times T$*, and* t_0 *is a tile in* T*. Such a tiling system admits a* recurring solution *if there is a function* $f : \mathbb{N} \times \mathbb{N} \to T$ *such that:*

1. *For every* $i, j \geq 0$*, we have that* $\langle f(i,j), f(i+1,j)\rangle \notin HI$ *and* $\langle f(i,j), f(i,j+1)\rangle \notin VI$.

2. *There is an infinite subset* S *of* $\mathbb{N}$ *such that* $f(0,j) = t_0$ *for every* $j \in S$.

Proposition 2 ([23, Theorem 6.4][2])**.** *Checking if a tiling system admits a recurring solution is* Σ_1^1*-hard.*

We can encode this problem with the following program.

Definition 4. *For a* tiling system $\mathfrak{T} = \langle T, HI, VI, t_0\rangle$*, let* $P_{\mathfrak{T}}$ *be the program that contains the ground atom* $\mathit{Dom}(c_0)$ *and all of the following rules:*

$$\mathit{Dom}(s(X)) \leftarrow \mathit{Dom}(X)$$
$$\mathit{Tile}_t(X,Y) \leftarrow \mathit{Dom}(X), \mathit{Dom}(Y), \{\neg \mathit{Tile}_{t'}(X,Y) \mid t' \in T \setminus \{t\}\} \ \forall t \in T$$
$$\leftarrow \mathit{Tile}_t(X,Y), \mathit{Tile}_{t'}(s(X),Y) \ \forall \langle t, t'\rangle \in HI$$
$$\leftarrow \mathit{Tile}_t(X,Y), \mathit{Tile}_{t'}(X,s(Y)) \ \forall \langle t, t'\rangle \in VI$$
$$\mathit{Below}_{t_0}(Y) \leftarrow \mathit{Tile}_{t_0}(c_0, s(Y))$$
$$\mathit{Below}_{t_0}(Y) \leftarrow \mathit{Below}_{t_0}(s(Y))$$
$$\leftarrow \mathit{Dom}(Y), \neg \mathit{Below}_{t_0}(Y)$$

We obtain that $\mathfrak{T}$ has a recurring solution if and only if $P_{\mathfrak{T}}$ is consistent. This concludes that consistency for ASP programs is Σ_1^1-complete. To show that frugality is Π_1^1-complete, we essentially use the same reductions but for the complement of frugality.

Theorem 3 ([22, Theorem 3])**.** *Deciding if a program is frugal is* Π_1^1*-complete.*

For membership, we need to slightly change the machine M_P to halt in step 3 instead of looping on the start state. Then, the modified machine admits a run on the empty word that visits the start state infinitely many times if and only if the program has an infinite answer set, i.e. the program is not frugal. The hardness reduction works as is since $P_{\mathfrak{T}}$ either has an infinite answer set or none at all. Therefore it is not frugal if and only if it admits an answer set if and only if the tiling-system has a recurring solution.

Even if a program is both frugal and non-proliferous, consistency is still undecidable.

Theorem 4 ([22, Theorem 5])**.** *Consistency for frugal and non-proliferous programs is* Σ_1^0*-hard.*

Still, whenever a program is frugal, we can semi-decide consistency by enumerating all answer set candidates.

Theorem 5 ([22, Theorem 4])**.** *Consistency for frugal programs is in* Σ_1^0.

[2]The original result shows Σ_1^1-completeness.

4. Finite Groundings for Frugal and Non-Proliferous Programs

Based on the results from the previous section, we cannot hope to obtain a procedure that produces finite valid groundings for all frugal and non-proliferous programs. If this was the case, we could decide consistency for such programs. Instead, we sketch a procedure that requires to check if atoms are *forbidden*. While this property is undecidable itself, we give a checkable sufficient condition.

An atom is *forbidden* in the context of a program P, if it does not occur in any answer set of P. Undecidability follows by reducing from the halting problem of TMs, which is also done in Theorem 4. For a sufficient condition, the idea is to check all possible ways in which a given atom could be proven. If each of these ways is unsuccessful, e.g. by requiring contradictory literals, then the atom must be forbidden (see [22]). With the notion of forbidden atoms in place, we define the following grounding procedure.

Definition 5. *Define* $\mathit{GroundNotForbidden}(\cdot)$ *that takes a program* P *as input and executes the following instructions:*

1. *Initialize* $i := 1$*,* $A_0 := \emptyset$*, and* $P_g := \emptyset$.

2. *Initialize* $A_i := A_{i-1}$ *and for each* $r \in \mathit{Ground}(P)$ *with* $B_r^+ \subseteq A_{i-1}$*, do the following. If all atoms in* H_r *are forbidden in* P*, add* $\leftarrow B_r$ *to* P_g*. Otherwise, add* r *to* P_g *and add the single atom in* H_r *to* A_i.

3. Stop *if* $A_i = A_{i-1}$*; else set* $i := i + 1$ *and go to 2.*

The output of the procedure is P_g.

Correctness of the procedure is not hard to verify.

Theorem 6 ([22, Theorem 7])**.** *For a program* P*,* $\mathit{GroundNotForbidden}(P)$ *is a valid grounding, i.e.* P *and* $\mathit{GroundNotForbidden}(P)$ *have the same answer sets.*

Indeed, we also obtain that the procedure always terminates for frugal and non-proliferous programs.

Proposition 3 ([22, Proposition 5])**.** *For a frugal and non-proliferous program* P*,* $\mathit{GroundNotForbidden}(P)$ *is finite.*

However, we need to stress again that the procedure is not computable. To obtain a computable version, we need to exchange the check for forbidden atoms with a computable sufficient condition. While this retains correctness, it might not retain termination of the procedure.

5. Outlook

We hope that our results inspire further theoretical and practical research on support for function symbols in ASP. In particular, we hope that common workarounds like artificially limiting the size of the grounding with a "step" predicate as in Example 1 will become obsolete in the future. Obvious future work revolves around implementing our proposed grounding approach. An efficient implementation including a viable sufficient check for forbidden atoms requires in-depth considerations. Boosting generality of such a sufficient condition or taking disjunctions into account are interesting directions for theoretical research. In practice, a tradeoff between generality and performance needs to be found. Our work can be a reference for a first prototypical implementation that enables further experiments. Ideally, sufficient checks for forbidden atoms could then directly be integrated into existing grounders and ASP systems.

Acknowledgments

We want to acknowledge that the full modeling of the wolf, goat cabbage puzzle from the introduction is inspired by lecture slides created by Jean-François Baget.

On TU Dresden side, this work was partly supported by Deutsche Forschungsgemeinschaft (DFG) in project 389792660 (TRR 248, CPEC); by the Bundesministerium für Bildung und Forschung (BMBF) in the ScaDS.AI; by BMBF and DAAD (German Academic Exchange Service) in project 57616814 (SECAI); and by the cfaed.

Carral was financially supported by the ANR project CQFD (ANR-18-CE23-0003).

Hecher is funded by the Austrian Science Fund (FWF), grants J 4656 and P 32830, the Society for Research Funding in Lower Austria (GFF, Gesellschaft für Forschungsförderung NÖ) grant ExzF-0004, as well as the Vienna Science and Technology Fund (WWTF) grant ICT19-065. Parts of the research were carried out while visiting the Simons institute for the theory of computing at UC Berkeley.

References

[1] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103. URL: https://doi.org/10.1145/2043174.2043195. doi:10.1145/2043174.2043195.

[2] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer Set Solving in Practice, Morgan & Claypool, 2012. doi:10.2200/S00457ED1V01Y201211AIM019.

[3] M. Gebser, B. Kaufmann, T. Schaub, Solution enumeration for projected boolean search problems, in: CPAIOR'09, volume 5547, 2009, pp. 71–86.

[4] M. Alviano, C. Dodaro, S. Fiorentino, A. Previti, F. Ricca, Enumeration of Minimal Models and MUSes in WASP, in: LPNMR, volume 13416 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 29–42.

[5] V. W. Marek, A. Nerode, J. B. Remmel, The Stable Models of a Predicate Logic Program, The Journal of Logic Programming 21 (1994) 129–154. URL: https://www.sciencedirect.com/science/article/pii/S0743106614800083. doi:10.1016/S0743-1066(14)80008-3.

[6] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (2001) 374–425. URL: https://dl.acm.org/doi/10.1145/502807.502810. doi:10.1145/502807.502810.

[7] V. Marek, J. B. Remmel, Effectively Reasoning about Infinite Sets in Answer Set Programming, in: M. Balduccini, T. C. Son (Eds.), Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2011, pp. 131–147. URL: https://doi.org/10.1007/978-3-642-20832-4_9. doi:10.1007/978-3-642-20832-4_9.

[8] M. Alviano, F. Calimeri, W. Faber, G. Ianni, N. Leone, Function Symbols in ASP: Overview and Perspectives (2012).

[9] F. Calimeri, S. Cozza, G. Ianni, N. Leone, Computable functions in ASP: theory and implementation, in: M. G. de la Banda, E. Pontelli (Eds.), Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings, volume 5366 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 407–424. URL: https://doi.org/10.1007/978-3-540-89982-2_37. doi:10.1007/978-3-540-89982-2_37.

[10] A. Weinzierl, R. Taupe, G. Friedrich, Advancing Lazy-Grounding ASP Solving Techniques - Restarts, Phase Saving, Heuristics, and More, Theory Pract. Log. Program. 20 (2020) 609–624.

[11] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, Theory Pract. Log. Program. 19 (2019) 27–82.

[12] M. Alviano, W. Faber, N. Leone, S. Perri, G. Pfeifer, G. Terracina, The Disjunctive Datalog System DLV, in: O. de Moor, G. Gottlob, T. Furche, A. J. Sellers (Eds.), Datalog Reloaded - First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers, volume 6702 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 282–301. URL: https://doi.org/10.1007/978-3-642-24206-9_17. doi:10.1007/978-3-642-24206-9_17.

[13] F. Calimeri, D. Fuscà, S. Perri, J. Zangari, I-DLV: the new intelligent grounder of DLV, Intelligenza Artificiale 11 (2017) 5–20. URL: https://doi.org/10.3233/IA-170104. doi:10.3233/IA-170104.

[14] R. Kaminski, T. Schaub, On the foundations of grounding in answer set programming, CoRR abs/2108.04769 (2021).

[15] N. Hippen, Y. Lierler, Estimating grounding sizes of logic programs under answer set semantics, in: JELIA, volume 12678, Springer, 2021, pp. 346–361.

[16] M. Banbara, B. Kaufmann, M. Ostrowski, T. Schaub, Clingcon: The next generation, Theory Pract. Log. Program. 17 (2017) 408–461.

[17] T. Janhunen, R. Kaminski, M. Ostrowski, S. Schellhorn, P. Wanko, T. Schaub, Clingo goes linear constraints over reals and integers, Theory Pract. Log. Program. 17 (2017) 872–888.

[18] P. Cabalar, J. Fandinno, T. Schaub, P. Wanko, A uniform treatment of aggregates and constraints in hybrid ASP, in: KR, 2020, pp. 193–202.

[19] M. Bichler, M. Morak, S. Woltran, lpopt: A Rule Optimization Tool for Answer Set Programming, Fundamenta Informaticae 177 (2020) 275–296.

[20] L. Gerlach, D. Carral, M. Hecher, Finite groundings for asp with functions: A journey through consistency, in: K. Larson (Ed.), Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24, International Joint Conferences on Artificial Intelligence Organization, 2024, pp. 3386–3394. URL: https://doi.org/10.24963/ijcai.2024/375. doi:10.24963/ijcai.2024/375, main Track.

[21] H. Rogers, Jr., Theory of recursive functions and effective computability (Reprint from 1967), MIT Press, 1987. URL: http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=3182.

[22] L. Gerlach, D. Carral, M. Hecher, Finite groundings for ASP with functions: A journey through consistency, CoRR abs/2405.15794 (2024). URL: https://doi.org/10.48550/arXiv.2405.15794. doi:10.48550/ARXIV.2405.15794. arXiv:2405.15794.

[23] D. Harel, Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness, J. ACM 33 (1986) 224–248. URL: https://doi.org/10.1145/4904.4993. doi:10.1145/4904.4993.