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

Wolfgang Dvořák, Zeynep G. Saribatur, and Stefan Woltran

Institute of Logic and Computation, TU Wien

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). Finally, we briefly discuss our results in relation to other classes of programs.

Keywords: strong equivalence · uniform equivalence · guess-and-check program.

1 Introduction

We are interested 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 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 2-coloring problem for a graph with two vertices and edges are provided as facts.

$$\begin{array}{ll} color1red \leftarrow not\ color1blue. & color1blue \leftarrow not\ color1red. \\ color2red \leftarrow not\ color2blue. & color2blue \leftarrow not\ color2red. \\ \bot \leftarrow not\ color1red, not\ color1blue. & \bot \leftarrow not\ color2red, not\ color2blue. \\ \bot \leftarrow color1red, color1blue. & \bot \leftarrow color2red, color2blue. \\ \bot \leftarrow edge12, color1red, color2red. & \bot \leftarrow edge12, color1blue, color2blue. \end{array}$$

Moreover, SGC programs comprise the class of atomic logic programs [7] which received some attention due to their close relations to different kinds of abstract argumentation frameworks [1, 11, 6].

In this work we investigate different equivalence notions for SGC and atomic programs. That is, classic equivalence where two programs are equivalent if they

provide the same answer-sets; uniform equivalence [9, 10, 2] where two programs are equivalent if they provide the same answer-sets when extended by the same set of facts; and strong equivalence [8] where two programs are equivalent if they provide the same answer-sets when extended by the same SGC program.

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, and can be characterised via SE-models. Though, these notions still deviate from classical equivalence, which is in contrast to the class of positive programs for which these three equivalence notions coincide.

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

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

and constraints are of the form

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

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 also write r as $H(r) \leftarrow B(r)$ or $H(r) \leftarrow B^+(r), \mathit{not}\ B^-(r)$. We call $H(r) = h$ the head of r, $B(r) = \{b_1, \ldots, b_m, \mathit{not}\ b_{m+1}, \ldots, \mathit{not}\ b_n\}$ 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 (non-disjunctive) fact if $B(r) = \emptyset$. A rule or constraint is positive if $B^-(r) = \emptyset$, atomic if $B^+(r) = \emptyset$, and unary if $|B^+(r)| \leq 1$. Answer sets of a program P (in symbols $AS(P)$) are defined via the GL-reduct as usual.

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 $\mathcal{U}$. We consider two subclasses of normal logic programs, namely atomic [7] and simple guess-and-check (SGC) programs, which we define next.

Definition 1. *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. We next recall the different equivalence notions for logic programs, i.e., standard equivalence, strong equivalence, and uniform equivalence. Notice that we use a strong equivalence notion that is tailored to the specific classes of programs we study.

Definition 2. *Let P, Q be two SGC, resp atomic, programs:*

- *P, Q are* equivalent*, denoted by $P \equiv Q$, if $AS(P) = AS(Q)$.*

- P, Q *are* uniformly equivalent *(UE), denoted by* $P \equiv_u Q$ *if* $AS(P \cup R) = AS(P \cup R)$ *for any set of facts* R *over* $\mathcal{U}$.
- P, Q *are* strongly equivalent *(SE), denoted by* $P \equiv_s Q$, *if* $AS(P \cup R) = AS(P \cup R)$ *for every SGC, atomic resp., program* R *over* $\mathcal{U}$.

Common characterisations of uniform and strong equivalence are based on UE- and SE-models which we recall next [2, 12].

Definition 3. *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 [12] and uniformly equivalent iff they have the same UE-models [2].

3 Characterising Strong and Uniform Equivalence of SGC-programs

When characterising strong equivalence for subclasses of logic programs we can distinguish two variants. First, studying whether programs of the subclass are strongly equivalent in the larger class, i.e., with respect to arbitrary expansions. Second, studying whether programs of the subclass are strongly equivalent within the subclass, i.e., with respect to expansions within the subclass. For most classes the result on strong equivalence where context can equally be restricted to unary programs (see [8]) makes such a differentiation unnecessary.

Here we consider the strong equivalence notion tailored to the class of SGC-programs: given two SGC-program P, Q, does $AS(P \cup R) = AS(Q \cup R)$ hold for any further SGC-program R? Due to the restriction of the expansion R to SGC-programs, it is not clear whether the known characterisation for NLPs via SE-models applies here. In the following will re-establish this characterisation and show that on SGC-programs both variants of strong equivalence coincide with uniform-equivalence.

Let us start with some observations on certain properties regarding the SE-models $SE(P)$ of a SGC program P.

Proposition 1. *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 can 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 $(Y \setminus \{x\}, Y) \notin SE(P)$. □

Then our main result follows.

Theorem 1. *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) by the fact that logic programs are uniformly equivalent iff their UE-models coincide [2].
(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 1(b), $(Y \setminus \{x\}, Y) \notin SE(Q)$ for some $x \in Y \setminus X$. Also, by Proposition 1(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) by the fact that logic programs are strongly equivalent iff their SE-models coincide [12]. □

Hence, strong and uniform equivalence coincide for SGC-programs but it turns out that in contrast to positive LPs where all three notions of equivalence coincide [4], classical equivalence is too weak here.

Example 2. Consider the two programs

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

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 $AS(Q \cup R) = \{a, b\}$.

Finally, we consider the computational complexity of the different notions of equivalence.

Proposition 2. *For SGC programs P and Q, the following problems are* coNP*-complete:*

1. *Deciding $P \equiv Q$,*

2. *Deciding* $P \equiv_s Q$, *and*
3. *Deciding* $P \equiv_u Q$.

Proof. As these problems are in coNP for normal logic programs [4] they are also in coNP for the subclasses of SGC.

For the hardness it suffice to consider standard 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 $\alpha(\neg x) = \bar{x}$.

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

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.

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

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. By Theorem 1 this also shows the hardness of strong equivalence. □

4 Discussion

The class closest to our SGC programs are singular programs [5], 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 [7] in terms of expressibility. We leave a detailed comparison to these classes (where constraints are restricted) for future work.

It is known that for infinite domains, deciding uniform equivalence of non-ground programs is undecidable in general, while deciding strong equivalence is co-NEXPTIME-complete [3]. Similar to positive programs, our results suggest that for SGC programs uniform equivalence becomes decidable.

References

1. Caminada, M., Sá, S., Alcântara, J.F.L., Dvořák, W.: On the equivalence between logic programming semantics and argumentation semantics. Int. J. Approx. Reason. **58**, 87–111 (2015). https://doi.org/10.1016/J.IJAR.2014.12.004
2. Eiter, T., Fink, M.: 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. Lecture Notes in Computer Science, vol. 2916, pp. 224–238. Springer (2003). https://doi.org/10.1007/978-3-540-24599-5_16
3. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case. In: AAAI. pp. 695–700 (2005)
4. Eiter, T., Fink, M., Woltran, S.: Semantical characterizations and complexity of equivalences in answer set programming. ACM Trans. Comput. Log. **8**(3), 17 (2007). https://doi.org/10.1145/1243996.1244000
5. Fichte, J.K., Truszczynski, M., Woltran, S.: Dual-normal logic programs - the forgotten class. Theory Pract. Log. Program. **15**(4-5), 495–510 (2015). https://doi.org/10.1017/S1471068415000186
6. Giovanni Buraglio, Wolfgang Dvořák, S.W.: On strong equivalence notions in logic programming and abstract argumentation. In: Rapberger, A., Rudolph, S. (eds.) Proceedings of the 23st International Workshop on Non-Monotonic Reasoning. pp. 31–44 (2025)
7. Janhunen, T.: Some (in)translatability results for normal logic programs and propositional theories. J. Appl. Non Class. Logics **16**(1-2), 35–86 (2006). https://doi.org/10.3166/JANCL.16.35-86
8. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Log. **2**(4), 526–541 (2001). https://doi.org/10.1145/383779.383783
9. Maher, M.J.: 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. Lecture Notes in Computer Science, vol. 225, pp. 410–424. Springer (1986). https://doi.org/10.1007/3-540-16492-8_91
10. Sagiv, Y.: 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 (1986)
11. Samy Sá, Wolfgang Dvořák, M.C.: Syntactic and semantic connections between logic programming and argumentation systems. In: Gabbay, D., Kern-Isberner, G., Simari, G.R., Thimm, M. (eds.) Handbook of Formal Argumentation, vol. 3, chap. 7, pp. 429–511. College Publications (2024)
12. Turner, H.: Strong equivalence for logic programs and default theories (made easy). In: Eiter, T., Faber, W., Truszczynski, M. (eds.) Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2173, pp. 81–92. Springer (2001). https://doi.org/10.1007/3-540-45402-0_6