Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Simple Guess-and-Check Programs: Strong and Uniform Equivalence Meet Again

Wolfgang Dvořák , Zeynep G. Saribatur , Stefan Woltran
Institute of Logic and Computation, TU Wien
{wolfgang.dvorak, zeynep.saribatur, stefan.woltran}@tuwien.ac.at

Abstract

We consider a particular subclass of normal programs that we call simple guess-and-check (SGC) programs. SGC programs consist of guess rules (i.e. rules without positive body atoms) and arbitrary constraints. Many simple combinatorial problems such as graph coloring can be encoded via SGC programs. Moreover, constraint-free SGC programs are known to have a close relation to abstract argumentation frameworks. Our main result shows that for SGC programs the notions of strong and uniform equivalence coincide (in contrast to general normal programs), but do not amount to classical equivalence (as is the case of positive programs). Moreover, we study the characteristics of SE-models for SGC programs; this allows to check whether an arbitrary program (part) can be equivalently formulated within the simpler class of SGC programs. Finally, we briefly discuss our results in relation to other classes of programs.

1 Introduction

In the realm of Answer-Set Programming (Brewka, Eiter, and Truszczynski 2011), the study of particular subclasses of programs has provided a fine-grained picture in terms of complexity (see, e.g. (Truszczynski 2011) or in the context of equivalence checking and program simplification (see, e.g. (Eiter et al. 2013). Prominent program classes are disjunctive, normal, or Horn programs.

We are interested here in a particular subclass of normal programs which we call simple guess-and-check (SGC) programs. A *simple guess-and-check (SGC)* program consists of rules without positive body atoms and of constraints. Despite the noticeably limited syntax, many fundamental combinatorial problems can be naturally expressed as SGC programs. We illustrate this with an example.

Example 1. *Below shows a representation of the 2-coloring problem for a graph.*

$$
\begin{aligned}
color(X, red) &\leftarrow not\ color(X, blue), node(X)\\
color(X, blue) &\leftarrow not\ color(X, red), node(X)\\
\bot &\leftarrow color(X, C_1), color(X, C_2), C_1 < C_2\\
\bot &\leftarrow edge(X, Y), X < Y, color(Y, C), color(X, C)
\end{aligned}
$$

Grounding the program with the facts $node(1)$, $node(2)$, *and* $edge(1, 2)$, *gives the following SGC program.*

$$
\begin{aligned}
color(1, red) &\leftarrow not\ color(1, blue)\\
color(2, red) &\leftarrow not\ color(2, blue)\\
color(1, blue) &\leftarrow not\ color(1, red)\\
color(2, blue) &\leftarrow not\ color(2, red)\\
\bot &\leftarrow color(1, red), color(1, blue)\\
\bot &\leftarrow color(2, red), color(2, blue)\\
\bot &\leftarrow color(1, red), color(2, red)\\
\bot &\leftarrow color(1, blue), color(2, blue)
\end{aligned}
$$

Moreover, SGC programs comprise the class of atomic logic programs (Janhunen 2006) which received some attention due to their close relations to different kinds of abstract argumentation frameworks (Caminada et al. 2015; Sá, Dvořák, and Caminada 2024; Buraglio, Dvořák, and Woltran 2025). In particular, there is a one-to-one correspondence between atomic logic programs and Dung-style abstract argumentation frameworks via simple and purely syntactic transformations (Caminada et al. 2015; Sá, Dvořák, and Caminada 2024). This correspondence does not only hold for stable semantics but connects the major abstract argumentation semantics with their counterparts for logic programming. Prominently, strong equivalence notions have been investigated in abstract argumentation (Baumann 2012; Baumann and Brewka 2018) and recently some differences between strong equivalence in several abstract argumentation formalisms and strong equivalence for logic programs have been detected (Buraglio, Dvořák, and Woltran 2025). Insights for strong equivalence in the corresponding classes of logic programs, e.g., atomic logic programs, can thus add to a better understanding of the different natures of these equivalence notions.

In this work, we investigate different equivalence notions for SGC and atomic programs. That is, (standard) equivalence where two programs are equivalent if they provide the same answer sets; uniform equivalence (Maher 1986; Sagiv 1986; Eiter and Fink 2003) where two programs are equivalent if they provide the same answer sets when extended by the same set of facts; and strong equivalence (Lifschitz, Pearce, and Valverde 2001) where two programs are equivalent if they provide the same answer sets when extended by the same programs. It is well known, that in general (and also for normal programs), these notions are differ-

ent relations. For some classes, like positive (i.e., negation-free) programs, strong and uniform equivalence coincide, however (Eiter, Fink, and Woltran 2007).

As main result of this paper, we show that for SGC programs (and thus also for atomic programs) the notions of strong and uniform equivalence coincide, contrasting the case of general normal programs. On the other hand, we will see that SGC programs behave differently compared to positive programs when another notion (classical equivalence) is considered. We also provide necessary and sufficient conditions the SE-models of SGC programs fulfill, thus providing means to simplify program (parts) to SGC programs.

2 Preliminaries

In the work we will consider (subclasses of) normal logic programs. We will distinguish rules and constraints as follows. Rules are of the form

$$h \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n.$$

and constraints are of the form

$$\bot \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n.$$

here h and b_i $(1 \leq i \leq n, 0 \leq m \leq n)$ are atoms from a first-order language, and not is default negation. We sometimes also write rules r as $H(r) \leftarrow B(r)$ or $H(r) \leftarrow B^+(r), not\ B^-(r)$, and constraints as $\bot \leftarrow B(r)$ or $\bot \leftarrow B^+(r), not\ B^-(r)$. We call $H(r) = h$ the head of r, $B(r) = \{b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n\}$ the body of r $B^+(r) = \{b_1, \ldots, b_m\}$ the positive body of r and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$ the negative body of r. A rule r is a fact if $B(r) = \emptyset$. A rule or constraint is positive if $B^-(r) = \emptyset$, atomic if $B^+(r) = \emptyset$, and unary if $|B^+(r)| \leq 1$ and $B^-(r) = \emptyset$.

A normal logic program (NLP) is a finite set of rules and constraints. In the rest of the paper, we focus on propositional programs over a set of atoms from the universe $\mathcal{U}$. A program is positive/atomic/unary if all its rule are of this restriction.

Answer sets of a program P, in symbols $AS(P)$, are defined via the GL-reduct as usual. Let $I \subseteq \mathcal{U}$ be an interpretation. The *GL-reduct* of a program P w.r.t. I is given by $P^I = \{H(r) \leftarrow B^+(r) \mid r \in P, B^-(r) \cap I = \emptyset\}$. An interpretation I is a *model* of a program P (in symbols $I \models P$) if, for each $r \in P$, $(H(r) \cup B^-(r)) \cap I \neq \emptyset$ or $B^+(r) \not\subseteq I$; I is an *answer set*, if it is a minimal model of P^I.

Two programs P_1, P_2 are *(standard) equivalent* if $AS(P_1) = AS(P_2)$, *strongly equivalent (SE)*, denoted by $P_1 \equiv P_2$, if $AS(P_1 \cup R) = AS(P_2 \cup R)$ for every R over $\mathcal{U}$, and *uniformly equivalent (UE)*, denoted by $P_1 \equiv_u P_2$ if $AS(P_1 \cup R) = AS(P_2 \cup R)$ for any set of facts R over $\mathcal{U}$. Common characterisations of uniform and strong equivalence are based on UE- and SE-models which we recall next (Eiter and Fink 2003; Turner 2001).

Definition 1. *An* SE-interpretation *is a pair (X, Y) such that $X \subseteq Y \subseteq \mathcal{U}$; it is total if $X = Y$ and non-total otherwise. An SE-interpretation (X, Y) is an* SE-model *of a program P if $Y \models P$ and $X \models P^Y$. An SE-model (X, Y) of P is called* UE-model *of P if $X = Y$ or there is no SE-model (X', Y) with $X \subset X' \subset Y$.*

We have that (normal) logic programs are strongly equivalent iff they have the same SE-models (Turner 2001) and uniformly equivalent iff they have the same UE-models (Eiter and Fink 2003).

Normal logic programs can be precisely characterized in terms of SE-models. Here we show the characterization as put together in (Gonçalves et al. 2020) to ease reading.

Proposition 2 ((Eiter et al. 2004))**.** *Let M be a set of SE-interpretations. Then, there exists a normal program P such that $M = SE(P)$ iff M satisfies the following conditions:*

(N_1) $(X, Y) \in M$ implies $(Y, Y) \in M$;

(N_2) $(X, Y) \in M$, and $(Y', Y') \in M$ with $Y \subseteq Y'$ implies $(X, Y') \in M$;

(N_3) $(X, Y) \in M$ and $(X', Y) \in M$ implies $(X \cap X', Y) \in M$.

3 SGC-Programs and Strong Equivalence

We consider two subclasses of normal logic programs, namely atomic (Janhunen 2006) and simple guess-and-check (SGC) programs, which we define next.

Definition 3. *A normal logic program is called* atomic *if all its rules and constraints are atomic and it is called a* simple guess-and-check *(SGC) program if all its rules are atomic (but constraints are not restricted).*

By definition we have that every atomic program is also a SGC program but not vice versa.

The notions of uniform and standard equivalence carry over to SGC programs in the expected way. However, for the notions of strong equivalence, some care is needed. In fact, when considering strong equivalence between programs from a particular subclass, a fundamental question is whether the context programs R in the test for $AS(P \cup R) = AS(Q \cup R)$ should stem from the same subclass as P and Q or not. One reason why this question has not received much attention yet is due to an implicit, but important, result in the seminal paper on strong equivalence (Lifschitz, Pearce, and Valverde 2001). There it is shown that testing strong equivalence between two programs coincides with testing $AS(P \cup R) = AS(Q \cup R)$ for each *unary* program R, i.e. programs which are either facts or of the simple form $a \leftarrow b$. Many of the studied subclasses are indeed superclasses of unary programs and thus results on strong equivalence between programs of such a subclass do not need to distinguish between arbitrary programs or programs from the subclasses for the context program.

It is crucial to observe that SGC programs are *not* a superclass of unary programs, thus the definition of strong equivalence needs to be made precise. In principle, two versions are feasible.

Definition 4. *Let P, Q be two SGC, resp. atomic, programs:*

- *P, Q are* strongly equivalent *(SE), denoted by $P \equiv_s Q$, if $AS(P \cup R) = AS(Q \cup R)$ for every program R over $\mathcal{U}$;*
- *P, Q are* strongly class-tailored equivalent *(SCE), denoted by $P \equiv_{sc} Q$, if $AS(P \cup R) = AS(Q \cup R)$ for every SGC, atomic resp., program R over $\mathcal{U}$.*

As we will see in our main result, the same observation as for the other classes nonetheless applies to SGC programs. In other words, the notions of SE and SCE coincide, but as outlined above we cannot make implicit use of the result involving unary programs.

4 Main Result

In this section, we show that for SGC programs the equivalence notions of SE, SCE, and UE coincide and can be decided via SE-models (or analogously via UE-models).

Let us start with some observations on certain properties regarding the SE-models of an SGC program.

Proposition 5. *Given SGC program P, the following hold*

a) $(X,Y) \in SE(P) \Rightarrow (X',Y) \in SE(P)$ *for each* X' *with* $X \subseteq X' \subseteq Y$

b) $X \subset Y, (X,Y) \notin SE(P) \Rightarrow$ *there exists* $x \in Y \setminus X$ *with* $(Y \setminus \{x\}, Y) \notin SE(P)$

Proof. Point (a) is by the fact that for SGC programs the reduct P^Y only contains facts and constraints that are not violated by Y. As $X \models P^Y$ and $X \subseteq X'$ we have that X' also contains the facts of P^Y. Moreover, as $X' \subseteq Y$ it does not violate the constraints in P^Y. That is, $X' \models P^Y$ and thus $(X',Y) \in SE(P)$.

(b) If $Y \not\models P$ or $X = Y \setminus \{x\}$ for some $x \in Y \setminus X$ the statement is trivially true. Thus let us assume otherwise. As $Y \models P$ neither Y nor $X \subset Y$ violate a constraint in P^Y. Now, as $(X,Y) \notin SE(P)$ there must be a fact $x \in P^Y$ that is not in X. That is, $Y \setminus \{x\} \not\models P^Y$ and thus $(Y \setminus \{x\}, Y) \notin SE(P)$. □

We can now give our main result. Note that Condition 1 refers to SE between SGC programs, Condition 2 to SCE between SGC programs and Condition 4 to UE between SGC programs.

Theorem 6. *For SGC programs P and Q, the following are equivalent*

1. $AS(P \cup R) = AS(Q \cup R)$ *for any program* R
2. $AS(P \cup R) = AS(Q \cup R)$ *for any SGC program* R
3. $AS(P \cup R) = AS(Q \cup R)$ *for any atomic program* R
4. $AS(P \cup R) = AS(Q \cup R)$ *for any set* R *of facts*
5. $UE(P) = UE(Q)$
6. $SE(P) = SE(Q)$

Proof. (1) ⇒ (2) ⇒ (3) ⇒ (4) is by the fact that, in each step, we strictly refine the class of programs R we consider. (4) ⇒ (5) is by the fact that logic programs are uniformly equivalent iff their UE-models coincide (Eiter and Fink 2003).
(5) ⇒ (6): Assume $SE(P) \neq SE(Q)$ and that, w.l.o.g., there exists $(X,Y) \in SE(P) \setminus SE(Q)$. If $X = Y$, then we have $AS(P) \neq AS(Q)$, which contradicts (4). Otherwise, by Proposition 5(b), $(Y \setminus \{x\}, Y) \notin SE(Q)$ for some $x \in Y \setminus X$. Also, by Proposition 5(a), $(Y \setminus \{x\}, Y) \in SE(P)$. Since there cannot be another $(M,Y) \in SE(P)$ with $Y \setminus \{x\} \subset M \subset Y$, $(Y \setminus \{x\}, Y) \in UE(P)$. This however then contradicts (4).
(6) ⇒ (1) is by the fact that logic programs are strongly equivalent iff their SE-models coincide (Turner 2001). □

From above result, in particular Condition 3, it follows immediately that also for atomic programs SE-, SCE-, and UE-equivalence coincide.

We recall that also for the class of positive programs, strong and uniform equivalence coincide (Eiter, Fink, and Woltran 2007). However, there is a notable distinction to SGC programs, when another equivalence notion is considered, namely classical equivalence (which states that the total SE-models of the compared programs coincide).

For positive programs, it is the case that SE, UE and classical equivalence coincide; formally, this is due to the fact that $(X,Y) \in SE(P)$ implies $(X,X) \in SE(P)$ for positive programs. However, characterizing SE (or UE) via classical equivalence does not work for SGC programs.

Example 2. *Consider the two atomic programs*

$$P : \bot \leftarrow not\ a \quad \bot \leftarrow not\ b \quad a \leftarrow$$
$$Q : \bot \leftarrow not\ a \quad \bot \leftarrow not\ b \quad b \leftarrow$$

P and Q have the same (classical) models, but are neither strongly nor uniformly equivalent. That is, for both programs the only (classical) model is $\{a,b\}$, but for R=$\{a \leftarrow\}$ we have $AS(P \cup R) = \emptyset$ while $AS(Q \cup R) = \{a,b\}$.

5 Characterization and Complexity Results

In this section, we refine the result from Proposition 2 to the class of SGC programs. Characterization results of this kind allow to "recast" program (parts) P to a simpler class, in the sense that when P satisfies the relevant conditions it can be rewritten to a syntactically simpler class, which in turn might speed-up solving a program via ASP systems.

Proposition 7. *Let M be a set of SE-interpretations. Then, there exists an SGC program P such that $M = SE(P)$ iff M satisfies*

(N_1') $(X,Y) \in M$ *implies* $(X',Y) \in M$ *for each* X' *with* $X \subseteq X' \subseteq Y$*;*

(N_2) $(X,Y) \in M$ *, and* $(Y',Y') \in M$ *with* $Y \subseteq Y'$ *implies* $(X,Y') \in M$ *;*

(N_3) $(X,Y) \in M$ *and* $(X',Y) \in M$ *implies* $(X \cap X', Y) \in M$.

Proof (Sketch). The only-if direction follows directly from Proposition 2 and Proposition 5(a).

For the if-direction, let M be a set of SE-models over atoms $\mathcal{A} \subset \mathcal{U}$ satisfying conditions N_1', N_2, and N_3. Let $M' = \{Y \mid (Y,Y) \in M\}$ be the set of classical models in M; we use a standard construction for providing a CNF $\bigwedge_{i=1}^{m}(\ell_{i_1} \vee \cdots \vee \ell_{i_{n_i}})$ with models M' and take program

$$P' = \{\bot \leftarrow \bar{\ell}_{i_1}, \ldots, \bar{\ell}_{i_{n_i}} \mid 1 \leq i \leq m\}$$

consisting of arbitrary constraints ($\bar{\ell}$ denotes the negation of literal ℓ). Note that the SE-models of P' are given by $SE(P') = \{(X,Y) \mid Y \in M', X \subseteq Y\}$; by condition N_1', we get $M \subseteq SE(P')$. In order to get rid of the additional SE-models of P', for each $Y \in M'$ take the minimal

X from all $(X, Y) \in M$ (a unique such X exists due to condition N_3), denoted it as X_Y and consider the program

$$P = P' \cup \bigcup_{Y \in M'} \{x \leftarrow not\ (\mathcal{A} \setminus X_Y) \mid x \in X_Y\}.$$

Clearly, P is still an SGC program. Notice that, by condition N_2, for $Y, Y' \in M'$ we have that $Y \subset Y'$ implies $X_Y \supseteq X_{Y'}$. That is, there is no interference between rules introduced for different $Y, Y' \in M'$: first if Y, Y' are not in subset relation then the rules for Y do not contribute facts to the reduct $P^{Y'}$ and vice versa; if $Y \subset Y'$ then the rules for Y do not add facts to the reduct $P^{Y'}$ and the rules of Y', due to $X_Y \supseteq X_{Y'}$, only add facts to the reduct P^Y that are also added by the rules for Y. Given that it can be shown that $SE(P) = M$. □

A corresponding result for atomic programs is left for future work (note that the construction sketched in the if-direction makes use of arbitrary constraints).

As a final contribution, we consider the computational complexity of the studied problems. First, we consider deciding different notions of equivalence when comparing SGC programs.

Proposition 8. *For SGC programs P and Q, the following problems are* coNP*-complete:*

1. *Deciding $P \equiv Q$,*
2. *Deciding $P \equiv_s Q$,*
3. *Deciding $P \equiv_{sc} Q$, and*
4. *Deciding $P \equiv_u Q$.*

Proof. As these problems are in coNP for normal logic programs (Eiter, Fink, and Woltran 2007) they are also in coNP for the subclasses of SGC.

For the hardness, by Theorem 6, it suffices to consider standard and uniform equivalence. In both cases we provide a reduction from the coNP-hard CNF UNSAT problem. First, consider standard equivalence. Given a proposition formula in 3-CNF form $\bigwedge_{i=1}^{m}(\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3})$ with ℓ_{i_j} being literals over atoms in X we construct an instance (P, Q) with two atomic programs P, Q as follows: We define a operator α that maps literals over X to atoms over $X \cup \bar{X}$. Let $x \in X$, we define $\alpha(x) = x$ and $\alpha(\neg x) = \bar{x}$.

$$\begin{aligned} P =& \{x \leftarrow not\ \bar{x} \mid x \in X\} \cup \{\bar{x} \leftarrow not\ x \mid x \in X\} \cup \\ & \{\bot \leftarrow not\ \alpha(\ell_{i_1}), not\ \alpha(\ell_{i_2}), not\ \alpha(\ell_{i_3}), \mid 1 \leq i\ \leq m\} \\ Q =& \{\bot \leftarrow\} \end{aligned}$$

Obviously, Q has no answer-sets. It is easy to verify that P has an answer-set iff the formula φ has a model. That is, $P \equiv Q$ iff φ is unsatisfiable.

Now, consider uniform equivalence. Given a proposition formula in 3-CNF form $\bigwedge_{i=1}^{m}(\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3})$ with ℓ_{i_j} being literals over atoms in X we construct an instance (P, Q) with two atomic programs P, Q as follows: We use the operator α from above.

$$\begin{aligned} P =& \{\bot \leftarrow x, \bar{x} \mid x \in X\} \cup \{\bot \leftarrow not\ x, not\ \bar{x} \mid x \in X\} \cup \\ & \{\bot \leftarrow not\ \alpha(\ell_{i_1}), not\ \alpha(\ell_{i_2}), not\ \alpha(\ell_{i_3}), \mid 1 \leq i\ \leq m\} \\ Q =& \{\bot \leftarrow\} \end{aligned}$$

As before, Q has no answer-sets. Moreover, one can easily verify that $P \cup R$ has an answer-set iff the facts in R represent a valid truth assignment on X and this truth assignment is a model of φ. That is, $P \equiv_u Q$ iff φ is unsatisfiable. □

Finally, we consider the complexity of deciding whether a normal logic program can be recasted into a SGC-program.

Proposition 9. *Given a normal logic program P. It is* coNP*-complete to decide whether there is a SGC-program Q such that $P \equiv_s Q$.*

Proof. The membership is by a simple guess and check algorithm. If the SE-models of P violate one of the conditions of Proposition 7, there is witness consisting of at most three SE-interpretations. One can guess these SE-interpretations and check whether they are SE-models in polynomial time.

Hardness is by a reduction from the UNSAT problem. Given a 3-CNF formula $\varphi = \bigwedge_{i=1}^{m}(\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3})$ with ℓ_{i_j} being literals over X we construct a program P as follows ($\bar{\ell}$ denotes the negation of literal ℓ and $s, t \notin X$).

$$\begin{aligned} P =& \{\bot \leftarrow \bar{\ell}_{i_1}, \ldots, \bar{\ell}_{i_3} \mid 1 \leq i \leq m\} \cup \\ & \{\bot \leftarrow not\ s\} \cup \{t \leftarrow s\} \end{aligned}$$

The classical models of P are $\{M \cup \{s, t\} \mid M$ is model of $\varphi\}$. If φ is unsatisfiable then P has no classical models, no SE-models, and thus $Q = \{\bot\ \leftarrow\}$ is a strongly equivalent SGC program. If φ has a model M then, for $Y = M \cup \{s, t\}$, we have that $(\emptyset, Y)$ is an SE-model but $(\{s\}, Y)$ is not. As this violates condition N_1' of Proposition 7, there is no strongly equivalent SGC program. □

6 Discussion

In this paper, we have investigated the class of simple guess-and-check (SGC) programs and showed that for this class the notions of strong and uniform equivalence coincide. Moreover, we provided accompanying characterization and complexity results.

By their guess and check nature SGC programs can express a large variety of (NP-complete) combinatorial problems, the limitation being that no reachability or transitive closure computation is needed. That encompasses a variety of graph problems. Beside the 2-coloring problem (that we used as an illustrative example on the structure of these programs) this includes prominent NP-complete problems like 3-coloring and dominating set. Other examples from well-known ASP benchmarks that can be expressed with SGC programs include sudoku, n-queens, and stable marriage problems. Notice that, while the encodings as non-ground programs do not strictly follow the syntactic restrictions of SGC programs (they use positive atoms in order to obtain safe rules) the groundings of these programs typically result in SGC programs.

It is known that for infinite domains, deciding uniform equivalence of non-ground programs is undecidable in general, while deciding strong equivalence is coNEXPTIME-complete (Eiter et al. 2005). Similar to positive programs, our results suggest that for non-ground programs in the spirit of the class of SGC programs, uniform equivalence becomes decidable; however, a closer inspection is subject of future

work since the requirement of safe rules needs particular attention here.

The class closest to our SGC programs are singular programs (Fichte, Truszczynski, and Woltran 2015), i.e., programs that are both normal and dual-normal, which consist of either constraints or rules with at most one positive body atom. It is interesting to observe that allowing for one positive atom in the body already causes strong and uniform equivalence to differ. Similar classes (including atomic programs) have been investigated by (Janhunen 2006) in terms of expressibility. We leave a detailed comparison to these classes (where constraints are restricted) for future work.

Lastly, there remains the open question on whether there is a more general property on the programs that guarantees that strong and uniform equivalence coincide. Given that we now have two classes of programs (positive and SGC programs) that satisfy this property, the question can be approached either on the syntactical or semantical (via set-theoretic relationships on the SE-models) level with the ultimate goal to have a result that captures both classes. However, this cannot be done naively since, as is known from the literature, comparing the positive program $\{a \vee b \leftarrow\}$ with the SGC program $\{a \leftarrow not\ b; b \leftarrow not\ a\}$ already yields an example of programs that are uniform but not strongly equivalent.

Acknowledgments

We thank the reviewers for their valuable feedback. This work has been supported by the Austrian Science Fund (FWF) under grants T-1315 and 10.55776/COE12, and by the Vienna Science and Technology Fund (WWTF) project ICT25-044.

AI Declaration

The authors have not employed any Generative AI tools.

References

Baumann, R., and Brewka, G. 2018. The equivalence zoo for dung-style semantics. *J. Log. Comput.* 28(3):477–498.

Baumann, R. 2012. Normal and strong expansion equivalence for argumentation frameworks. *Artif. Intell.* 193:18–44.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12):92–103.

Buraglio, G.; Dvořák, W.; and Woltran, S. 2025. On strong equivalence notions in logic programming and abstract argumentation. In Rapberger, A., and Rudolph, S., eds., *Proceedings of the 23st International Workshop on Non-Monotonic Reasoning*, 31–44.

Caminada, M.; Sá, S.; Alcântara, J. F. L.; and Dvořák, W. 2015. On the equivalence between logic programming semantics and argumentation semantics. *Int. J. Approx. Reason.* 58:87–111.

Eiter, T., and Fink, M. 2003. Uniform equivalence of logic programs under the stable model semantics. In Palamidessi, C., ed., *Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings*, volume 2916 of *Lecture Notes in Computer Science*, 224–238. Springer.

Eiter, T.; Fink, M.; Tompits, H.; and Woltran, S. 2004. On eliminating disjunctions in stable logic programming. *KR* 4:447–458.

Eiter, T.; Fink, M.; Tompits, H.; and Woltran, S. 2005. Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case. In *AAAI*, 695–700.

Eiter, T.; Fink, M.; Pührer, J.; Tompits, H.; and Woltran, S. 2013. Model-based recasting in answer-set programming. *J. Appl. Non Class. Logics* 23(1-2):75–104.

Eiter, T.; Fink, M.; and Woltran, S. 2007. Semantical characterizations and complexity of equivalences in answer set programming. *ACM Trans. Comput. Log.* 8(3):17.

Fichte, J. K.; Truszczynski, M.; and Woltran, S. 2015. Dual-normal logic programs - the forgotten class. *Theory Pract. Log. Program.* 15(4-5):495–510.

Gonçalves, R.; Knorr, M.; Leite, J.; and Woltran, S. 2020. On the limits of forgetting in answer set programming. *Artif. Intell.* 286:103307.

Janhunen, T. 2006. Some (in)translatability results for normal logic programs and propositional theories. *J. Appl. Non Class. Logics* 16(1-2):35–86.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Trans. Comput. Log.* 2(4):526–541.

Maher, M. J. 1986. Eqivalences of logic programs. In Shapiro, E., ed., *Third International Conference on Logic Programming, Imperial College of Science and Technology, London, United Kingdom, July 14-18, 1986, Proceedings*, volume 225 of *Lecture Notes in Computer Science*, 410–424. Springer.

Sá, S.; Dvořák, W.; and Caminada, M. 2024. Syntactic and semantic connections between logic programming and argumentation systems. In Gabbay, D.; Kern-Isberner, G.; Simari, G. R.; and Thimm, M., eds., *Handbook of Formal Argumentation*, volume 3. College Publications. chapter 7, 429–511.

Sagiv, Y. 1986. Optimizing datalog programs. In Korth, H. F., ed., *XP / 7.52 Workshop on Database Theory, University of Texas at Austin, TX, USA, August 13-15, 1986*.

Truszczynski, M. 2011. Trichotomy and dichotomy results on the complexity of reasoning with disjunctive logic programs. *Theory Pract. Log. Program.* 11(6):881–904.

Turner, H. 2001. Strong equivalence for logic programs and default theories (made easy). In Eiter, T.; Faber, W.; and Truszczynski, M., eds., *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings*, volume 2173 of *Lecture Notes in Computer Science*, 81–92. Springer.