Applying Visible Strong Equivalence in Answer-Set Program Transformations

JORI BOMANSON, Aalto University, Finland
TOMI JANHUNEN, Aalto University, Finland and Tampere University, Finland
ILKKA NIEMELÄ, Aalto University, Finland

Strong equivalence is one of the basic notions of equivalence that have been proposed for logic programs subject to the answer-set semantics. In this article, we propose a new generalization of strong equivalence (SE) that takes the visibility of atoms into account and we characterize it in terms of appropriately revised SE-models. Our design resembles (relativized) strong equivalence but is substantially different due to adopting a strict one-to-one correspondence of models from the notion of visible equivalence. We additionally tailor the characterization for more convenient use with positive programs and provide formal tools to exploit the tailored version also in the case of some programs that use negation. We illustrate the use of visible strong equivalence and the characterizations in showing the correctness of program transformations that make use of atom visibility. Moreover, we present a translation that enables us to automate the task of verifying visible strong equivalence for particular fragments of answer-set programs. We experimentally study the efficiency of verification when the goal is to check whether an extended rule is visibly strongly equivalent to its normalization, i.e., a subprogram expressing the original rule in terms of normal rules only. In the process, we verify the outputs of several real implementations of normalization schemes on a considerable number of input rules.

CCS Concepts: • **Theory of computation → Constraint and logic programming**; *Logic and verification*; • **Computing methodologies → Logic programming and answer set programming**;

Additional Key Words and Phrases: Stable models, strong equivalence, hidden atoms, auxiliary atoms, program transformations, normalization

ACM Reference format:
Jori Bomanson, Tomi Janhunen, and Ilkka Niemelä. 2020. Applying Visible Strong Equivalence in Answer-Set Program Transformations. *ACM Trans. Comput. Logic* 21, 4, Article 33 (October 2020), 41 pages.
https://doi.org/10.1145/3412854

An earlier version of this article appeared with the same title in Essays on Logic-Based AI in Honour of Vladimir Lifschitz, volume 7265 of LNCS, pages 363–379. Springer, 2012. DOI: 10.1007/978-3-642-30743-0_24.
This work has been supported in part by the Finnish centre of excellence in *Computational Inference Research* (COIN) (Academy of Finland, project #251170). Moreover, Jori Bomanson has been supported by Helsinki Doctoral Network in Information and Communication Technology (HICT) and Tomi Janhunen partially by the Academy of Finland project *Ethical AI for the Governance of Society* (ETAIROS, grant no. 327352).
Authors' addresses: J. Bomanson and I. Niemelä, Aalto University, Computer Science, School of Science, P. O. Box 15400, FI-00076, Aalto, Finland; emails: jori.bomanson@aalto.fi, ilkka.niemela@aalto.fi; T. Janhunen, Aalto University, Computer Science, School of Science, P. O. Box 15400, FI-00076, Aalto, Finland, and Tampere University, Information Technology and Communication Sciences, Kanslerinrinne 1, 33100, Tampere, Finland; emails: tomi.janhunen@aalto.fi, tomi.janhunen@tuni.fi.

1529-3785/2020/10-ART33
https://doi.org/10.1145/3412854

1 INTRODUCTION

Answer-set programming (ASP) [8] is a declarative problem solving paradigm where the solutions of a problem are determined by computing *answer sets* [21, 22, 28], originally coined as *stable models* by Gelfond and Lifschitz [20], for a logic program representing the problem. Rapid improvement of answer-set *solvers* has lifted ASP into an efficient way to solve search problems arising in many applications. The development of answer-set programs and the implementation of related solving tools often involves program transformations that substitute parts of input programs with semantically equivalent, but syntactically different representations, obtained for example via normalization [4] or simplification. Typically the aim is to improve performance [7] or facilitate further processing [39]. The formal soundness of such transformations is ideally established by analyzing each substitution in isolation from the rest of the program. This is, however, highly nontrivial due to the global nature of answer-set semantics, which causes the answer sets of a program to be more than a function of the answer sets of its parts.

The importance of these practical soundness considerations has motivated a broad line of research founded on *strong equivalence* by Lifschitz et al. [29] (see, e.g., Reference [13]). The strong equivalence relation holds for pairs of programs that are completely interchangeable not only in isolation but also in the context of any larger program. It has been characterized in terms of superintuitionistic logic [17, 29, 38] as well as *strong equivalence models* [43, 44], which helps to understand and use the relation. The global nature of answer sets has also been addressed by compositionality results [10, 26, 30, 36] that relate the answer sets of an entire program with the answer sets of its parts meeting certain criteria. For example, *modular equivalence* [36] concerns so-called program *modules* and is suitable for justifying module-level transformations. In contrast with strong equivalence, however, it provides no help in analyzing transformations within modules on the level of rules, for example.

Strong equivalence is oblivious to encapsulation, which is a drawback that has defied a solution that precisely models standard answer-set solver behavior until this article, despite extensive existing research. The drawback is substantial, because encapsulation is ubiquitous in practical ASP. Indeed, answer-set solving tools provide means to partition the *signature* of a program by labeling atoms visible or hidden, for example via the purpose-built `#show` directive of the modern *grounder* GRINGO [18] or via similar constructs of the more historical grounder LPARSE.[1] Atoms left hidden take part in computation as usual but evade inclusion in the finally presented answer sets. Thereby encapsulation enables developers to express implementation details using hidden *auxiliary atoms* without burdening the view of solver users. Unfortunately, exercise of this ability immediately diminishes the usefulness of strong equivalence along with all other equivalence notions that do not distinguish visible and hidden atoms. From the perspective that hidden atoms are indeed implementation details, strong equivalence requires that not only are the answer sets of two programs identical in any context but also that the programs are implemented much the same. This becomes an issue in intuitively valid transformation scenarios where a part of a program is transformed, hidden atoms are changed, and user visible answer sets are preserved. Despite the lack of user visible differences, such cases contradict the strong equivalence of the old and new versions of the transformed parts. For instance, simplification of rules $a \leftarrow c$ and $c \leftarrow b$ into $a \leftarrow b$ breaks strong equivalence, despite an intention to hide c. There are two independent reasons for this. First, if the rules are in a context such as the fact c, then the answer sets change from $\{\{a, c\}\}$ to $\{\{c\}\}$, where even the visible parts differ. Although the context c is intuitively irrelevant due to its interference with a hidden atom, it provides a counterexample to strong equivalence. Second,

[1] http://www.tcs.hut.fi/Software/smodels/lparse.ps.

if the original rules form a program with the fact b, then the simplification changes its answer sets from $\{\{a, b, c\}\}$ to $\{\{a, b\}\}$. Although the visible parts of the answer sets stay intact, strong equivalence is again contradicted by the change.

The above considerations motivate the inspection of equivalence notions via two questions: (1) which context programs are relevant for establishing the equivalence and (2) in what sense are the programs compared once a context is given? Standard notions of equivalence do not comprehensively address encapsulation in answering these questions. To begin with, the basic notion of equivalence, also known as *weak equivalence*, simply classifies programs with the same sets of answer sets as equivalent. It (1) involves no context and (2) indiscriminately compares everything. Some existing approaches do focus on question (2) by tailoring program equivalence for use cases where a part of the signature is hidden. Indeed, the notion of *visible equivalence* [23] emphasizes the user's perspective, i.e., the answer sets printed out by the solver. Although (1) it involves no context, (2) it promisingly insists on a one-to-one correspondence between answer sets having identical projections on the visible signature. As a further benefit, verification of visible equivalence has an implementation for cases where hidden program parts behave deterministically and thus keep computational complexity in check[2] [25]. However, there are derivatives of strong equivalence that promisingly approach question (1) by accounting for context and auxiliary predicates [45, 46]. Namely, *relativized* variants of strong equivalence constrain context programs in terms of subsidiary signatures. However, this is reflected only in terms of (1) allowed context programs but not (2) considerations of visibility in a given context (cf. Examples 2.11 and 2.13 in Section 2). The general framework of *correspondence frames* [16, 40] does address both considerations (1) and (2) as it captures equivalence relations parameterized by arbitrary classes of context programs and arbitrary comparison relations. Whereas the scope of the framework is vast, the related studies are limited to specific types of comparison relations that compare answer sets projected to specified signatures. The respective notion of equivalence is called *relativized strong equivalence with projection*. The comparisons resemble visible equivalence checks [23] but do not stipulate a one-to-one correspondence between answer sets. Instead, the projective comparisons collapse multiple versions of visibly identical answer sets into one. Although such projective semantics may be desirable in some circumstances, it represents a mismatch between typical answer-set solver behavior, which is to enumerate all answer sets, including copies. Moreover, projection renders the equivalence notion unsuitable for applications where answer set counts bear significance. This notably includes probabilistic applications where auxiliary atoms are used to express frequencies of events [1, 3] and where multiple copies of answer sets are essential for the determination of probabilities.

In this article, we solve the shortcomings discussed above and pinpoint a notion that generalizes both visible and strong equivalence. Similarly to relativized strong equivalence with projection, it considers encapsulation both (1) in selecting relevant context programs and (2) in comparing programs in a given context. However, it is designed moreover to insist on a one-to-one correspondence between answer sets and to be a *congruence relation* for program union. The resulting notion is coined as *visible strong equivalence*. The combination of its properties makes it particularly applicable to the correctness analysis of program transformations that introduce auxiliary atoms. Moreover, it is distinctive its applicability whenever the preservation of answer set counts is desirable. Besides being a tool for formal correctness analysis, a further important application for visible strong equivalence is automatic verification. As regards our purposes, the foremost verification task is to check that the rule-level normalization tools devised in References [4–6] produce correct output. The contributions of this article can be summarized as follows:

[2]If all atoms are hidden, then visible equivalence reduces to model counting—a #P-hard problem.

- *Visible strong equivalence:* The visible strong equivalence relation is defined.
- *Model-based characterization:* Visible strong equivalence is shown to admit a sound and complete model-based characterization in terms of so-called *VSE-models*. That is, it is shown that the visibility-aware semantics of programs is captured by their VSE-models so that the visible strong equivalence of the programs can be determined by comparing their VSE-models. The characterization result is an aid that can help in proving or disproving visible strong equivalence.
- *Positive programs:* For positive programs, an easier, alternative model-based characterization is established using a simpler class of models, namely *hidden minimal models*. Whenever applicable, this further aids in assessing visible strong equivalence and simultaneously relativized strong equivalence with projection, because the two relations coincide on positive programs when the latter is parameterized appropriately for a given pair of programs at hand.
- *Nearly positive programs:* The benefits and simplicity of reasoning based on hidden minimal models is extended beyond positive programs by developing a proof strategy that applies to some "nearly" positive programs. As a downside, the strategy is only sound, not complete. It is applicable to programs in which all negative atoms intuitively serve the role of input literals. Prominent examples of such programs include singleton programs of frequently used rule types without repeated atoms, but not choice rules (see rules (1) and (3)–(5) in Section 2.1). As with positive programs, the results apply to relativized strong equivalence with projection as well.
- *Application:* To illustrate the benefits of visible strong equivalence, use of the characterization results is demonstrated by proving example encodings of choice and cardinality rules correct. The encodings constitute *normalizations* by virtue of consisting of only normal rules. These correctness results are noteworthy contributions in themselves, in addition to serving as valuable examples.
- *Implementation:* Automated verification of visible strong equivalence is implemented via translations and standard ASP solvers in analogy to Ref. [25]. The implementation forbids the use of hidden non-determinism due to underlying computational complexity. Despite this limitation, the class of remaining programs is meaningful in practice, as evidenced by the feasibility of the experiments highlighted next.
- *Experiments:* The scalability of the implementation is explored computationally on normalizations of choice and cardinality rules. As a significant by-product of the experiments, a number of nontrivial normalization implementations are proven correct for the explored parameter range, and the respective formal correctness results regarding normalization are double checked.
- *Normalization Verification Performance:* The experiments reveal that the automated comparison of cardinality rule normalizations against one another requires substantially less computation time than comparison against the original cardinality rule. This result reduces the computational effort required to test new normalization schemes, as their outputs can be compared against already verified normalizations as opposed to plain cardinality rules that lack internal structure.

The rest of this article is organized as follows. Basic concepts of logic programs and definitions of equivalence relations are recalled in Section 2, including an account of visibility-based variants from the literature. The proposed notion of visible strong equivalence is then worked out in Section 3 in close connection to relativized strong equivalence. Simplified proof strategies for positive and nearly positive programs are given in Section 4. Application of visible strong equivalence

to program transformations is exemplified in Section 5. Automated verification of visible strong equivalence by way of a translation is devised in Section 6. The translation is put to use in scalability experiments in Section 7. Contrast with other notions of equivalence is further discussed in Section 8. The article is concluded in Section 9.

2 BACKGROUND

This section has a number of objectives. First, in Section 2.1, we recall several kinds of rules used to form logic programs in ASP. The resulting standard classes of programs are also defined. The stable model semantics is then recalled in Section 2.2, after which the basic notions of equivalence, viz. weak and strong equivalence, are reviewed in Section 2.3. Finally, we prepare for a new *visibility-based* generalization of strong equivalence by surveying previous approaches in Section 2.4.

2.1 Frequently Used Classes of Logic Programs

In the sequel, we will study *propositional logic programs*, i.e., finite sets rules of the forms (1)–(5), where a, a_i's, b_j's, and c_k's are *propositional atoms* (or *atoms* for short) and $\sim$ denotes *default negation*,

$$a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m. \tag{1}$$

$$\{a_1, \ldots, a_h\} \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m. \tag{2}$$

$$a \leftarrow l \leq \{b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m\}. \tag{3}$$

$$a \leftarrow w \leq \{b_1 = w_{b_1}, \ldots, b_n = w_{b_n}, \sim c_1 = w_{c_1}, \ldots, \sim c_m = w_{c_m}\}. \tag{4}$$

$$a_1 \mid \cdots \mid a_h \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m. \tag{5}$$

The intuition behind a *normal* rule of the form (1) is that the *head* atom a can be derived whenever the *body conditions* of the rule are satisfied, i.e., when the *positive* body atoms $b_1, \ldots, b_n$ are derivable by the other rules in the program but none of the *negative* body atoms $c_1, \ldots, c_m$ are. An abbreviation $a \leftarrow B, \sim C$ of (1), where $B = \{b_1, \ldots, b_n\}$ and $C = \{c_1, \ldots, c_m\}$ will also be used. The other rule types (2)–(5) extend the idea of a normal rule as follows. The head $\{a_1, \ldots, a_h\}$ of a *choice rule* (2) denotes a choice to be made when the body of the rule is satisfied: Any of a_i's can be derived or none. A *cardinality rule* (3) is similar to a normal rule, but its body becomes already satisfied whenever the number of satisfied body conditions is at least l. More generally, the body of a *weight rule* (4) is satisfied if the sum of weights (denoted by w_{b_j}'s and w_{c_k}'s above) of satisfied body conditions is at least w. Finally, the head $a_1 \mid \cdots \mid a_h$ of a *disjunctive rule* (5) must be satisfied, i.e., at least one of the head atoms is (minimally) derived if the body is satisfied. We use analogous set-based short hands for the extended rule types like $a \leftarrow w \leq \{B = W_B, \sim C = W_C\}$ for a weight rule (4) and $A \leftarrow B, \sim C$ for a disjunctive rule (5).

Typical (syntactic) classes of logic programs are as follows. *Normal logic programs* (NLPs) solely consist of normal rules of the form (1). The fragment internally supported by the historical SMODELS solver, also known as SMODELS programs, is based on the forms (1)–(4). This fragment is also directly understood by the state-of-the-art solver CLASP [19]. The class of *weight constraint programs* (WCPs) allows more liberal use of cardinality and weight constraints, but WCPs are easy to translate into standard SMODELS programs using auxiliary atoms [41]. The class of *disjunctive logic programs* (DLPs) allows disjunctive rules (5), which includes normal rules (1) as their special case ($h = 1$). In the sequel, the specific class of considered logic programs is generally not crucial, and thus we use the term *logic program* (or *program* for short) to refer to any finite set of rules—even allowing for hybrid programs that mix the rule types above. This is also the spirit of the ASP-core-

2 language standard [9]. Finally, we say that a rule is *positive* if $m = 0$ and it is of the forms (1) or (3)–(5). A program P is called a positive program if its rules are all positive.

2.2 Stable Model Semantics

To define the formal semantics of programs, we write $\mathrm{At}(P)$ for the *signature*[3] of a program P, which is a set of atoms including all atoms occurring in P. An *interpretation* $I \subseteq \mathrm{At}(P)$ of P determines which atoms $a \in \mathrm{At}(P)$ are *true* ($a \in I$) and which atoms are *false* ($a \in \mathrm{At}(P) \setminus I$). Atoms are also called *positive literals*. Any *negative literal* $\sim c$, where c is an atom, is treated classically, i.e., $\sim c$ is satisfied in I, denoted $I \models \sim c$, iff $I \not\models c$. The satisfaction relation $\models$ extends to other arguments as follows. For a body $B \cup \sim C$ of a normal/choice/disjunctive rule, let $I \models B \cup \sim C$ iff $I \models b$ for each $b \in B$ and $I \models \sim c$ for each $c \in C$. Quite similarly, the body $l \leq \{B, \sim C\}$ of a cardinality rule is satisfied in I iff $l \leq |B \cap I| + |C \setminus I|$. This can be generalized for the body $w \leq \{B = W_B, \sim C = W_C\}$ of a weight rule—satisfied in I iff the *weight sum* $\mathrm{WS}_I(B = W_B, \sim C = W_C) = \sum_{b \in B \cap I} w_b + \sum_{c \in C \setminus I} w_c$ is at least w. A rule r is satisfied in I, denoted $I \models r$, iff the satisfaction of its body implies the satisfaction of its head. As special cases, the head $\{A\}$ of a choice rule is always satisfied in I, and the head A of a disjunctive rule is satisfied in I iff $A \cap I \neq \emptyset$. An interpretation $I \subseteq \mathrm{At}(P)$ is a (*classical*) *model* of a program P, denoted $I \models P$, iff $I \models r$ for each rule $r \in P$. A model $M \models P$ is a $\subseteq$-*minimal model* of a program P iff there is no other model $M' \models P$ such that $M' \subset M$. The set of minimal models of P is denoted by $\mathrm{MM}(P)$. If P is positive and it has no disjunctive rules (5), then $|\mathrm{MM}(P)| = 1$.

Definition 2.1 (Reduct). Given a program P and an interpretation $I \subseteq \mathrm{At}(P)$, the *reduct* of P with respect to I, denoted by P^M, contains

(1) a rule $a \leftarrow B$ for each normal rule $a \leftarrow B, \sim C$ in P such that $I \models \sim C$, and for each choice rule $\{A\} \leftarrow B, \sim C$ in P such that $a \in A \cap I$ and $I \models \sim C$;
(2) a rule $a \leftarrow l' \leq \{B\}$ for each cardinality rule $a \leftarrow l \leq \{B, \sim C\}$ in P, where $l' = \max(0, l - |C \setminus I|)$;
(3) a rule $a \leftarrow w' \leq \{B = W_B\}$ for each weight rule
$$a \leftarrow w \leq \{B = W_B, \sim C = W_C\}$$
in P, where $w' = \max(0, w - \mathrm{WS}_I(\sim C = W_C))$; and
(4) a rule $A \leftarrow B$ for each disjunctive rule $A \leftarrow B, \sim C$ in P such that $I \models \sim C$.

Lemma 2.2. *For a program P and $I \subseteq \mathrm{At}(P)$, $I \models P$ if and only if $I \models P^I$.*

Proof. Suppose that P contains a choice rule $r = \{A\} \leftarrow B, \sim C$. The rule is satisfied in any interpretation $I \subseteq \mathrm{At}(P)$ by definition. The head of every rule in the reduct $\{r\}^I$ is in I so that $I \models \{r\}^I$. Therefore, choice rules do not affect the iff-relationship.

Let us then consider a weight rule $a \leftarrow w \leq \{B = W_B, \sim C = W_C\}$ in P and any interpretation $I \subseteq \mathrm{At}(P)$. The reduct P^I contains the rule $a \leftarrow w' \leq \{B = W_B\}$ with $w' = \max(0, w - \mathrm{WS}_I(\sim C = W_C))$ unconditionally. Then, the following statements are equivalent:

- $I \not\models a \leftarrow w \leq \{B = W_B, \sim C = W_C\}$
- $w \leq \mathrm{WS}_I(B = W_B, \sim C = W_C)$ and $a \notin I$
- $w' \leq \mathrm{WS}_I(B = W_B)$ and $a \notin I$, as all the weights and weight limits are non-negative
- $I \not\models a \leftarrow w' \leq \{B = W_B\}$.

[3] Some related works assume a global universe $\mathcal{U}$ of atoms that serves as $\mathrm{At}(P)$ when necessary.

Normal rules (1) and cardinality rules (3) are covered as special cases of weight rules. The treatment of proper disjunctive rules (5) is analogous to that of normal rules except that the condition $a \notin I$ is replaced by $A \cap I = \emptyset$. □

It is worth noting that P^M is a positive program for any interpretation $M \subseteq \mathrm{At}(P)$ so that $\mathrm{MM}(P^M) \neq \emptyset$. The following generalizes the respective definitions of stable models from Refs. [20, 22, 41].

Definition 2.3 (Stable Model). An interpretation $M \subseteq \mathrm{At}(P)$ of a *logic program* P is a *stable model* of P if and only if $M \in \mathrm{MM}(P^M)$.

Example 2.4. Consider a logic program P consisting of the following rules:

$$a \mid b. \qquad \{c\} \leftarrow b.$$

The program has three stable models: $M_1 = \{a\}$, $M_2 = \{b\}$, and $M_3 = \{b, c\}$. To verify the last one, note that P^{M_3} is $\{a \mid b.\ \ c \leftarrow b.\ \}$ having minimal models $\{a\}$ and $\{b, c\}$. ■

The number of stable models of P, also known as the *answer sets* of P, can vary in general and we let $\mathrm{SM}(P)$ stand for the set of stable models associated with P.

2.3 Basic Notions of Equivalence

The formal semantics provided by stable models gives rise to a straightforward notion of equivalence. Given two logic programs P and Q, they are defined to be *weakly equivalent*, denoted $P \equiv Q$, if and only if $\mathrm{SM}(P) = \mathrm{SM}(Q)$, i.e., the stable models of P and Q are exactly the same subsets of $\mathrm{At}(P)$ and $\mathrm{At}(Q)$, respectively. The syntactic class of programs is not crucial for the definition so that even inter-class comparisons using $\equiv$ make sense. The definition of strong equivalence [29], however, assumes a context program R, which makes its definition specific to a particular class of programs. So, given logic programs P and Q from the class of interest, they are *strongly equivalent*, denoted $P \equiv_\mathrm{s} Q$, if and only if $P \cup R \equiv Q \cup R$ for any logic program R from the same class. By setting $R = \emptyset$, it is easy to see that $P \equiv_\mathrm{s} Q$ implies $P \equiv Q$. The converse does not hold in general as illustrated by the following example.

Example 2.5. Consider logic programs $P = \{a \mid b.\ \}$ and $Q = \{a \leftarrow {\sim}b.\ b \leftarrow {\sim}a.\ \}$. The sets of stable models are $\mathrm{SM}(P) = \{\{a\}, \{b\}\} = \mathrm{SM}(Q)$ so that $P \equiv Q$ holds. The programs are not strongly equivalent as witnessed by the context $R = \{a \leftarrow b.\ b \leftarrow a.\ \}$: $\mathrm{SM}(P \cup R) = \{\{a, b\}\}$ and $\mathrm{SM}(Q \cup R) = \emptyset$. ■

In contrast to weak equivalence, strong equivalence allows for substitutions of mutually equivalent programs in arbitrary contexts. Hence $\equiv_\mathrm{s}$ is a *congruence relation* for $\cup$, i.e., $P \equiv_\mathrm{s} Q$ implies $P \cup R \equiv_\mathrm{s} Q \cup R$ for any context program R from the class under consideration. It is possible to characterize strong equivalence without explicitly referring to context programs [43, 44]. A strong equivalence model (SE-model for short) of a program P is a pair $\langle X, Y \rangle$ of interpretations where $X \subseteq Y \subseteq \mathrm{At}(P)$, $Y \models P$, and $X \models P^Y$. The set of SE-models of P is denoted by $\mathrm{SE}(P)$. It follows for any programs P and Q that $P \equiv_\mathrm{s} Q$ if and only if $\mathrm{SE}(P) = \mathrm{SE}(Q)$ [43].

Example 2.6. The difference between P and Q from Example 2.5 is captured by an SE-model $\langle \emptyset, \{a, b\} \rangle$ of Q that is not an SE-model of P. To verify this, we note that $\{a, b\} \models Q$, $Q^{\{a,b\}} = \emptyset$, so that $\emptyset \models Q^{\{a,b\}}$ but $\emptyset \not\models a \mid b$ for the rule $a \mid b$ in $P^{\{a,b\}}$. ■

The characterization in terms of SE-models makes the notion of strong equivalence independent of the class of programs under consideration—enabling the substitution of P by Q, or vice versa, as long as the respective unions $P \cup R$ and $Q \cup R$ are well defined in the context R. In case $P \not\equiv_\mathrm{s} Q$ holds, the space of witnessing contexts R, for which $\mathrm{SM}(P \cup R) \neq \mathrm{SM}(Q \cup R)$ holds, appears to be

much larger than that of countermodels certifying $\mathrm{SE}(P) \neq \mathrm{SE}(Q)$. However, there are results [29, 46], indicating that contexts consisting of *unary* rules only are sufficient. In our case, such rules are obtained as special cases of (1)–(5) when $n = 1$ and $m = 0$. Actually, any *countermodel* $\langle X, Y \rangle$ for strong equivalence can be mapped to a context program $R(X, Y)$ that contains every atom $x \in X$ as a fact and for each pair of atoms $y_1, y_2 \in Y \setminus X$ with $y_1 \neq y_2$, a unary rule $y_1 \leftarrow y_2$. In Example 2.6, we obtain the context $R(\emptyset, \{a, b\}) = \{a \leftarrow b.\ b \leftarrow a.\ \}$ that was used in Example 2.5 to establish difference.

2.4 Visibility-based Variants

A typical answer-set program involves auxiliary atoms that formalize some secondary concepts for the problem being solved. They are not directly needed to inspect a solution to the problem and hence it is customary to hide them from the user whereas the rest of atoms remain visible. To formalize this idea, we follow the approach of Reference [23] and write $\mathrm{At_v}(P)$ and $\mathrm{At_h}(P)$ for the *visible* and *hidden* signatures of P, respectively, so that $\mathrm{At}(P) = \mathrm{At_v}(P) \cup \mathrm{At_h}(P)$ and $\mathrm{At_v}(P) \cap \mathrm{At_h}(P) = \emptyset$. The idea is that given $M \in \mathrm{SM}(P)$ only $M \cap \mathrm{At_v}(P)$ is visible to the user and relevant when comparing P with other programs. Given an interpretation $I \subseteq \mathrm{At}(P)$ and a set of atoms $A \subseteq \mathrm{At}(P)$, we write $I|_A$ for the *projection* $I \cap A$ of the interpretation I over A and, in particular, I_v and I_h for the respective projections $I|_{\mathrm{At_v}(P)}$ and $I|_{\mathrm{At_h}(P)}$. We extend these notations for sets of interpretations $S \subseteq 2^{\mathrm{At}(P)}$ by $S|_A = \{I|_A \mid I \in S\}$, $S_\mathrm{v} = \{I_\mathrm{v} \mid I \in S\}$, and analogously for S_h. Thus, e.g., $\mathrm{SM}(P)_\mathrm{v} = \{M_\mathrm{v} \mid M \in \mathrm{SM}(P)\}$.

The basic relations $\equiv$ and $\equiv_\mathrm{s}$ introduced in Section 2.3 count on all atoms being visible, i.e., $\mathrm{At_v}(P) = \mathrm{At}(P)$ and $\mathrm{At_h}(P) = \emptyset$ hold for programs subject to comparison. The notion of *visible equivalence* [23], denoted by $\equiv_\mathrm{v}$, generalizes $\equiv$ to comparisons where the visibility of atoms matters. Projections are compared as follows.

Definition 2.7. Given logic programs P and Q such that $\mathrm{At_v}(P) = \mathrm{At_v}(Q)$, the sets of interpretations $S_1 \subseteq 2^{\mathrm{At}(P)}$ and $S_2 \subseteq 2^{\mathrm{At}(Q)}$ are *visibly equal*, denoted by $S_1 =_\mathrm{v} S_2$, if and only if there is a bijection $f : S_1 \to S_2$ such that for every $I \in S_1$, $I_\mathrm{v} = f(I)_\mathrm{v}$.

Definition 2.8 (Visible Equivalence [23]). Logic programs P and Q are *visibly equivalent*, denoted by $P \equiv_\mathrm{v} Q$, iff $\mathrm{At_v}(P) = \mathrm{At_v}(Q)$ and $\mathrm{SM}(P) =_\mathrm{v} \mathrm{SM}(Q)$.

It is straightforward to show that both $=_\mathrm{v}$ and $\equiv_\mathrm{v}$ are equivalence relations in their respective domains. The strict (bijective) correspondence of stable models underlying $\equiv_\mathrm{v}$ stems from the goal of formalizing the user's perspective in the presence of hidden atoms: It is possible that a particular projection of stable models is printed out by the solver multiple times, because the mutual differences concern only hidden atoms. From a more theoretical perspective, such a tight relationship caters to the interests of users of various different reasoning modes of ASP such as *brave* and *cautions* reasoning as well as *counting* of stable models.

Example 2.9. If $\mathrm{SM}(P) = \{\{a\}, \{b\}\}$ and $\mathrm{SM}(Q) = \{\emptyset, \{b\}\}$, then $P \equiv_\mathrm{v} Q$ holds if $\mathrm{At_v}(P) = \{b\} = \mathrm{At_v}(Q)$, i.e., a is hidden. It is easy to see that b is only a brave but not a cautious consequence with respect to both P and Q. That is, in both cases, b is in some but not all stable models. However, if a is made visible, a bijective mapping between stable models in the sense of $=_\mathrm{v}$ becomes impossible. ■

Definition 2.8 treats the visible projections of $\mathrm{SM}(P)$ and $\mathrm{SM}(Q)$ as multisets, i.e., the number of copies of each projection with respect to $\mathrm{At_v}(P)$ and $\mathrm{At_v}(Q)$ matters. Indeed, the condition $\mathrm{SM}(P) =_\mathrm{v} \mathrm{SM}(Q)$ differs from $\mathrm{SM}(P)_\mathrm{v} = \mathrm{SM}(Q)_\mathrm{v}$, which would lead us to the *projective notions* of equivalence addressed in Reference [16]. It is also worth pointing out that the definition of $\equiv_\mathrm{v}$ is

independent of the syntax of programs, which enables natural comparison of programs belonging to different syntactic classes.

There are also existing generalizations of strong equivalence [29], which take the visibility of atoms into account. For instance, the idea behind a *relativized* version of strong equivalence [45] is to constrain potential contexts R using a fixed set of atoms A such that $\mathrm{At}(R) \subseteq A$. The concept can be further refined [46] by introducing distinct sets of atoms H and B to constrain head and body atoms appearing in the rules of context programs, respectively. In the case that H and B coincide, the more general notion reduces to relativized strong equivalence whose definition is recalled next.

Definition 2.10 (Relativized Strong Equivalence [45]). Logic programs P and Q are strongly equivalent relative to A, denoted by $P \equiv_{\mathrm{s}}^{A} Q$, if and only if $\mathrm{SM}(P \cup R) = \mathrm{SM}(Q \cup R)$ for all contexts R with $\mathrm{At}(R) \subseteq A$.

Definitions 2.8 and 2.10 yield different relations even in the interesting special case where $\mathrm{At_v}(P) = A = \mathrm{At_v}(Q)$. In this setting, the context program R may interact with P and Q through visible atoms only, and in this sense R respects the hidden atoms of P and Q. However, visible equivalence allows those hidden atoms to differ, whereas relativized strong equivalence does not, as can be seen in the example below.

Example 2.11. Consider a program P consisting of a single choice rule $\{a\} \leftarrow {\sim}b$ and let Q be its tentative translation to *normal* rules $a \leftarrow {\sim}\overline{a}, {\sim}b$ and $\overline{a} \leftarrow {\sim}a$, where $\overline{a}$ is an auxiliary atom to be hidden, i.e., let $\mathrm{At_v}(P) = \{a, b\} = \mathrm{At_v}(Q)$ and $\overline{a} \in \mathrm{At_h}(Q)$. For the context $R = \emptyset$, we obtain $\mathrm{SM}(P \cup R) = \{\emptyset, \{a\}\}$ and $\mathrm{SM}(Q \cup R) = \{\{\overline{a}\}, \{a\}\}$ so that $P \not\equiv_{\mathrm{s}}^{\{a,b\}} Q$. For comparison, we have $P \cup R \equiv_{\mathrm{v}} Q \cup R$. Thus $\equiv_{\mathrm{s}}^{\{a,b\}}$ does not capture our visibility-focused view about the translation Q, by which Q should be equivalent to P in any context that respects the definition of $\overline{a}$, such as R. ■

This example suggests potential relaxations of Definition 2.10, allowing stable models to differ over atoms in $\mathrm{At_h}(P)$ and $\mathrm{At_h}(Q)$. Designing such a variant shall be the topic of Section 3. Before that, we recall the definition of *A-SE-models* below. These models are interesting, because $\equiv_{\mathrm{s}}^{A}$ can be characterized in terms of them: $P \equiv_{\mathrm{s}}^{A} Q$ holds iff $\mathrm{SE}^{A}(P) = \mathrm{SE}^{A}(Q)$ holds for the sets of A-SE-models associated with P and Q, respectively.

Definition 2.12 (Relativized SE-Models [45]).

(1) A pair $\langle X, Y\rangle$ of interpretations is an A-SE-interpretation of P if and only if $Y \subseteq \mathrm{At}(P)$ and either $X = Y$ or $X \subseteq Y\,|_A$.
(2) An A-SE-interpretation $\langle X, Y\rangle$ of P is an A-SE-model of P if and only if (i) $Y \models P$, (ii) there is no $Y' \subset Y$ such that $Y'\,|_A = Y\,|_A$ and $Y' \models P^Y$, and (iii) if $X \subset Y$, then there is $X' \subseteq Y$ such that $X'\,|_A = X$ and $X' \models P^Y$.

The idea is that Y is a $\subseteq$-minimal model of P^Y when the interpretation of the atoms in A, i.e., $Y\,|_A$, is kept *fixed* in the sense of *parallel circumscription* [27, 32]. The third condition ensures that any *non-total* $X \subset Y$ interpretation can be extended to a model X' of P^Y such that $X'\,|_A = X$. Note that $Y \models P$ implies $Y \models P^Y$ by the fact that taking the reduct P^Y partially evaluates the negative body literals of P with respect to Y. Thus, for the interpretations X and Y of an A-SE-model $\langle X, Y\rangle$, being a model of P^Y is an essential property, since Y has this property and X can be extended to have the same property by assigning values to atoms residing outside A.

Example 2.13. Recall the program $P = \{\{a\} \leftarrow {\sim}b.\ \}$ from Example 2.11 and its potential normalization $Q = \{a \leftarrow {\sim}\overline{a}, {\sim}b.\ \ \overline{a} \leftarrow {\sim}a.\ \}$ subject to $A = \{a, b\}$. Due to the choice rule of P, any subset Y of $\mathrm{At}(P) = \{a, b\}$ is a model of P. The reduct $P^Y = \emptyset$ except when $Y = \{a\}$, which implies $P^Y = \{a.\ \}$.

Thus the A-SE-models of P contain $\langle\{a\},\{a\}\rangle$ and all A-SE-interpretations of the form $\langle X, Y\rangle$ where $X \subseteq Y$ and $Y \neq \{a\}$.

The classical models $Y \subseteq \mathrm{At}(Q) = \{a, \overline{a}, b\}$ of Q that satisfy the conditions (i) and (ii) of Definition 2.12 in item (2) are $Y_1 = \{\overline{a}\}$, $Y_2 = \{a\}$, $Y_3 = \{\overline{a}, b\}$, and $Y_4 = \{a, b\}$. Note that the minimality of models $\{a, \overline{a}\}$ and $\{a, \overline{a}, b\}$ is defeated by Y_2 and Y_4, respectively. For Y_1, we obtain $Q^{Y_1} = \{\overline{a}.\ \}$ and an $\{a, b\}$-SE-model $\langle\emptyset, \{\overline{a}\}\rangle$. Similarly, $Q^{Y_2} = \{a.\ \}$ giving rise to $\langle\{a\},\{a\}\rangle$. The reduct $Q^{Y_3} = \{\overline{a}.\ \}$ results in $\langle X_3, Y_3\rangle$ with $X_3 \subseteq \{b\}$ and, finally, $Q^{Y_4} = \emptyset$ in $\langle X_4, Y_4\rangle$ where $X_4 \subseteq \{a, b\}$. The $\{a, b\}$-SE-models $\langle\emptyset, \{\overline{a}\}\rangle$, $\langle\emptyset, \{\overline{a}, b\}\rangle$, and $\langle\{b\}, \{\overline{a}, b\}\rangle$ of Q make the difference with respect to the $\{a, b\}$-SE-models of P. Again, the difference is due to the hidden atom $\overline{a}$ involved in Q and its interpretations. ■

Eiter et al. [16] present a general framework to capture a variety of program equivalences. According to their definitions, a *correspondence frame* $\mathcal{F}$ is a triple $\langle \mathcal{U}, \mathcal{C}, \rho\rangle$, where $\mathcal{U}$ is a set of atoms known as the *universe* of $\mathcal{F}$, $\mathcal{C}$ is a class of *context programs* based on $\mathcal{U}$, and ρ is a relation used to compare stable models. Given two programs P and Q based on the signature $\mathcal{U}$ of $\mathcal{F}$, they are deemed $\mathcal{F}$*-corresponding*, if for all context programs $R \in \mathcal{C}$, the relation ρ holds between $\mathrm{SM}(P \cup R)$ and $\mathrm{SM}(Q \cup R)$. Furthermore, Eiter et al. [16] write $\mathcal{P}_A$ for the class of programs based on a set of atoms $A \subseteq \mathcal{U}$. Since the concepts of visible $\mathrm{At}_\mathrm{v}(\cdot)$ and hidden $\mathrm{At}_\mathrm{h}(\cdot)$ signatures are not used therein, the programs $R \in \mathcal{P}_A$ can be understood to have no hidden atoms in the context of this article so that $\mathrm{At}_\mathrm{h}(R) = \emptyset$. By these definitions, the weak and strong equivalence of two programs $P, Q \in \mathcal{P}_\mathcal{U}$ are captured by $\langle \mathcal{U}, \{\emptyset\}, =\rangle$-correspondence and $\langle \mathcal{U}, \mathcal{P}_\mathcal{U}, =\rangle$-correspondence, respectively. Moreover, the case of relativized strong equivalence is obtained as $\langle \mathcal{U}, \mathcal{P}_A, =\rangle$-correspondence for any $A \subseteq \mathcal{U}$. Using the framework, the study [16] introduces a relation called *relativized strong equivalence with projection*, which addresses differences arising from auxiliary atoms. Besides the set $A \subseteq \mathcal{U}$ used to limit the class of context programs $\mathcal{P}_A$, the relation introduces an additional set B of atoms to project stable models before comparison. In terms of correspondence frames, the relation is captured by $\langle \mathcal{U}, \mathcal{P}_A, =_B\rangle$-correspondence where $=_B$ stands for the equality of B-projections of stable models. To generalize Definition 2.10 to reflect the correspondence frame, we formulate the resulting weakening of relativized strong equivalence as follows.

Definition 2.14 (Relativized Strong Equivalence with Projection [16]). Logic programs P and Q are strongly equivalent relative to A up to projection with respect to B, denoted by $P \equiv_\mathrm{s}^{A,B} Q$, if and only if $\mathrm{SM}(P \cup R)\,|_B = \mathrm{SM}(Q \cup R)\,|_B$ for all context programs R with $\mathrm{At}(R) \subseteq A$.

Example 2.15. Recalling programs P and Q from Example 2.11, their differences essentially disappear, e.g., if we pick $A = \{a, b\} = B$. Then, we obtain $\mathrm{SM}(P \cup R)\,|_B = \{\emptyset, \{a\}\} = \mathrm{SM}(Q \cup R)\,|_B$ for the context $R = \emptyset$ as intuitively desired. ■

It is obvious that the set B in Definition 2.14 serves the purpose of a *visible signature* for programs under consideration. However, the signature A determines the contexts of interest and the interface for interaction. Intriguingly, the A-SE-models introduced in Definition 2.12 are insufficient to characterize the projective variant, and more complex structures such as *certificates* are needed [16]. More formally, each interpretation $Y \subseteq \mathrm{At}(P)$ such that $\langle Y, Y\rangle \in \mathrm{SE}^A(P)$ induces an $\langle A, B\rangle$-certificate $\langle \mathcal{X}, Y \cap (A \cup B)\rangle$ of P where $\mathcal{X}$ consists of all interpretations $X \subset Y$ such that $\langle X, Y\rangle \in \mathrm{SE}^A(P)$. By construction $Y \cap A$ is excluded from the first component of a certificate, and thus the set $\mathcal{X}$ becomes empty if P^Y cannot be satisfied by any smaller interpretation than Y. There can also be several interpretations Y inducing the same projection over $A \cup B$ as the second component, and thus Eiter et al. [16] distinguish a *minimal* $\langle A, B\rangle$*-certificate* $\langle \mathcal{X}, Y \cap (A \cup B)\rangle$ of P as

a certificate of P for which there is no other certificate $\langle \mathcal{X}', Y \cap (A \cup B)\rangle$ of P such that $\mathcal{X}' \subset \mathcal{X}$. Whenever $A = B$ holds, we call $\langle A, B\rangle$-certificates A-certificates for short.

Theorem 2.16. *[16] Logic programs P and Q are strongly equivalent relative to A up to projection with respect to B, if and only if the minimal $\langle A, B\rangle$-certificates of P and Q coincide.*

Example 2.17. As pointed out in Example 2.13, the differentiating $\{a, b\}$-SE-models of Q are $\langle \emptyset, \{\overline{a}\}\rangle$, $\langle \emptyset, \{\overline{a}, b\}\rangle$, and $\langle \{b\}, \{\overline{a}, b\}\rangle$. However, if we project the $\{a, b\}$-SE-models of Q onto $\{a, b\}$, only the following four $\{a, b\}$-certificates remain:

$$\langle \emptyset, \emptyset\rangle,\ \langle \emptyset, \{a\}\rangle,\ \langle \{\emptyset\}, \{b\}\rangle, \text{ and } \langle \{\emptyset, \{a\}, \{b\}\}, \{a, b\}\rangle.$$

The certificates are trivially minimal, since their second components are unique. Given that $\mathrm{SE}(P) = \mathrm{SE}^{\{a,b\}}(P)$, it is also easy to inspect that the minimal $\{a, b\}$-certificates of P are the same. Therefore, we may conclude by Theorem 2.16 that P and Q are strongly equivalent relative to $\{a, b\}$ up to projection with respect to $\{a, b\}$. ■

The fact that non-minimal certificates do not affect program equivalence in the sense of $\equiv_s^{A,B}$ is illustrated by the following example (communicated by S. Woltran).

Example 2.18. Consider two logic programs P and Q without stable models:

$$P\colon\quad \begin{array}{ll} a \leftarrow \sim a, \sim b. & b \leftarrow \sim a, \sim b. \\ a \leftarrow b. & b \leftarrow a. \end{array} \qquad Q\colon\quad \begin{array}{ll} a \leftarrow \sim a, \sim b. & b \leftarrow \sim a, \sim b. \\ a \leftarrow b. & b \leftarrow a, \sim c. \\ a \leftarrow c. & b \leftarrow c. \\ a \leftarrow d. & b \leftarrow d. \\ \multicolumn{2}{c}{c \mid d \leftarrow a, b.} \end{array}$$

The only classical model of P is $Y = \{a, b\}$, and since P^Y contains only the rules $a \leftarrow b$ and $b \leftarrow a$, there are two SE-models $\langle \emptyset, Y\rangle$ and $\langle Y, Y\rangle$. Thus, P has a unique $\{a, b\}$-certificate $\langle \{\emptyset\}, \{a, b\}\rangle$. As regards the program Q, we consider projections with respect to $\{a, b\}$, which amounts to hiding c and d. The models $Y_1 = \{a, b, c\}$ and $Y_2 = \{a, b, d\}$ induce $\{a, b\}$-certificates $\langle \{\emptyset, \{a\}\}, \{a, b\}\rangle$ and $\langle \{\emptyset\}, \{a, b\}\rangle$, respectively. Only the latter certificate is minimal. Since P and Q have identical minimal $\{a, b\}$-certificates, we know by Theorem 2.16 that $\mathrm{SM}(P \cup R)\,|_{\{a,b\}} = \mathrm{SM}(Q \cup R)\,|_{\{a,b\}}$ for any context program R with $\mathrm{At}(R) \subseteq \{a, b\}$. ■

The relativized strong equivalence with projection established in Example 2.17 indicates that the programs involved, i.e., P and its normalization Q, have the same meaning in a number of contexts. However, since the set A is fixed to $A = \{a, b\}$ in the relation $\equiv_s^{A,B}$, the restriction $\mathrm{At}(R) \subseteq A$ limits the scope of the established correspondence between P and Q to context programs R having no further atoms. Therefore, to establish a respective correspondence having relevance to a larger class of context programs, the parameters A and B need to be picked appropriately, not only to avoid the hidden signatures of P and Q, but also to cover the signatures of any potential context programs of interest. Aiming to simplify such questions related to signatures in the sequel, we will employ an alternative approach relying on program interfaces while defining a variant of strong equivalence that is readily applicable in a broad class of contexts. The idea is to equip programs with signatures together with visibility information so that rather than embedding such information in the equivalence relation. Consequently, the choice of an appropriate relation is streamlined. Afterward, in Section 3.5, we will return to discuss signatures while contrasting correspondence frames with the equivalence relation to be devised in the following.

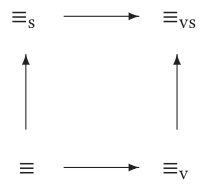

Fig. 1. Orthogonal generalizations of weak equivalence (≡).

3 VISIBLE STRONG EQUIVALENCE

The goal of this section is to make visible equivalence context-specific by incorporating context programs in analogy to strong equivalence. Our strategy is illustrated in Figure 1 where weak equivalence ≡ is generalized in two orthogonal ways as strong equivalence $\equiv_s$ and as visible equivalence $\equiv_v$. The essential question is whether these ways to generalize ≡ can actually meet—hence arriving at a new relation, *visible strong equivalence* $\equiv_{vs}$, in the upper right corner. It turns out in the sequel that such a generalization is indeed feasible and relaxes some limitations of existing generalizations of strong equivalence. In particular, we strive for a stronger characterization theorem using *visibility-based* SE-models, to establish wider applicability for the resulting variant of strong equivalence. Some subsidiary results are first derived with respect to classical equivalence and ordinary visible equivalence in Sections 3.1 and 3.2, respectively. The two halves of the characterization theorem, i.e., soundness and completeness with respect to SE-models, are then established in Sections 3.3 and 3.4. Finally, Section 3.5 demonstrates how the VSE relation can be represented as a correspondence frame.

Let us use the projective variant $\equiv_s^{A,B}$ of relativized strong equivalence from Definition 2.14 as a starting point for our analysis. When comparing two programs P and Q in a context R, it is natural to choose the signature B for projections based on visible atoms. Using the notations introduced in the context of visible equivalence, we obtain $B = \mathrm{At}_v(P) \cup \mathrm{At}_v(R) = \mathrm{At}_v(Q) \cup \mathrm{At}_v(R)$ where $\mathrm{At}_v(P) = \mathrm{At}_v(Q)$ in analogy to visible equivalence. Due to the presumed interaction of P and Q with the context R, it is reasonable to expect that $\mathrm{At}_v(P) \cap \mathrm{At}_v(R) = \mathrm{At}_v(Q) \cap \mathrm{At}_v(R)$ is non-empty, although this is not absolutely necessary. Contexts R that do not interact with P and Q are uninteresting and should not affect their equivalence. As regards the relativized strong equivalence, the context program P is constrained by the signature A, i.e., $\mathrm{At}(R) \subseteq A$. This leaves completely open how R might interact with P and Q as far as the hidden parts of the programs are concerned. Since visible equivalence makes the visibility of atoms explicit, a part of our solution is to restrict context programs on the basis of hidden atoms. For instance, if the context program R should not interfere with the hidden part of P, then $\mathrm{At}_h(P) \cap A = \emptyset$ should hold. However, rather than deriving complex constraints for the signature A, we formulate a more generic notion by saying that logic programs P and R *mutually respect* the hidden atoms of each other iff $\mathrm{At}(P) \cap \mathrm{At}_h(R) = \emptyset$ and $\mathrm{At}_h(P) \cap \mathrm{At}(R) = \emptyset$. Note that this condition is completely symmetric for P and R and can be similarly stated for Q and R in particular. For instance, given the normalization Q from Example 2.13, no context R such that $\overline{a} \in \mathrm{At}(R)$ respects the hidden atoms of Q. The following definition rules out such contexts when comparing programs in general.

Definition 3.1 (Visible Strong Equivalence). Programs P and Q are *visibly strongly equivalent*, denoted by $P \equiv_{vs} Q$, if and only if $\mathrm{At}_v(P) = \mathrm{At}_v(Q)$ and $\mathrm{SM}(P \cup R) =_v \mathrm{SM}(Q \cup R)$ for any context R that mutually respects the hidden atoms of P and Q.

It should be emphasized that Definition 3.1 does not require that P and Q mutually respect their hidden atoms. Thus $\mathrm{At}_h(P)$ and $\mathrm{At}_h(Q)$ may overlap and the respective definitions of (shared) hidden atoms in P and Q need not coincide. For instance, Q could be a rewrite of P obtained by

simply permuting the names of hidden atoms but regardless of this, we may argue that P and Q are equivalent, since their difference is hidden so that stable models are not effectively altered. Moreover, the context R may utilize its own hidden atoms if appropriate, since P and Q cannot refer to them.

Proposition 3.2 (Congruence for $\cup$). *If $P \equiv_{\mathrm{vs}} Q$ and R is a (context) program that mutually respects the hidden atoms of P and Q, then $P \cup R \equiv_{\mathrm{vs}} Q \cup R$.*

Given $M \models P$, we say that M is $\mathrm{At_h}(P)$-minimal if and only if there is no $N \models P$ such that $N_\mathrm{v} = M_\mathrm{v}$ and $N_\mathrm{h} \subset M_\mathrm{h}$. Due to the importance of the $\mathrm{At_h}(P)$-minimal models of P, we call them *hidden minimal* models of P and denote the respective set of models by $\mathrm{MM_h}(P)$. Moreover, we assume that computing the reduct P^Y for any $Y \subseteq \mathrm{At}(Y)$ does not interfere with the visibility of atoms, i.e., $\mathrm{At_v}(P^Y) = \mathrm{At_v}(P)$ and $\mathrm{At_h}(P^Y) = \mathrm{At_h}(P)$. The *visibility-based* SE-models (VSE-models for short) are defined as follows.

Definition 3.3. A VSE-model of a logic program P is a pair $\langle X, Y\rangle$ of interpretations where $X \subseteq Y \subseteq \mathrm{At}(P)$, $X \in \mathrm{MM_h}(P^Y)$, and $Y \in \mathrm{MM_h}(P^Y)$.

The set of VSE-models of P is denoted by $\mathrm{VSE}(P)$. The intuition of a VSE-model $\langle X, Y\rangle$ is that the second component Y represents a context for P against which the rules of P are reduced (to form P^Y) and $Y \models P$. Within Y, visible atoms are interpreted classically whereas hidden ones are false by default. The first component X is a model capturing a potential closure of P^Y also defined to be hidden minimal.

Example 3.4. Consider the logic program $P = \{a \mid b.\ \}$ from preceding examples and $Q = \{a \leftarrow c.\ \ b \leftarrow d.\ c \mid d.\ \}$ subject to $\mathrm{At_v}(P) = \{a, b\} = \mathrm{At_v}(Q)$. The program P has five SE/VSE-models: $\langle\{a\}, \{a\}\rangle$, $\langle\{b\}, \{b\}\rangle$, $\langle\{a\}, \{a, b\}\rangle$, $\langle\{b\}, \{a, b\}\rangle$, and $\langle\{a, b\}, \{a, b\}\rangle$. However, there are six VSE-models for Q: $\langle\{a, c\}, \{a, c\}\rangle$, $\langle\{b, d\}, \{b, d\}\rangle$, $\langle\{a, c\}, \{a, b, c\}\rangle$, $\langle\{b, d\}, \{a, b, d\}\rangle$, $\langle\{a, b, c\}, \{a, b, c\}\rangle$, and $\langle\{a, b, d\}, \{a, b, d\}\rangle$. ■

Since Definition 3.3 reformulates Definition 2.12, we relate them when $A = \mathrm{At_v}(P)$:

- If $\langle X, Y\rangle$ is an A-SE model of P, then Y is a hidden minimal model of P^Y and there is a hidden minimal model X' of P^Y such that $X'_\mathrm{v} = X_\mathrm{v}$ and $\langle X', Y\rangle \in \mathrm{VSE}(P)$.
- If $\langle X, Y\rangle \in \mathrm{VSE}(P)$, then $X = Y$ implies that $\langle X, Y\rangle$ is an A-SE model of P and, otherwise, $\langle X_\mathrm{v}, Y\rangle$ is an A-SE model of P.

It is also worth pointing out two corner cases of Definitions 3.1 and 3.3. If $\mathrm{At_h}(P) = \emptyset$ holds for the programs under consideration, then $\mathrm{VSE}(P) = \mathrm{SE}(P)$ and $\equiv_{\mathrm{vs}}$ coincides with $\equiv_\mathrm{s}$. However, if $\mathrm{At_v}(P) = \emptyset$, then $\mathrm{VSE}(P) = \{\langle M, M\rangle \mid M \in \mathrm{SM}(P)\}$ and $\equiv_{\mathrm{vs}}$ coincides with $\equiv_\mathrm{v}$ subject to $\mathrm{At_v}(P) = \emptyset$, so that in which case only the numbers of stable models are being compared. Lemma 2.2 and hidden minimality imply the following.

Proposition 3.5. *If $\langle X, Y\rangle \in \mathrm{VSE}(P)$, then (i) $Y \models P$ and (ii) $X = Y$ if $X_\mathrm{v} = Y_\mathrm{v}$.*

The latter part shows how *total* SE-models in the sense of Definition 2.12 are captured with VSE-models. Our next objective is to define when two sets of VSE-models can be considered to be the same. In contrast to References [45, 46], we partly disregard hidden information at this point. To ease the treatment of VSE-interpretations in the sequel, we adopt a few operations that are familiar from database theory. Given a set S of VSE-interpretations, define two *projections* of S by setting $\pi_1(S) = \{X \mid \langle X, Y\rangle \in S\}$ and $\pi_2(S) = \{Y \mid \langle X, Y\rangle \in S\}$. Moreover, given an interpretation $Z \subseteq \mathrm{At}(P)$, we may *select* the set of VSE-interpretations $\sigma_Z(S) = \{\langle X, Y\rangle \in S \mid Y = Z\}$. The same

notation is used for $Z_{\mathrm{v}} \subseteq \mathrm{At}_{\mathrm{v}}(P)$, but then the selection condition is $Y_{\mathrm{v}} = Z_{\mathrm{v}}$. Sets of VSE-models are compared as follows to capture $\equiv_{\mathrm{vs}}$.

Definition 3.6. Given programs P and Q such that $\mathrm{At}_{\mathrm{v}}(P) = \mathrm{At}_{\mathrm{v}}(Q)$, the respective sets $\mathrm{VSE}(P)$ and $\mathrm{VSE}(Q)$ *visibly match*, denoted $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$, if and only if

(1) $\pi_2(\mathrm{VSE}(P)) =_{\mathrm{v}} \pi_2(\mathrm{VSE}(Q))$ via a bijection f and
(2) for each matching pair of models $Y \in \pi_2(\mathrm{VSE}(P))$ and $f(Y) \in \pi_2(\mathrm{VSE}(Q))$,

$$\pi_1(\sigma_Y(\mathrm{VSE}(P)))_{\mathrm{v}} = \pi_1(\sigma_{f(Y)}(\mathrm{VSE}(Q)))_{\mathrm{v}}. \tag{6}$$

Example 3.7. Given the sets $\mathrm{VSE}(P)$ and $\mathrm{VSE}(Q)$ from Example 3.4, we obtain $\pi_2(\mathrm{VSE}(P)) = \{\{a\}, \{b\}, \{a, b\}\}$ and $\pi_2(\mathrm{VSE}(Q)) = \{\{a, c\}, \{b, d\}, \{a, b, c\}, \{a, b, d\}\}$. Thus, $\mathrm{VSE}(P) \overset{\mathrm{v}}{\neq} \mathrm{VSE}(Q)$, which reflects $P \not\equiv_{\mathrm{v}} Q$ and $P \not\equiv_{\mathrm{vs}} Q$. This is easily confirmed by the context $R = \{a.\ b.\ \}$: $\mathrm{SM}(P \cup R) = \{\{a, b\}\}$ but $\mathrm{SM}(Q \cup R) = \{\{a, b, c\}, \{a, b, d\}\}$. ■

The bijective relationship of models, which is an inherent property of visible equivalence in the first place, is expected of the second components of VSE-models. No such one-to-one restriction is imposed on the first components: The condition (6) simply checks the VSE-models associated with Y and $f(Y)$ and ensures that the visible parts of the first components are the same. Indeed, it is possible that a particular projection X_{v} is perceived from multiple VSE-models of P, i.e., $X_{\mathrm{v}} = X'_{\mathrm{v}}$ for $\langle X, Y\rangle \in \mathrm{VSE}(P)$ and $\langle X', Y\rangle \in \mathrm{VSE}(P)$ such that $X \neq X'$. See below for a concrete example.

Example 3.8. Let P be the program consisting of the following rules:

$$a \leftarrow c. \quad a \leftarrow d. \quad c \mid d. \quad b \leftarrow c, d. \quad c \leftarrow b. \quad d \leftarrow b.$$

Given $\mathrm{At}_{\mathrm{v}}(P) = \{a, b\}$, the program P has, e.g., VSE-models $M_1 = \langle\{a, c\}, \{a, b, c, d\}\rangle$ and $M_2 = \langle\{a, d\}, \{a, b, c, d\}\rangle$. Then, we observe that $\pi_1(M_1)_{\mathrm{v}} = \{a\} = \pi_1(M_2)_{\mathrm{v}}$. ■

The set S of interpretations $\pi_1(\sigma_Y(\mathrm{VSE}(P)))$ associated with an interpretation $Y \in \pi_2(\mathrm{VSE}(P))$ essentially forms a *certificate* $\langle S \setminus \{Y_{\mathrm{v}}\}, Y_{\mathrm{v}}\rangle$ in the sense of Reference [16] as detailed in Section 2.4. Conversely, the set S can be recovered from the first component of the certificate for Y by adding Y_{v} as a member.[4] Given the characterization of relativized strong equivalence in Theorem 2.16, our main contribution on the semantic side is the identification of the bijective relationship of certificate structures when it comes to establishing the visible strong equivalence of programs.

Example 3.9. Consider a logic program P consisting of the rules

$$a \leftarrow c. \quad a \leftarrow d. \quad c \mid d.$$

and the logic program Q whose only rule is a as a fact. Let $\mathrm{At}_{\mathrm{v}}(P) = \mathrm{At}_{\mathrm{v}}(Q) = \{a\}$. The VSE-models of P are $\langle\{a, c\}, \{a, c\}\rangle$ and $\langle\{a, d\}, \{a, d\}\rangle$, whereas $\langle\{a\}, \{a\}\rangle$ is the unique VSE-model of Q. These models induce a unique minimal certificate $\langle\emptyset, \{a\}\rangle$ for both programs. It follows by Theorem 2.16 that $\mathrm{SM}(P \cup R)_{\mathrm{v}} = \mathrm{SM}(Q \cup R)_{\mathrm{v}}$ for every context R such that $\mathrm{At}(R) \subset \{a\}$. But since $\mathrm{SM}(P) = \{\{a, c\}, \{a, d\}\}$ and $\mathrm{SM}(Q) = \{\{a\}\}$, it is clear that $P \not\equiv_{\mathrm{v}} Q$ and $P \not\equiv_{\mathrm{vs}} Q$ by the empty context $R = \emptyset$. ■

The *module theorem* of References [26, 36] shows how the stable models of a program P can be obtained from those of its component programs also known as *modules*. The key idea is that mutually *compatible* stable models of the modules can be joined together to form a stable model of P. As regards stable models, there is an important restriction to this result: Arbitrary unions

[4] The difference is a matter of bookkeeping, not truly affecting the notion of minimality in Theorem 2.16.

of component programs cannot be supported. The main principle is that *strongly connected components* (SCCs) with respect to positively interdependent atoms cannot be split between modules. However, such settings cannot arise when considering VSE-models under the assumption that component programs mutually respect the hidden atoms of each other. Thus any SCCs subject to stable semantics will stay distinct from each other. Let us also emphasize that the visible parts of VSE-models are interpreted classically, which effectively cuts any loops that would result across visible parts of programs.

Definition 3.10. Let P_1 and P_2 be two programs that mutually respect the hidden atoms of each other. Then, the respective VSE-interpretations $\langle X_1, Y_1\rangle$ and $\langle X_2, Y_2\rangle$ of P_1 and P_2 are *compatible* if and only if

(1) $X_1 \cap \mathrm{At_v}(P_2) = X_2 \cap \mathrm{At_v}(P_1)$ and
(2) $Y_1 \cap \mathrm{At_v}(P_2) = Y_2 \cap \mathrm{At_v}(P_1)$.

Theorem 3.11. *If programs P_1 and P_2 mutually respect the hidden atoms of each other, then* $\mathrm{VSE}(P_1 \cup P_2) = \mathrm{VSE}(P_1) \bowtie \mathrm{VSE}(P_2)$ *where the* join $\mathrm{VSE}(P_1) \bowtie \mathrm{VSE}(P_2)$ *is defined as the set of VSE-interpretations $\langle X_1 \cup X_2, Y_1 \cup Y_2\rangle$ for each pair of VSE-interpretations $\langle X_1, Y_1\rangle \in \mathrm{VSE}(P_1)$ and $\langle X_2, Y_2\rangle \in \mathrm{VSE}(P_2)$ that are compatible.*

Proof. Let $\langle X_1, Y_1\rangle$ such that $X_1 \subseteq Y_1 \subseteq \mathrm{At}(P_1)$ and $\langle X_2, Y_2\rangle$ such that $X_2 \subseteq Y_2 \subseteq \mathrm{At}(P_2)$ be compatible. As a consequence, the interpretations X_1, X_2, Y_1, and Y_2 can be extracted from $X_1 \cup X_2$ and $Y_1 \cup Y_2$ as simple projections over $\mathrm{At}(P_1)$ and $\mathrm{At}(P_2)$. Since $Y_1 \cup Y_2$ is a well-defined interpretation for $P_1 \cup P_2$, we obtain

$$\begin{aligned} Y_1 \cup Y_2 \models (P_1 \cup P_2)^{Y_1 \cup Y_2} &\iff Y_1 \cup Y_2 \models P_1 \cup P_2 && \text{[Lemma 2.2]} \\ &\iff Y_1 \models P_1 \text{ and } Y_2 \models P_2 && \text{[Modularity of } \models\text{]} \\ &\iff Y_1 \models P_1^{Y_1} \text{ and } Y_2 \models P_2^{Y_2}. && \text{[Lemma 2.2]} \end{aligned}$$

In analogy, since $X_1 \cup X_2$ is an interpretation for $P_1 \cup P_2$, we have

$$\begin{aligned} X_1 \cup X_2 \models (P_1 \cup P_2)^{Y_1 \cup Y_2} &\iff X_1 \cup X_2 \models P_1^{Y_1} \cup P_2^{Y_2} && \text{[Definition 2.1]} \\ &\iff X_1 \models P_1^{Y_1} \text{ and } X_2 \models P_2^{Y_2}. && \text{[Modularity of } \models\text{]} \end{aligned}$$

Moreover, since P_1 and P_2 mutually respect each other's hidden atoms, we have $\mathrm{At_h}(P_1 \cup P_2) = \mathrm{At_h}(P_1) \cup \mathrm{At_h}(P_2)$ and $\mathrm{At_h}(P_1) \cap \mathrm{At_h}(P_2) = \emptyset$. It follows that $Y_1 \cup Y_2$ is a $\mathrm{At_h}(P_1 \cup P_2)$-minimal model of $(P_1 \cup P_2)^{Y_1 \cup Y_2} = P_1^{Y_1} \cup P_2^{Y_2}$ iff Y_1 is a $\mathrm{At_h}(P_1)$-minimal model of $P_1^{Y_1}$ and Y_2 is a $\mathrm{At_h}(P_2)$-minimal model of $P_2^{Y_2}$. In summary, we have shown that $\langle X_1 \cup X_2, Y_1 \cup Y_2\rangle \in \mathrm{VSE}(P_1 \cup P_2)$ if and only if $\langle X_1, Y_1\rangle \in \mathrm{VSE}(P_1)$ and $\langle X_2, Y_2\rangle \in \mathrm{VSE}(P_2)$. □

Example 3.12. Let us consider the logic programs

$$P\colon\quad a \leftarrow c. \quad c \leftarrow b. \qquad\qquad Q\colon\quad b \leftarrow d. \quad d \leftarrow a.$$

where $\mathrm{At_v}(P) = \{a, b\} = \mathrm{At_v}(Q)$. Despite the positive loop in $P \cup Q$, we may safely apply Theorem 3.11 to compute $\mathrm{VSE}(P \cup Q)$. Both programs have six VSE-models:

$$\begin{aligned}
\mathrm{VSE}(P): \quad & \langle \emptyset, \emptyset \rangle \\
& \langle \emptyset, \{a\} \rangle \\
& \langle \{b,c\}, \{b,c\} \rangle \\
& \langle \emptyset, \{a,b,c\} \rangle \\
& \langle \{b,c\}, \{a,b,c\} \rangle \\
& \langle \{a,b,c\}, \{a,b,c\} \rangle
\end{aligned}
\qquad
\begin{aligned}
\mathrm{VSE}(Q): \quad & \langle \emptyset, \emptyset \rangle \\
& \langle \emptyset, \{b\} \rangle \\
& \langle \{a,d\}, \{a,d\} \rangle \\
& \langle \emptyset, \{a,b,d\} \rangle \\
& \langle \{a,d\}, \{a,b,d\} \rangle \\
& \langle \{a,b,d\}, \{a,b,d\} \rangle
\end{aligned}$$

Of these VSE-models, three pairs are compatible. First, since $\langle \emptyset, \emptyset \rangle$ is a member of both $\mathrm{VSE}(P)$ and $\mathrm{VSE}(Q)$, it is also a member of $\mathrm{VSE}(P \cup Q)$. Second, the respective members $\langle \emptyset, \{a,b,c\} \rangle$ and $\langle \emptyset, \{a,b,d\} \rangle$ of $\mathrm{VSE}(P)$ and $\mathrm{VSE}(Q)$ are compatible and contribute $\langle \emptyset, \{a,b,c,d\} \rangle$ to $\mathrm{VSE}(P \cup Q)$. In analogy, $\langle \{a,b,c\}, \{a,b,c\} \rangle$ and $\langle \{a,b,d\}, \{a,b,d\} \rangle$ give rise to $\langle \{a,b,c,d\}, \{a,b,c,d\} \rangle$. For contrast, an example of an incompatible pair is given by the VSE-models $\langle \{b,c\}, \{a,b,c\} \rangle$ and $\langle \{a,b,d\}, \{a,b,d\} \rangle$, which differ in the visible projections $\{b\}$ and $\{a,b\}$ of their X components, although they match in their Y components. Due to the difference, the pair does not contribute a VSE-model to $\mathrm{VSE}(P \cup Q)$. ■

3.1 Relationship with Classical Equivalence

Given a program P and an SE-model $\langle X, Y \rangle$ of P, the second component Y is a classical model of P. By Proposition 3.5, the same holds for VSE-models, but more importantly, Y is a hidden minimal model of P^Y. In what follows, we take such models as representatives for classical equivalence in settings where some atoms are hidden and define

$$\mathrm{VCE}(P) = \{Y \subseteq \mathrm{At}(P) \mid Y \in \mathrm{MM}_\mathrm{h}(P^Y)\}. \tag{7}$$

As hinted above, the visible part Y_v of $Y \in \mathrm{VCE}(P)$ serves the purpose of *satisfying in the classical sense* rules that define visible atoms in P. However, the hidden part Y_h is essentially a *stable model* for rules that define hidden atoms in P. By recognizing the importance of such models for visible strong equivalence, we arrive at the following generalization of (ordinary) classical equivalence as well as a number of auxiliary results that facilitate the study of visible strong equivalence in the sequel.

Definition 3.13 (Visible Classical Equivalence). Programs P and Q are *visibly classically equivalent*, denoted by $P \equiv_\mathrm{vc} Q$, iff $\mathrm{At}_\mathrm{v}(P) = \mathrm{At}_\mathrm{v}(Q)$ and $\mathrm{VCE}(P) =_\mathrm{v} \mathrm{VCE}(Q)$.

Definition 3.14. Given a program P, define a context program $\mathrm{Ctx}(P)$ that contains for every visible atom $a \in \mathrm{At}_\mathrm{v}(P)$, the respective choice rule $\{a\}$.

Theorem 3.15. *For a program P, the context program $\mathrm{Ctx}(P)$, and an interpretation $M \subseteq \mathrm{At}(P)$, $M \in \mathrm{VCE}(P)$ iff $M \in \mathrm{SM}(P \cup \mathrm{Ctx}(P))$.*

Proof. For the program P and any interpretation $M \subseteq \mathrm{At}(P)$,

$$\begin{aligned}
M \in \mathrm{VCE}(P) \quad & \iff \quad M \in \mathrm{MM}_\mathrm{h}(P^M) \text{ and } M \models \{a \leftarrow \mid a \in M\} \\
& \iff \quad M \in \mathrm{MM}(P^M \cup \{a \leftarrow \mid a \in M\}) \\
& \iff \quad M \in \mathrm{MM}((P \cup \mathrm{Ctx}(P))^M) \\
& \iff \quad M \in \mathrm{SM}(P \cup \mathrm{Ctx}(P)),
\end{aligned}$$

where $\mathrm{Ctx}(P)^M = \{a \leftarrow \mid a \in M\}$ holds in general. □

Corollary 3.16. *For programs P and Q with $\mathrm{At}_\mathrm{v}(P) = \mathrm{At}_\mathrm{v}(Q)$,*

(1) $P \equiv_\mathrm{vc} Q$ if and only if $P \cup \mathrm{Ctx}(P) \equiv_\mathrm{v} Q \cup \mathrm{Ctx}(Q)$, and
(2) $P \equiv_\mathrm{vs} Q$ implies $P \equiv_\mathrm{vc} Q$.

It is clear by the definition of VSE-models that $Y \in \mathrm{VCE}(P)$ iff $\langle Y, Y\rangle \in \mathrm{VSE}(P)$. Thus, we obtain the following result as a restriction of Theorem 3.11.

Corollary 3.17. *If P_1 and P_2 are logic programs that mutually respect the hidden atoms of each other, then* $\mathrm{VCE}(P_1 \cup P_2) = \mathrm{VCE}(P_1) \bowtie \mathrm{VCE}(P_2)$.

Proposition 3.18. *For a program P and a hidden minimal model $Y \in \mathrm{VCE}(P)$, $Y \in \mathrm{SM}(P)$ if and only if $\langle \emptyset, Y_\mathrm{v}\rangle$ is the certificate induced by Y.*

Proof. Let $Y \in \mathrm{VCE}(P)$. ($\Longrightarrow$) Suppose that $Y \in \mathrm{SM}(P)$ and that Y induces a certificate $\langle S, Y_\mathrm{v}\rangle$ such that $S \neq \emptyset$. Then, there is $\langle X, Y\rangle \in \mathrm{VSE}(P)$ such that $X_\mathrm{v} \in S$ and $X_\mathrm{v} \subset Y_\mathrm{v}$. Thus $Y \models P^Y$ and X contradicts the $\subseteq$-minimality of Y as $Y \in \mathrm{SM}(P)$.

($\Longleftarrow$) Suppose that $\langle \emptyset, Y_\mathrm{v}\rangle$ is the certificate induced by Y and $Y \notin \mathrm{SM}(P)$, i.e., $X \models P^Y$ for some $X \subset Y$. Then, there is a hidden minimal model $X' \subseteq X$ such that $X' \models P^Y$, $X'_\mathrm{v} = X_\mathrm{v}$, and $X'_\mathrm{v} \subset Y_\mathrm{v}$ by Proposition 3.5. It follows that $\langle X', Y\rangle \in \mathrm{VSE}(P)$ and $X'_\mathrm{v} = X_\mathrm{v}$ should be a member of the certificate induced by Y, a contradiction. □

3.2 Relationship with Ordinary Visible Equivalence

It is clear by the definition of visible strong equivalence that $P \equiv_\mathrm{vs} Q$ implies $P \equiv_\mathrm{v} Q$. The following lemma establishes the respective implication but starting from the model-theoretic correspondence of VSE-models laid out in Definition 3.6.

Lemma 3.19. *If* $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$ *for programs P and Q, then $P \equiv_\mathrm{v} Q$.*

Proof. Let us assume that $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$. Then, consider any $M \in \mathrm{SM}(P)$, i.e., $M \in \mathrm{MM}(P^M)$. Assuming that $M \notin \mathrm{MM}_\mathrm{h}(P^M)$, there is a model M' of P^M such that $M'_\mathrm{v} = M_\mathrm{v}$ and $M'_\mathrm{h} \subset M_\mathrm{h}$. But then $M' \subset M$, contradicting the overall minimality of M. Thus $M \in \mathrm{MM}_\mathrm{h}(P^M)$, $\langle M, M\rangle \in \mathrm{VSE}(P)$, and $M \in \pi_2(\mathrm{VSE}(P))$.

Using the bijection f inducing $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$, we obtain $N = f(M) \in \pi_2(\mathrm{VSE}(Q))$ with $N_\mathrm{v} = M_\mathrm{v}$. Since $N \in \mathrm{MM}_\mathrm{h}(Q^N)$, we obtain $\langle N, N\rangle \in \mathrm{VSE}(Q)$ and $N \models Q^N$. Let us then assume that $N' \models Q^N$ for some $\subseteq$-minimal model $N' \subset N$.

- If $N'_\mathrm{v} = N_\mathrm{v}$, then $N'_\mathrm{h} \subset N_\mathrm{h}$ and $N \notin \mathrm{MM}_\mathrm{h}(Q^N)$, a contradiction.
- Thus $N'_\mathrm{v} \subset N_\mathrm{v}$ and $N' \in \mathrm{MM}_\mathrm{h}(Q^N)$ as $N' \in \mathrm{MM}(Q^N)$. It follows that $\langle N', N\rangle \in \mathrm{VSE}(Q)$ and $N'_\mathrm{v} \in \pi_1(\sigma_N(\mathrm{VSE}(Q)))$. The inverse of f and (6) interpreted over M and $N = f(M)$ give us a corresponding pair $\langle M', M\rangle \in \mathrm{VSE}(P)$ such that $M'_\mathrm{v} = N'_\mathrm{v}$. Thus $M' \subset M$ and $M' \models P^M$ as $\langle M', M\rangle \in \mathrm{VSE}(P)$. A contradiction, since $M \in \mathrm{SM}(P)$.

Thus, we have shown that f maps a stable model $M \in \mathrm{SM}(P)$ to a distinguished stable model $N = f(M) \in \mathrm{SM}(Q)$ such that $N_\mathrm{v} = M_\mathrm{v}$. The converse holds by symmetry making the restriction of f from $\mathrm{SM}(P)$ to $\mathrm{SM}(Q)$ a bijection. Thus $P \equiv_\mathrm{v} Q$. □

3.3 Characterization of Visible Strong Equivalence: Soundness

Our next goal is to characterize $\equiv_\mathrm{vs}$ in terms of VSE-models and in analogy to a number of similar results (Reference [44, Theorem 1], Reference [31, Theorem 1], and Reference [14, Theorem 4.5]). It is natural that in our case, the bijective relationship of models plays its particular role in the resulting characterization of $\equiv_\mathrm{vs}$.

Lemma 3.20. *Let P and Q be programs such that* $\mathrm{At}_\mathrm{v}(P) = \mathrm{At}_\mathrm{v}(Q)$*, and R a context program such that P and R as well as Q and R mutually respect the hidden atoms of each other. Then,* $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$ *implies* $\mathrm{VSE}(P \cup R) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q \cup R)$.

Proof. Let $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$ hold via a bijection f from $\pi_2(\mathrm{VSE}(P)) = \mathrm{VCE}(P)$ to $\pi_2(\mathrm{VSE}(Q)) = \mathrm{VCE}(Q)$ and let R be any context program as described above. It follows that $\mathrm{VSE}(P \cup R) = \mathrm{VSE}(P) \bowtie \mathrm{VSE}(R)$ and $\mathrm{VSE}(Q \cup R) = \mathrm{VSE}(Q) \bowtie \mathrm{VSE}(R)$ by Theorem 3.11. Then, consider any $Y \in \pi_2(\mathrm{VSE}(P \cup R))$ for which there is $\langle X, Y\rangle \in \mathrm{VSE}(P \cup R)$. It follows that $\langle X_1, Y_1\rangle \in \mathrm{VSE}(P)$ and $\langle X_2, Y_2\rangle \in \mathrm{VSE}(R)$, where $X_1 = X \cap \mathrm{At}(P)$, $Y_1 = Y \cap \mathrm{At}(P)$, $X_2 = X \cap \mathrm{At}(R)$, and $Y_2 = Y \cap \mathrm{At}(R)$.

(1) Since $Y_1 \in \pi_2(\mathrm{VSE}(P))$, we can map it to $Y_1' = f(Y_1) \in \pi_2(\mathrm{VSE}(Q))$ such that $(Y_1')_{\mathrm{v}} = (Y_1)_{\mathrm{v}}$. Thus, Y_1' is compatible with Y_2, because Y_1 is compatible with Y_2 and R mutually respects the hidden atoms of P and Q.
(2) From $X_1 \in \pi_1(\sigma_{Y_1}(\mathrm{VSE}(P)))$ and Equation (6), we obtain $X_1' \subseteq Y_1'$ such that $\langle X_1', Y_1'\rangle \in \mathrm{VSE}(Q)$ and $(X_1')_{\mathrm{v}} = (X_1)_{\mathrm{v}}$. This makes X_1' compatible with X_2 as X_1 is compatible with X_2 and R mutually respects the hidden atoms of P and Q.

It follows that $\langle X', Y'\rangle \in \mathrm{VSE}(Q \cup R)$ holds for $X' = X_1' \cup X_2$ and $Y' = Y_1' \cup Y_2$. Thus, $Y' \in \pi_2(\mathrm{VSE}(Q \cup R))$ and we have obtained a function $f'(Y) = Y' = Y_1' \cup Y_2 = f(Y \cap \mathrm{At}(P)) \cup (Y \cap \mathrm{At}(R))$, which is a bijection as f is. To establish Equation (6) in the presence of R, we have to consider any $X_{\mathrm{v}} \subseteq Y_{\mathrm{v}}$ with $\langle X, Y\rangle \in \mathrm{VSE}(P \cup R)$, i.e., $X_{\mathrm{v}} \in \pi_1(\sigma_Y(\mathrm{VSE}(P \cup R)))$. Using the same line of reasoning as above, we obtain $\langle X', Y'\rangle \in \mathrm{VSE}(Q \cup R)$. A simple calculation yields $X'_{\mathrm{v}} = (X_1')_{\mathrm{v}} \cup (X_2)_{\mathrm{v}} = (X_1)_{\mathrm{v}} \cup (X_2)_{\mathrm{v}} = X_{\mathrm{v}}$, i.e., $X'_{\mathrm{v}} \in \pi_1(\sigma_{Y'}(\mathrm{VSE}(Q \cup R)))$ where $Y' = f(Y)$. This shows one half of Equation (6) and the other follows by symmetry using the inverse of f'. Thus $\mathrm{VSE}(P \cup R) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q \cup R)$ via f'. □

Theorem 3.21. *For programs P and Q with $\mathrm{At}_{\mathrm{v}}(P) = \mathrm{At}_{\mathrm{v}}(Q)$, $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$ implies $P \equiv_{\mathrm{vs}} Q$.*

Proof. Let P and Q be programs with $\mathrm{At}_{\mathrm{v}}(P) = \mathrm{At}_{\mathrm{v}}(Q)$ and $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$. Now, if R is any context such that P and R as well as Q and R mutually respect the hidden atoms of each other, we obtain $\mathrm{VSE}(P \cup R) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q \cup R)$ by Lemma 3.20. It follows by Lemma 3.19 that $\mathrm{SM}(P \cup R) =_{\mathrm{v}} \mathrm{SM}(Q \cup R)$. Since R was an arbitrary context program respecting the hidden atoms of P and Q, we obtain $P \equiv_{\mathrm{vs}} Q$ as desired. □

3.4 Characterization of Visible Strong Equivalence: Completeness

Next, we would like to establish in general that if $\mathrm{VSE}(P)$ and $\mathrm{VSE}(Q)$ do not visibly match, this boils down to P and Q not being visibly strongly equivalent. To understand the difference with respect to Theorem 2.16, let us illustrate how even non-minimal certificates may affect visible strong equivalence. Basically, we use the programs P and Q from Example 2.18 but extend P by one rule to make the programs visibly *classically* equivalent subject to $\mathrm{At}_{\mathrm{v}}(P) = \mathrm{At}_{\mathrm{v}}(Q) = \{a, b\}$.

Example 3.22. So, let us analyze the following programs in more detail:

$$P\colon\quad a \leftarrow {\sim}a, {\sim}b. \qquad b \leftarrow {\sim}a, {\sim}b. \qquad a \leftarrow b. \qquad b \leftarrow a. \qquad c \mid d.$$

$$Q\colon\quad a \leftarrow {\sim}a, {\sim}b. \qquad b \leftarrow {\sim}a, {\sim}b. \qquad a \leftarrow b. \qquad b \leftarrow a, {\sim}c. \qquad a \leftarrow c. \qquad b \leftarrow c. \qquad a \leftarrow d. \qquad b \leftarrow d. \qquad c \mid d \leftarrow a, b.$$

It is easy to satisfy all rules concerning a and b by setting these atoms true and it remains to satisfy the hidden disjunction $c \mid d$, present in both P and Q when a and b are true, in a minimal way. It follows that $Y^1 = \{a, b, c\}$ and $Y^2 = \{a, b, d\}$ are the hidden minimal models of both programs. Thus, we have $P \equiv_{\mathrm{vc}} Q$. However, an exact visible match between $\mathrm{VSE}(P)$ and $\mathrm{VSE}(Q)$ is pre-empted by the certificate $\langle\{\emptyset, \{a\}\}, \{a, b\}\rangle$ associated with Q and Y^1, since the respective certificate for P

and Y^1 is $\langle\{\emptyset\},\{a,b\}\rangle$. But it is not obvious how to capture such differences in terms of context programs. ■

Therefore, for showing the converse of Theorem 3.21, we need a sophisticated context program that is specific to a given hidden minimal model $Y \in \mathrm{VCE}(P)$. The overall goal of the context program is to capture the *complement* of the set $\pi_1(\sigma_Y(\mathrm{VSE}(P)))_\mathrm{v}$ from Equation (6) by detecting all interpretations $Z^j_\mathrm{v} \subset Y_\mathrm{v}$ for which there is no $\langle Z, Y\rangle \in \mathrm{VSE}(P)$ such that $Z_\mathrm{v} = Z^j_\mathrm{v}$. The context program given below encodes all such subsets. Note that since $\langle Y, Y\rangle \in \mathrm{VSE}(P)$ holds for Y, the set Y_v trivially lacks the property of interest and hence it is excluded from consideration as one of the sets Z^j_v.

Definition 3.23 (Context Program). Let P be a logic program and $Y \subseteq \mathrm{At}(P)$ an interpretation. Given any collection $Z^1_\mathrm{v}, \ldots, Z^n_\mathrm{v}$ of proper subsets of the visible part Y_v so that $n < 2^{|Y_\mathrm{v}|}$, the context program R denoted by $\mathrm{Ctx}(Y_\mathrm{v}; Z^1_\mathrm{v}, \ldots, Z^n_\mathrm{v})$ consists of:

$$z \mid u_j. \qquad \left(1 \leq j \leq n \text{ and } z \in Z^j_\mathrm{v}\right), \tag{8}$$

$$z_1 \mid u_j \leftarrow z_2. \qquad \left(1 \leq j \leq n, z_1 \in Y_\mathrm{v} \setminus Z^j_\mathrm{v}, \text{and } z_2 \in Y_\mathrm{v} \setminus Z^j_\mathrm{v}\right), \tag{9}$$

$$u \leftarrow u_1, \ldots, u_n. \tag{10}$$

$$u_j \leftarrow u. \qquad (1 \leq j \leq n), \tag{11}$$

$$y \leftarrow u. \qquad (y \in Y_\mathrm{v}), \tag{12}$$

$$u \leftarrow Y_\mathrm{v}. \qquad (\text{where } Y_\mathrm{v} \text{ abbreviates a conjunction}), \tag{13}$$

$$u \leftarrow {\sim}u. \tag{14}$$

The hidden signature $\mathrm{At}_\mathrm{h}(R)$ is defined to be $\emptyset$ so that $\mathrm{At}_\mathrm{v}(R) = \mathrm{At}(R) = Y_\mathrm{v} \cup U$, where $U = \{u, u_1, \ldots, u_n\}$ is the set of new (visible) atoms used above.

Since $Y_\mathrm{v} \subseteq \mathrm{At}_\mathrm{v}(P)$, it is clear that the program above respects the hidden atoms of P. The rules encode a parallel test for the sets $Z^1_\mathrm{v}, \ldots, Z^n_\mathrm{v}$ using a technique known as *saturation* from Reference [15]. The test for a particular subset Z^j_v where $1 \leq j \leq n$ is encoded by the rules (8) and (9). Assigning u_j to false activates a subprogram analogous to $R(X_\mathrm{v}, Y_\mathrm{v})$ in the end of Section 2.3 (X_v is being replaced by Z^j_v). The atom u_j is derived upon a successful test (Z^j_v is not extendible to a model of P^Y) and eventually u is derived by rule (10) to indicate the success of all tests as well as overall inconsistency. The derivation of u implies the derivation of all other atoms using rules (11) and (12). This is necessary to stabilize $Y \cup U$. The tests concern only proper subsets of Y_v that is implemented by the rule (13). The final rule (14) ensures that only stable models containing u are accepted. The context program works as follows.

Lemma 3.24. *Let P be a program, $Y \subseteq \mathrm{At}(P)$ an interpretation, and $R = \mathrm{Ctx}(Y_\mathrm{v}; Z^1_\mathrm{v}, \ldots, Z^n_\mathrm{v})$ a context program from Definition 3.23 based on a collection $Z^1_\mathrm{v}, \ldots, Z^n_\mathrm{v}$ of proper subsets of Y_v. Then, $Y \cup U \in \mathrm{SM}(P \cup R)$ iff $Y \in \mathrm{VCE}(P)$ and no Z^j_v is extendible to a model $Z \subset Y$ of P^Y such that $Z_\mathrm{v} = Z^j_\mathrm{v}$.*

Proof. ($\Longrightarrow$): Assume that $Y \cup U \in \mathrm{SM}(P \cup R)$, i.e., $Y \cup U$ is a minimal model of $(P \cup R)^{Y \cup U}$. The reduct equals to $P^Y \cup R$, since R is a positive disjunctive program respecting the hidden atoms of P, and $\mathrm{At}(P) \cap U = \emptyset$. It follows that $Y \models P^Y$. Let us then assume that Y is not a $\mathrm{At}_\mathrm{h}(P)$-minimal model of P^Y, i.e., there is $Y' \models P^Y$ such that $Y' \subset Y$ and $Y'_\mathrm{v} = Y_\mathrm{v}$. But then also $Y' \cup U$ is a model of $P^Y \cup R$, which contradicts the $\subseteq$-minimality of $Y \cup U$. Thus Y is a $\mathrm{At}_\mathrm{h}(P)$-minimal model of P^Y and $Y \in \mathrm{VCE}(P)$.

Then, suppose that some Z^j_v with $1 \leq j \leq n$ can be extended to a model $Z \subset Y$ of P^Y such that $Z_\mathrm{v} = Z^j_\mathrm{v} \subset Y_\mathrm{v}$. Let $Z' = Z \cup \{u_k \mid 1 \leq k \leq n \text{ and } k \neq j\}$ for which $Z' \not\models u_j$ and $Z' \models P^Y$ holds as

$Z \models P^Y$. The structure of R implies $Z' \models R$ as $Z'_{\mathrm{v}} = Z^j_{\mathrm{v}}$. In particular, the respective instances of rules (8) and (9) involving u_j (that is falsified by Z') are satisfied, since $Z^j_{\mathrm{v}} \subset Y_{\mathrm{v}}$ satisfies them. Since $Z' \subset Y \cup U$, this contradicts the $\subseteq$-minimality of $Y \cup U \models P^Y \cup R$. Hence, no Z^j_{v} is extendible in the intended way.

($\Longleftarrow$) Suppose that $Y \in \mathrm{VCE}(P)$, i.e., $Y \in \mathrm{MM_h}(P^Y)$, and no Z^j_{v} is extendible to a model $Z \models P^Y$ such that $Z \subset Y$ and $Z_{\mathrm{v}} = Z^j_{\mathrm{v}}$. Define $Y' = Y \cup U$. It is clear that $Y' \models P^Y$, since $\mathrm{At}(P) \cap U = \emptyset$ and $Y \models P^Y$. It is also easy to inspect $Y' \models R$ given the definition of Y' and the rules of R listed above. It follows by Lemma 2.2 that $Y' \models (P \cup R)^{Y'}$ where the reduct coincides with $P^Y \cup R$. Let us then assume $Z' \models (P \cup R)^{Y'}$ for some $Z' \subseteq Y'$. It follows that $Z' \models P^Y$ and $Z' \models R$. Now, assuming $u \notin Z'$ implies $u_j \notin Z'$ for some $1 \leq j \leq n$ as $Z' \models u \leftarrow u_1, \ldots, u_n$. Since $Z' \models z \mid u_j$ for each $z \in Z^j_{\mathrm{v}}$ and $Z' \not\models u_j$, we obtain $Z^j_{\mathrm{v}} \subseteq Z'$. Moreover, $Z' \models z_1 \leftarrow z_2$ for each $z_1, z_2 \in Y_{\mathrm{v}} \setminus Z^j_{\mathrm{v}}$, because $Z' \models z_1 \mid u_j \leftarrow z_2$ from R and $Z' \not\models u_j$. As $Z' \subseteq Y'$, it follows that $Z'_{\mathrm{v}} = Z^j_{\mathrm{v}}$ or $Z'_{\mathrm{v}} = Y_{\mathrm{v}}$. The former is impossible, as Z^j_{v} is not extendible to a model $Z \subset Y$ of P^Y such that $Z_{\mathrm{v}} = Z^j_{\mathrm{v}}$. But then $u \in Z'$ follows as $Z' \models u \leftarrow Y_{\mathrm{v}}$, a contradiction. Hence $u \in Z'$ and $Y_{\mathrm{v}} \subseteq Z'$ as $Z' \models y \leftarrow u$ for each $y \in Y_{\mathrm{v}}$. Moreover, we have $U \subseteq Z'$, because $Z' \models u_j \leftarrow u$ for each $1 \leq j \leq n$. It follows that $Z' = Y'$ as $Z' \models P^Y$ and Y is a $\mathrm{At_h}(P)$-minimal model of P^Y. Thus, we have established that $Y' = Y \cup U \in \mathrm{SM}(P \cup R)$. □

We are ready to form a context program that is able to detect the difference of the programs P and Q from Example 3.22.

Example 3.25. The certificate $\langle\{\emptyset\}, \{a, b\}\rangle$ yields subsets $Z^1_{\mathrm{v}} = \{a\}$ and $Z^2_{\mathrm{v}} = \{b\}$ for the respective tester program. Given $Y_{\mathrm{v}} = \{a, b\}$, we obtain the context program $R = \mathrm{Ctx}(Y_{\mathrm{v}}; Z^1_{\mathrm{v}}, Z^2_{\mathrm{v}})$ listed below:

$$
\begin{array}{llll}
a \mid u_1. & b \mid u_1 \leftarrow b. & b \mid u_2. & a \mid u_2 \leftarrow a. \\
u \leftarrow u_1, u_2. & u_1 \leftarrow u. & u_2 \leftarrow u. & \\
a \leftarrow u. & b \leftarrow u. & u \leftarrow a, b. & \\
u \leftarrow {\sim}u. & & &
\end{array}
$$

Then, let $U = \{u, u_1, u_2\}$, $Y^1 = \{a, b, c\}$, and $Y^2 = \{a, b, d\}$ so that $Y^1_{\mathrm{v}} = \{a, b\} = Y^2_{\mathrm{v}}$. It follows that $Y^1 \cup U \in \mathrm{SM}(P \cup R)$ and $Y^1 \cup U \notin \mathrm{SM}(Q \cup R)$, while $Y^2 \cup U \in \mathrm{SM}(P \cup R)$ and $Y^2 \cup U \in \mathrm{SM}(Q \cup R)$. Thus R is a sufficient context program to establish $P \not\equiv_{\mathrm{vs}} Q$. ■

Lemma 3.26. *Let P and Q be logic programs such that $P \equiv_{\mathrm{vs}} Q$ holds. Moreover, let $Y^1, \ldots, Y^k$ be distinct members of* $\mathrm{VCE}(P)$ *such that*

(1) the visible parts $Y^1_{\mathrm{v}} = \cdots = Y^k_{\mathrm{v}}$ coincide with some $Y_{\mathrm{v}} \subseteq \mathrm{At_v}(P) = \mathrm{At}(Q)$,
(2) $S \subseteq 2^{Y_{\mathrm{v}}}$ is an upper bound for certificates, and
(3) each Y^i with $1 \leq i \leq k$ induces a certificate $\langle S_i, Y_{\mathrm{v}}\rangle$ with $S_i \subseteq S$ for P.

Then, there are at least k distinct members $V^1, \ldots, V^k$ of $\mathrm{VCE}(Q)$ *such that $V^1_{\mathrm{v}} = \cdots = V^k_{\mathrm{v}} = Y_{\mathrm{v}}$ and for each $1 \leq i \leq k$, the certificate of V^i for Q is $\langle T_i, Y_{\mathrm{v}}\rangle$ where $T_i \subseteq S$.*

Proof. Let $Z^1_{\mathrm{v}}, \ldots, Z^n_{\mathrm{v}}$ be the *proper* subsets of Y_{v} not contained in S and R the context program $\mathrm{Ctx}(Y_{\mathrm{v}}; Z^1_{\mathrm{v}}, \ldots, Z^n_{\mathrm{v}})$ from Definition 3.23. Then, consider any of Y^i with $1 \leq i \leq k$ and the certificate $\langle S_i, Y_{\mathrm{v}}\rangle$ for P. The certificate with $S_i \subseteq S$ implies that no Z^j_{v} is extendible to a model $Z \subset Y^i$ of P^{Y^i} such that $Z_{\mathrm{v}} = Z^j_{\mathrm{v}}$. It follows by Lemma 3.24 that $Y^i \cup U \in \mathrm{SM}(P \cup R)$ for each $1 \leq i \leq k$ and the set U of new atoms involved in the context program R. Since $P \equiv_{\mathrm{vs}} Q$, the bijection f_R from $\mathrm{SM}(P \cup R)$ to $\mathrm{SM}(Q \cup R)$ conveys us k stable models $V^i \cup U \in \mathrm{SM}(Q \cup R)$ with $1 \leq i \leq k$ and $V^i_{\mathrm{v}} = Y^i_{\mathrm{v}} = Y_{\mathrm{v}}$. It follows by Lemma 3.24 that no Z^j_{v} is extendible to a model $Z \subset V^i$ of Q^{Y^i} such that $Z_{\mathrm{v}} = Z^j_{\mathrm{v}}$. Thus, the certificate $\langle T_i, V^i_{\mathrm{v}}\rangle$ for Q induced by V^i satisfies $V^i_{\mathrm{v}} = Y_{\mathrm{v}}$ and $T_i \subseteq S$. □

THEOREM 3.27. *For programs P and Q with $\mathrm{At_v}(P) = \mathrm{At_v}(Q)$, $P \equiv_{\mathrm{vs}} Q$ implies $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$.*

PROOF. Let $P \equiv_{\mathrm{vs}} Q$ hold, i.e., $\mathrm{SM}(P \cup R) =_{\mathrm{v}} \mathrm{SM}(Q \cup R)$ is induced by a bijection, say, f_R, for any context R that is mutually compatible with P and Q. The proof of $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$ takes place in two steps in accordance with Definition 3.6.

(1) Using the context program $\mathrm{Ctx}(P) = \mathrm{Ctx}(Q)$ from Definition 3.14, we obtain $P \cup \mathrm{Ctx}(P) \equiv_{\mathrm{v}} Q \cup \mathrm{Ctx}(Q)$ via a bijection f. Thus we know that $\mathrm{VCE}(P) =_{\mathrm{v}} \mathrm{VCE}(Q)$ by Theorem 3.15. But then f is also a bijection from $\pi_2(\mathrm{VSE}(P))$ to $\pi_2(\mathrm{VSE}(Q))$, since $Y \in \pi_2(\mathrm{VSE}(P))$ iff $\langle Y, Y\rangle \in \mathrm{VSE}(P)$ iff $Y \in \mathrm{VCE}(P)$ for $Y \subseteq \mathrm{At}(P)$ in general, and analogously for Q. Thus the classical models of P and Q, as represented by the hidden minimal models in $\mathrm{VCE}(P)$ and $\mathrm{VCE}(Q)$, are in a one-to-one correspondence.

(3) In what follows, we consider a particular projection Y_{v} associated with some hidden minimal model $Y \in \mathrm{VCE}(P)$ and all distinct members $Y^1, \ldots, Y^k$ of $\mathrm{VCE}(P)$ such that $Y^i_{\mathrm{v}} = Y_{\mathrm{v}}$ for every $1 \leq i \leq k$. Thus Y is one of the models $Y^1, \ldots, Y^k$ under consideration. Then, let $\langle S^i, Y_{\mathrm{v}}\rangle$ be the certificate induced by each Y^i for $1 \leq i \leq k$. Moreover, we may assume without loss of generality that the sequence $Y^1, \ldots, Y^k$ is ordered so that $S_i \subseteq S_j$ implies $i < j$ or $S_j = S_{j-1} = \cdots = S_i$, i.e., certificates are ordered by $\subseteq$-inclusion and hidden minimal models giving rise to the same certificate populate subsequent index values in the order. In the rest of the proof, our strategy is to establish a bijective mapping to hidden minimal models $V^1, \ldots, V^k$ from $\mathrm{VCE}(Q)$ so that $V^i_{\mathrm{v}} = Y_{\mathrm{v}}$ for each $1 \leq i \leq k$ and the certificate associate with V^i coincides with $\langle S_i, Y_{\mathrm{v}}\rangle$. This is a key property to show Equation (6), since $\pi_1(\sigma_Y(\mathrm{VSE}(P)))_{\mathrm{v}} = S \cup \{Y_{\mathrm{v}}\}$ for the certificate $\langle S, Y_{\mathrm{v}}\rangle$ of $Y \in \mathrm{VCE}(P)$ in general. We use induction on $1 \leq i \leq k$ to map Y^i to V^i while preserving certificates $\langle S_i, Y_{\mathrm{v}}\rangle$.

In the **base case** $i = 1$, the certificate $\langle S_1, Y_{\mathrm{v}}\rangle$ induced by Y^1 is necessarily a minimal one. Let $m \geq 1$ be the maximal index value such that $S_1 = \cdots = S_m$, i.e., distinct members $Y^1, \ldots, Y^m$ of $\mathrm{VCE}(P)$ induce exactly the same certificate $\langle S_1, Y_{\mathrm{v}}\rangle$. It follows by Lemma 3.26 that there are distinct members $V^1, \ldots, V^m$ of $\mathrm{VCE}(Q)$ inducing certificates $\langle T_i, Y_{\mathrm{v}}\rangle$ for Q. Now, if $T_i \subset S_1$ were to hold, we may appeal to Lemma 3.26 in the case $k = 1$ and $S = T_i$ to obtain a member Y' of $\mathrm{VCE}(P)$ such that $Y'_{\mathrm{v}} = Y_{\mathrm{v}}$ and the certificate induced by Y' for P is $\langle T_i, Y_{\mathrm{v}}\rangle$. But this would be a contradiction, since Y' should occur before Y^1 in $Y^1, \ldots, Y^k$. Thus $T_i = S_1$ for each $1 \leq i \leq m$ and the certificate associated with V^i is $\langle S_1, Y_{\mathrm{v}}\rangle$ for $1 \leq i \leq m$. Moreover, if we would assume that there were more than m such members V^i of $\mathrm{VCE}(Q)$, we obtain a contradiction by Lemma 3.26, since m is a maximal index value, i.e., there are further members of $\mathrm{VCE}(P)$ with certificate $\langle S_1, Y_{\mathrm{v}}\rangle$. Thus Equation (6) holds.

For the **induction step**, we may assume without loss of generality that $\langle S_i, Y_{\mathrm{v}}\rangle$ is a non-minimal certificate associated with some $Y^i \in \mathrm{VCE}(P)$. Recalling the order imposed on $Y^1, \ldots, Y^k$, let $i \leq m \leq k$ be the maximal index value so that $S_i = \cdots = S_m$ and let $\langle S, Y_{\mathrm{v}}\rangle$ be the identical certificate in question. Since $\langle S, Y_{\mathrm{v}}\rangle$ is non-minimal, there are $1 \leq l < i$ members $Y^j \in \mathrm{VCE}(P)$ with $1 \leq j < i$ inducing a certificate $\langle S_j, Y_{\mathrm{v}}\rangle$ such that $S_j \subset S$ for P. Since $S_j \subset S$, it follows by the inductive hypothesis that each such member V^j of $\mathrm{VCE}(P)$ is bijectively mapped to to a member V^j of $\mathrm{VCE}(Q)$ such that $V^j_{\mathrm{v}} = Y^j_{\mathrm{v}} = Y_{\mathrm{v}}$, preserving the certificate $\langle S_j, Y_{\mathrm{v}}\rangle$. In total, there are now $l + m$ members Y^j of $\mathrm{VCE}(P)$ inducing a certificate $\langle S_j, Y_{\mathrm{v}}\rangle$ with $S_j \subseteq S$. It follows by Lemma 3.26 that there are at least $l + m$ members of V^j of $\mathrm{VCE}(Q)$ such that $V^j_{\mathrm{v}} = Y^j_{\mathrm{v}} = Y_{\mathrm{v}}$ and the certificate $\langle T_j, Y_{\mathrm{v}}\rangle$ induced by V^j satisfies $T_j \subseteq S$. The inductive hypothesis implies that the first l models V^j induce a certificate $\langle T_j, S\rangle$ with $T_j \subset S$ and $T_j = S_j$. Moreover, assuming $T_j \subset S$ for some $i \leq j \leq m$ contradicts the one-to-one correspondence between Y^j and V^j for $1 \leq j < i$. Thus $T_j = S$ for $i \leq j \leq m$

is necessarily the case and the respective models Y^j and V^j induce identical certificates $\langle S, Y_v \rangle$ for P and Q. Therefore, an arbitrary bijective mapping between $Y^i, \ldots, Y^m$ and $V^i, \ldots, V^m$ can be selected and, as a consequence, we have established the intended one-to-one correspondence between $Y^1, \ldots, Y^m$ and $V^1, \ldots, V^m$.

To conclude the proof, we have shown that VSE(P) and VSE(Q) visibly match. □

3.5 Relationship with Correspondence Frames

This section shows how to capture visible strong equivalence $\equiv_{vs}$ with correspondence frames [16] when the hidden signatures of the compared programs are fixed. This requirement to fix signatures stems from the design of correspondence frames $\mathcal{F} = \langle \mathcal{U}, \mathcal{C}, \rho \rangle$, which carry signature information in themselves as opposed to in programs as per the definitions in Section 2.4. Consequently, there is no single correspondence frame that captures $\equiv_{vs}$, but rather a sub-family of frames that together cover $\equiv_{vs}$.

In more detail, recalling that the frame $\langle \mathcal{U}, \mathcal{P}_A, =_B \rangle$ captures relativized strong equivalence with projection, the frame can be developed into another frame $\mathcal{F} = \langle \mathcal{U}, \mathcal{P}_A, =_v^B \rangle$ involving another comparison relation $=_v^B$ that insists on a one-to-one correspondence between the B-projections of stable models. This obtained frame is close to $\equiv_{vs}$, but requires some care in picking the parameters A and B. Since the parameter set A controls which atoms are available for context programs R, it must contain all atoms except those hidden in P and Q in order for A to capture the context programs relevant for $P \equiv_{vs} Q$. Moreover, since B controls which atoms are included in answer set comparisons, it must contain at least all visible atoms of P and Q and no hidden ones. Both of these requirements are met by the parameterization $A = B = \mathcal{U} \setminus H$ where $H = \mathrm{At}_h(P) \cup \mathrm{At}_h(Q)$ and $\mathcal{U}$ is assumed to contain all atoms that may occur in any program. The connection between the resulting frame and $\equiv_{vs}$ is given formally after the following lemma that is useful in proving the connection.

Lemma 3.28. *For programs P and Q, $P \equiv_{vs} Q$ holds if and only if $\mathrm{At}_v(P) = \mathrm{At}_v(Q)$ and $\mathrm{SM}(P \cup R) =_v \mathrm{SM}(Q \cup R)$ for any context R that mutually respects the hidden atoms of P and Q and has no hidden atoms.*

Proof. The "only if" direction holds trivially.

To establish the "if" direction, let us assume the right-hand side to hold and take any context R with possibly some hidden atoms and a similar context R' with the same rules but without any hidden atoms so that $\mathrm{At}_h(R') = \emptyset$. The right-hand side then implies $\mathrm{SM}(P \cup R') =_v \mathrm{SM}(Q \cup R')$. Hence, there is a bijection $f : \mathrm{SM}(P \cup R') \to \mathrm{SM}(Q \cup R')$ such that for every $M \in \mathrm{SM}(P \cup R')$, $M \cap V' = f(M) \cap V'$, where $V' = \mathrm{At}_v(P) \cup \mathrm{At}_v(R') = \mathrm{At}_v(Q) \cup \mathrm{At}_v(R')$ and $\mathrm{At}_v(R') = \mathrm{At}(R') = \mathrm{At}(R)$. Certainly, the same holds with any $V \subseteq V'$ in place of V' and in particular with $V = \mathrm{At}_v(P) \cup \mathrm{At}_v(R) = \mathrm{At}_v(Q) \cup \mathrm{At}_v(R)$. Thus, f also serves to prove $\mathrm{SM}(P \cup R) =_v \mathrm{SM}(Q \cup R)$. □

Proposition 3.29. *For programs P and Q with $\mathrm{At}_v(P) = \mathrm{At}_v(Q)$, let $\mathcal{F}$ be the correspondence frame $\mathcal{F} = \langle \mathcal{U}, \mathcal{P}_{\mathcal{U} \setminus H}, =_v^{\mathcal{U} \setminus H} \rangle$ where $\mathcal{U}$ is a universe of all atoms that may occur in P, Q, or any context program, and $H = \mathrm{At}_h(P) \cup \mathrm{At}_h(Q)$ is the set of hidden atoms of P and Q, Then, $P \equiv_{vs} Q$ holds if and only if P and Q are $\mathcal{F}$-corresponding.*

Proof. Let P, Q, $\mathcal{U}$, and H be defined as above. Then, let us recall that by Lemma 3.28, $P \equiv_{vs} Q$ holds if and only if $\mathrm{SM}(P \cup R) =_v \mathrm{SM}(Q \cup R)$ for any context program R that mutually respects the hidden atoms of P and Q and for which $\mathrm{At}_h(R) = \emptyset$. Observe that this statement continues to hold if the conditions for the context program R are replaced with any of the conditions:

- $\mathrm{At}(R) \cap (\mathrm{At_h}(P) \cup \mathrm{At_h}(Q)) = \emptyset$ and $\mathrm{At_h}(R) = \emptyset$,
- $\mathrm{At}(R) \cap H = \emptyset$ and $\mathrm{At_h}(R) = \emptyset$,
- $R \in \mathcal{P}_{\mathcal{U}\setminus H}$.

Furthermore, in the context of these programs R without hidden atoms, the visible equality $\mathrm{SM}(P \cup R) =_\mathrm{v} \mathrm{SM}(Q \cup R)$ coincides with $\mathrm{SM}(P \cup R) =_\mathrm{v}^{\mathcal{U}\setminus H} \mathrm{SM}(Q \cup R)$. In summary, $P \equiv_\mathrm{vs} Q$ if and only if $\mathrm{SM}(P \cup R) =_\mathrm{v}^{\mathcal{U}\setminus H} \mathrm{SM}(Q \cup R)$ for any $R \in \mathcal{P}_{\mathcal{U}\setminus H}$. Hence, $P \equiv_\mathrm{vs} Q$ amounts to checking $\langle \mathcal{U}, \mathcal{P}_{\mathcal{U}\setminus H}, =_\mathrm{v}^{\mathcal{U}\setminus H}\rangle$-correspondence. □

4 ISOLATING VISIBLE NEGATIVE CONDITIONS

This section gives results on evaluating visible strong equivalence for specific subclasses of programs. Namely, for positive programs, it is shown that visible strong equivalence can be proved or disproved by comparing the visible parts of hidden minimal models of the programs. Finding such proofs is an easier alternative to comparing VSE-models and then using Theorem 3.21 or 3.27. Moreover, for the more general class of programs with negative conditions on visible atoms, a sufficient precondition for visible strong equivalence is given in terms of hidden minimal models of simple translations of the compared programs. This more general result is particularly applicable to verifying rule-by-rule translations of programs, which are the topic of Section 5.

The first main result of this section shows that the visibility-aware semantics of positive programs is completely captured by their hidden minimal models. The result builds on the following lemma, which states that checking whether the VSE-models of two positive programs visibly match can be determined by checking whether their sets of hidden minimal models are visibly equal.

Lemma 4.1. *For positive programs P and Q with $\mathrm{At_v}(P) = \mathrm{At_v}(Q)$, it holds that $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$ if and only if $\mathrm{MM_h}(P) =_\mathrm{v} \mathrm{MM_h}(Q)$.*

Proof. Since P and Q are positive programs, both are unchanged when reduced with respect to any interpretation. Thus for positive programs P and Q, the first requirement for $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$ to hold via a bijection f simplifies into:

$$\begin{aligned}
&\pi_2(\mathrm{VSE}(P)) =_\mathrm{v} \pi_2(\mathrm{VSE}(Q)) && \text{via } f\\
\iff &\{Y \subseteq \mathrm{At}(P) \mid Y \in \mathrm{MM_h}(P^Y)\} =_\mathrm{v} \{Y \subseteq \mathrm{At}(Q) \mid Y \in \mathrm{MM_h}(Q^Y)\} && \text{via } f\\
\iff &\mathrm{MM_h}(P) =_\mathrm{v} \mathrm{MM_h}(Q) && \text{via } f.
\end{aligned}$$

Moreover, the second requirement for $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$ simplifies into a redundant requirement implied by $\mathrm{MM_h}(P)_\mathrm{v} = \mathrm{MM_h}(Q)_\mathrm{v}$, and therefore also by the above. More precisely, for every matching pair of interpretations $Y \in \pi_2(\mathrm{VSE}(P)) = \mathrm{MM_h}(P)$ and $f(Y) \in \pi_2(\mathrm{VSE}(Q)) = \mathrm{MM_h}(Q)$,

$$\begin{aligned}
&\pi_1(\sigma_Y(\mathrm{VSE}(P)))_\mathrm{v} = \pi_1(\sigma_{f(Y)}(\mathrm{VSE}(Q)))_\mathrm{v}\\
\iff &\{X_\mathrm{v} \mid X \subseteq Y, X \in \mathrm{MM_h}(P^Y)\} = \{X_\mathrm{v} \mid X \subseteq f(Y), X \in \mathrm{MM_h}(Q^Y)\}\\
\iff &(2^Y \cap \mathrm{MM_h}(P))_\mathrm{v} = (2^{f(Y)} \cap \mathrm{MM_h}(Q))_\mathrm{v}\\
\iff &2^{Y_\mathrm{v}} \cap \mathrm{MM_h}(P)_\mathrm{v} = 2^{f(Y)_\mathrm{v}} \cap \mathrm{MM_h}(Q)_\mathrm{v}.
\end{aligned}$$

In summary, the requirements behind $\overset{\mathrm{v}}{=}$ and $=_\mathrm{v}$ coincide for positive programs. □

As a corollary of Theorem 3.21 and Lemma 4.1, a higher-level result follows stating that the visible strong equivalence of two positive programs can be determined by checking whether their sets of hidden minimal models are visibly equal.

Corollary 4.2. *For positive programs P and Q with $\mathrm{At_v}(P) = \mathrm{At_v}(Q)$, it holds that $P \equiv_{\mathrm{vs}} Q$ if and only if $\mathrm{MM_h}(P) =_{\mathrm{v}} \mathrm{MM_h}(Q)$.*

The use of this result for proving or disproving visible strong equivalence requires strictly less work than by using Theorem 3.21 or 3.27 directly and Definition 3.6 in testing whether sets of VSE-models visibly match. This holds, because the condition $\mathrm{MM_h}(P) =_{\mathrm{v}} \mathrm{MM_h}(Q)$ coincides with the first prerequisite condition in Definition 3.6 when the condition is applied to positive programs.

As Corollary 4.2 concerns only positive programs, an immediate question arises on whether similar results hold for more general subclasses of programs. The rest of this section answers the question in the affirmative by developing a generalization of the "if" direction of the corollary. The generalization accepts programs with negative conditions over visible atoms. Moreover, the main premises of the generalization are stated in terms of simple translations of the compared programs. In preparation for the generalization and its proof, a number of supporting lemmas and definitions are given.

The first lemma states that visible strong equivalence $\equiv_{\mathrm{vs}}$ is a congruence relation for a *hiding operator* $\hbar$ defined below. Simply put, mutually hiding visible atoms preserves visible strong equivalence. To state the lemma, the following notation is defined. Namely, given a program P and a set of visible atoms $A \subseteq \mathrm{At_v}(P)$, we denote by $P \hbar A$ a program with the same set of rules as P, the same signature $\mathrm{At}(P \hbar A) = \mathrm{At}(P)$, a subset of its visible signature $\mathrm{At_v}(P \hbar A) = \mathrm{At_v}(P) \setminus A$, and a superset of its hidden signature $\mathrm{At_h}(P \hbar A) = \mathrm{At_h}(P) \cup A$. That is, the program $P \hbar A$ is otherwise the same as P, but has the atoms in A hidden in addition to any atoms already hidden in P.

Lemma 4.3. *For programs P and Q with $\mathrm{At_v}(P) = \mathrm{At_v}(Q)$, and a set $A \subseteq \mathrm{At_v}(P)$ of visible atoms, $P \equiv_{\mathrm{vs}} Q$ implies $P \hbar A \equiv_{\mathrm{vs}} Q \hbar A$.*

Proof. Let P, Q, and A be as above, so that $\mathrm{At_v}(P) = \mathrm{At_v}(Q)$ and $A \subseteq \mathrm{At_v}(P)$, and take any context program R that mutually respects the hidden atoms of $P \hbar A$ and $Q \hbar A$. Since the signatures of P and Q are equal to those of $P \hbar A$ and $Q \hbar A$, respectively, also P and Q respect the hidden atoms of R, and since the hidden signatures of P and Q are subsets of those of $P \hbar A$ and $Q \hbar A$, respectively, R respects the hidden atoms of P and Q also. Therefore, R is an applicable context program for P and Q, and the assumption $P \equiv_{\mathrm{vs}} Q$ guarantees that $\mathrm{SM}(P \cup R) =_{\mathrm{v}} \mathrm{SM}(Q \cup R)$ via a bijection f. Since $\mathrm{At_v}(P \hbar A) \subseteq \mathrm{At_v}(P)$, we have for every stable model $Y \in \mathrm{SM}(P \cup R)$ that

$$\begin{aligned} Y \cap \mathrm{At_v}(P) &= f(Y) \cap \mathrm{At_v}(P) \\ \Longrightarrow Y \cap \mathrm{At_v}(P \hbar A) &= f(Y) \cap \mathrm{At_v}(P \hbar A). \end{aligned}$$

Thus the sets of interpretations $\mathrm{SM}(P \cup R)$ and $\mathrm{SM}(Q \cup R)$ are visibly equal also when atoms in A are hidden. Since stable models are unaffected by changes in signature visibility, it follows that $\mathrm{SM}((P \hbar A) \cup R) =_{\mathrm{v}} \mathrm{SM}((Q \hbar A) \cup R)$ and thus $P \hbar A \equiv_{\mathrm{vs}} Q \hbar A$. In summary, $P \equiv_{\mathrm{vs}} Q$ implies $P \hbar A \equiv_{\mathrm{vs}} Q \hbar A$. □

An example of hiding in the context of concrete programs follows.

Example 4.4. Consider the logic programs

$$P\colon \quad a \leftarrow {\sim}b. \qquad\qquad Q\colon \quad a \leftarrow c. \quad c \leftarrow {\sim}b.$$

with visible signatures $\mathrm{At_v}(P) = \mathrm{At_v}(Q) = \{a, b\}$. These programs are visibly strongly equivalent as witnessed by the sets of VSE-models

$$\begin{array}{rl} \mathrm{VSE}(P)\colon & \langle\{a\},\{a\}\rangle \\ & \langle\emptyset,\{b\}\rangle \\ & \langle\{b\},\{b\}\rangle \\ & \langle\emptyset,\{a,b\}\rangle \\ & \langle\{a\},\{a,b\}\rangle \\ & \langle\{b\},\{a,b\}\rangle \\ & \langle\{a,b\},\{a,b\}\rangle \end{array} \qquad \begin{array}{rl} \mathrm{VSE}(Q)\colon & \langle\{a,c\},\{a,c\}\rangle \\ & \langle\emptyset,\{b\}\rangle \\ & \langle\{b\},\{b\}\rangle \\ & \langle\emptyset,\{a,b\}\rangle \\ & \langle\{a\},\{a,b\}\rangle \\ & \langle\{b\},\{a,b\}\rangle \\ & \langle\{a,b\},\{a,b\}\rangle \end{array}$$

As per Lemma 4.3, hiding any subset of the atoms $\{a,b\}$ from both programs preserves visible strong equivalence. The claim can be verified in the case that a is hidden by observing that there is a visible match between the VSE-models $\mathrm{VSE}(P \mathbin{⌧} \{a\}) = \{\langle\{a\},\{a\}\rangle, \langle\emptyset,\{b\}\rangle, \langle\{b\},\{b\}\rangle\}$ and $\mathrm{VSE}(Q \mathbin{⌧} \{a\}) = \{\langle\{a,c\},\{a,c\}\rangle, \langle\emptyset,\{b\}\rangle, \langle\{b\},\{b\}\rangle\}$. Similarly, hiding b yields $\mathrm{VSE}(P \mathbin{⌧} \{b\}) = \{\langle\{a\},\{a\}\rangle\}$ and $\mathrm{VSE}(Q \mathbin{⌧} \{b\}) = \{\langle\{a,c\},\{a,c\}\rangle\}$. Moreover, hiding both a and b yields the same sets of VSE-models as when hiding b, and therefore visible strong equivalence is again preserved. ■

One may notice that Lemma 4.3 holds only in the stated direction. Indeed, revealing hidden atoms may break visible strong equivalence. This is evidenced, for example, by the fact that any two programs are visibly strongly equivalent if all of their atoms are hidden, as long as they have equal numbers of stable models.

Example 4.5. Consider the logic programs $P = \{a.\}$ and $Q = \{b.\}$ with visible signatures $\mathrm{At_v}(P) = \mathrm{At_v}(Q) = \{a,b\}$. When both atoms a and b are hidden, the programs are visibly strongly equivalent, i.e., $P \mathbin{⌧} \{a,b\} \equiv_{\mathrm{vs}} Q \mathbin{⌧} \{a,b\}$. Yet, certainly $P \not\equiv_{\mathrm{vs}} Q$. ■

Next, a simple translation is given that will be used in the statement of the main result of the rest of this section, Proposition 4.12. The translation eliminates negative body literals based on given visible atoms a by substituting them with positive body literals that refer to fresh atoms a^-. We extend this naming notation to sets A of atoms by writing $A^- = \{a^- \mid a \in A\}$.

Definition 4.6. Given a program P and a set $A \subseteq \mathrm{At_v}(P)$ of visible atoms, the *negation substitution translation* $\mathrm{Tr}_{\simeq}(P,A)$ consists of the rules of P with every ${\sim}a$ replaced with a^-, where $a \in A$. The signature of the translation consists of the visible part $\mathrm{At_v}(\mathrm{Tr}_{\simeq}(P,A)) = \mathrm{At_v}(P) \cup A^-$ and the hidden part $\mathrm{At_h}(\mathrm{Tr}_{\simeq}(P,A)) = \mathrm{At_h}(P)$.

Example 4.7. The negation substitution translation of the program P:

$$d \leftarrow {\sim}a,\ {\sim}b,\ {\sim}c. \qquad e \leftarrow a,\ b.$$

given the set $A = \{a,b,c\}$ of atoms is $\mathrm{Tr}_{\simeq}(P,A)$:

$$d \leftarrow a^-,\ b^-,\ c^-. \qquad e \leftarrow a,\ b.$$ ■

On its own, the translation $\mathrm{Tr}_{\simeq}(\cdot,\cdot)$ breaks the logical connection between the original negative body literals ${\sim}a$ and the respective atoms a. This connection can be re-established by the addition of a context program that defines the auxiliary atoms a^- as equivalent to the respective negative conditions ${\sim}a$. This context program, given below, will be used in proofs leading up to Proposition 4.12.

Definition 4.8. Given a set $A \subseteq \mathrm{At_v}(P)$ of visible atoms, the *complement definition program* $\mathrm{Ctx}_{\sim}(A)$ contains the rule $a^- \leftarrow {\sim}a$ for every $a \in A$. The signature of the program consists of the visible part $\mathrm{At_v}(\mathrm{Ctx}_{\sim}(A)) = A$ and the hidden part $\mathrm{At_h}(\mathrm{Ctx}_{\sim}(A)) = \emptyset$.

The idea of eliminating negative conditions from rule bodies and then compensating them with a context program is already investigated in Reference [33]. There the purpose is to eliminate *unstratified negation* from programs. The difference here is that the class of considered programs is larger and the context program simpler.

Along these definitions, given a program P and a set A of atoms, the set of introduced auxiliary atoms $A^- = \{a^- \mid a \in A\}$ is always visible in both programs $\mathrm{Tr}_{\simeq}(P, A)$ and $\mathrm{Ctx}_{\sim}(A)$. Once the atoms in A^- are hidden, the union of these programs gives a visibly strongly equivalent refactoring of the original program P.

Lemma 4.9. *For a program P and a set $A \subseteq \mathrm{At_v}(P)$ of visible atoms, such that $A^- \cap \mathrm{At}(P) = \emptyset$, it holds that $P \equiv_{\mathrm{vs}} (\mathrm{Tr}_{\simeq}(P, A) \cup \mathrm{Ctx}_{\sim}(A)) \circledR A^-$.*

Proof. Let P and A be as above, let $Q = f(P, A)$ be a translation of P denoted by $f(P, A) = (\mathrm{Tr}_{\simeq}(P, A) \cup \mathrm{Ctx}_{\sim}(A)) \circledR A^-$, and let R be any context program that mutually respects the hidden atoms of P and Q. Observe that $f(P, A) = f(\cdots f(P, \{a_1\}) \cdots, \{a_n\})$, where $A = \{a_1, \ldots, a_n\}$ and that therefore, a proof of the lemma for $n = 1$ generalizes by induction to any $n \geq 0$. Hence, let us assume that $A = \{a\}$. Consider a mapping of interpretations $I \subseteq \mathrm{At}(P \cup R)$ of $P \cup R$ to visibly equal interpretations $J = I \cup \{a^- \mid a \notin I\}$ of $Q \cup R$. We will show that $I \in \mathrm{SM}(P \cup R)$ iff $J \in \mathrm{SM}(Q \cup R)$, i.e., $I \in \mathrm{MM}((P \cup R)^I)$ iff $J \in \mathrm{MM}((Q \cup R)^J)$. To this end, first the classical and then the minimal models of $(Q \cup R)^J$ are inspected.

As regards the classical models, observe that $\mathrm{CM}(Q \cup R) = \mathrm{CM}(\mathrm{Tr}_{\simeq}(P, \{a\})^I) \bowtie \mathrm{CM}(\mathrm{Ctx}_{\sim}(\{a\})^I) \bowtie \mathrm{CM}(R^I)$, where the reducts can be computed with respect to I instead of J given that $J \setminus I \subseteq \{a^-\}$ appears only in positive conditions. From the structure of the translations $\mathrm{Tr}_{\simeq}(P, \{a\})$ and $\mathrm{Ctx}_{\sim}(\{a\})$ it can be seen that

$$\begin{aligned}\mathrm{CM}(\mathrm{Tr}_{\simeq}(P, \{a\})^I) &= \{M \cup \{a^-\} \mid M \models P^{I \setminus \{a\}}\} \cup \{M \mid M \models P^{I \cup \{a\}}\}.\\ \mathrm{CM}(\mathrm{Ctx}_{\sim}(\{a\})^I) &= \{\{a^-\}, \{a^-, a\} \mid a \notin I\} \cup \{\emptyset, \{a\}, \{a^-\}, \{a^-, a\} \mid a \in I\}.\end{aligned}$$

Therefore,

$$\begin{aligned}&\mathrm{CM}(\mathrm{Tr}_{\simeq}(P, \{a\})^I) \bowtie \mathrm{CM}(\mathrm{Ctx}_{\sim}(\{a\})^I)\\ &\qquad = \{M \cup \{a^-\} \mid M \models P^{I \setminus \{a\}}\} \cup \{M \mid M \models P^I, a \in I\}.\end{aligned}$$

Moving on to the minimal models, let us denote by $\mathrm{Min}(\cdot)$ the set of subset minimal elements of a set. Moreover, to express joins involving the set $\{\{a^-\}\}$ of models, let us consider it to have the signature $\{a^-\}$. Given these notations, if $a \notin I$, then

$$\begin{aligned}\mathrm{MM}((Q \cup R)^J) &= \mathrm{Min}(\mathrm{CM}(P^I) \bowtie \{\{a^-\}\} \bowtie \mathrm{CM}(R^I))\\ &= \mathrm{MM}((P \cup R)^I) \bowtie \{\{a^-\}\}.\end{aligned}$$

However, if $a \in I$, then

$$\begin{aligned}\mathrm{MM}((Q \cup R)^J) &= \mathrm{Min}((\{M \cup \{a^-\} \mid M \models P^{I \setminus \{a\}}\} \cup \mathrm{CM}(P^I)) \bowtie \mathrm{CM}(R^I))\\ &= \mathrm{Min}(\{M \cup \{a^-\} \mid M \models P^{I \setminus \{a\}} \cup R^I\} \cup \mathrm{CM}((P \cup R)^I)).\end{aligned}$$

By the properties of the reduct, $P^{I \setminus \{a\}}$ is never easier to satisfy than P^I and hence for every model $M \models P^{I \setminus \{a\}}$, we have $M \models P^I$. Thus, the interpretations $M' \in \{M \cup \{a^-\} \mid M \models P^{I \setminus \{a\}} \cup R^I\}$ are eliminated by their subsets $M' \setminus \{a^-\} \models (P \cup R)^I$ from the above minimal models. Given this and the fact that R mutually respects the hidden atoms of P and Q, it follows that $\mathrm{MM}((Q \cup R)^J) = \mathrm{MM}((P \cup R)^I)$.

In both cases $a \notin I$ and $a \in I$, we can thus write $\mathrm{MM}((Q \cup R)^J) = \mathrm{MM}((P \cup R)^I) \bowtie \{\{a^- \mid a \notin I\}\}$. Hence, $I \in \mathrm{MM}((P \cup R)^I)$ iff $J \in \mathrm{MM}((Q \cup R)^J)$. Certainly, every stable model of $Q \cup R$ is of the form of J, and therefore the above establishes $\mathrm{SM}(P \cup R) =_\mathrm{v} \mathrm{SM}(Q \cup R)$ via the mapping $I \mapsto I \cup \{a^- \mid a \notin I\}$.

The above lemma helps us to prove a general theorem on visible strong equivalence of programs P and Q and the role of visible negation. The theorem states that the visible strong equivalence of two negation substitution translations operating on visible atoms carries over to their two argument programs. Stated more directly, when proving the visible strong equivalence of two programs, one may choose to treat negative conditions on a certain set of visible atoms as positive conditions on fresh, logically unrelated atoms. In the main anticipated use case, the program P is a source program and Q is a tentative translation of P that is intended to preserve the semantics of P in the sense of visible strong equivalence.

Theorem 4.10. *For programs P and Q with $\mathrm{At_v}(P) = \mathrm{At_v}(Q)$, and a set $A \subseteq \mathrm{At_v}(P)$ of visible atoms, such that $A^- \cap (\mathrm{At}(P) \cup \mathrm{At}(Q)) = \emptyset$, it holds that $\mathrm{Tr}_\simeq(P, A) \equiv_\mathrm{vs} \mathrm{Tr}_\simeq(Q, A)$ implies $P \equiv_\mathrm{vs} Q$.*

Proof. Let P and Q be programs with matching visible signatures $\mathrm{At_v}(P) = \mathrm{At_v}(Q)$, and let $A \subseteq \mathrm{At_v}(P)$ be a subset of the visible atoms. Assume $\mathrm{Tr}_\simeq(P, A) \equiv_\mathrm{vs} \mathrm{Tr}_\simeq(Q, A)$ holds. Recall Proposition 3.2, which states that $\equiv_\mathrm{vs}$ is a congruence relation for $\cup$ as long as the added program part respects the hidden atoms of the compared programs. Since the complement definition program $\mathrm{Ctx}_\sim(A)$ mutually respects the hidden atoms of $\mathrm{Tr}_\simeq(P, A)$ and $\mathrm{Tr}_\simeq(Q, A)$, we obtain by congruence that $\mathrm{Tr}_\simeq(P, A) \cup \mathrm{Ctx}_\sim(A) \equiv_\mathrm{vs} \mathrm{Tr}_\simeq(Q, A) \cup \mathrm{Ctx}_\sim(A)$. The set A^- of visible atoms defined in $\mathrm{Ctx}_\sim(A)$ on both of the sides can be hidden while preserving the equivalence according to Lemma 4.3, and therefore we obtain $\mathrm{Tr}_\simeq(P, A) \cup \mathrm{Ctx}_\sim(A) \circledcirc A^- \equiv_\mathrm{vs} \mathrm{Tr}_\simeq(Q, A) \cup \mathrm{Ctx}_\sim(A) \circledcirc A^-$. Applying Lemma 4.9 to both sides of this equivalence, yields that the left-hand side is visibly strongly equivalent to P while the right-hand side to Q. Finally, $P \equiv_\mathrm{vs} Q$ by transitivity. □

The converse of Theorem 4.10 does not hold in general, as witnessed by the following example.

Example 4.11. Consider logic programs $P = \{a \leftarrow {\sim}a.\ \}$ and $Q = \{a \leftarrow {\sim}b.\ b \leftarrow a.\ \}$ with visible signatures $\mathrm{At_v}(P) = \mathrm{At_v}(Q) = \{a\}$. The programs translate to $\mathrm{Tr}_\simeq(P, \{a\}) = \{a \leftarrow a^-.\ \}$ and $\mathrm{Tr}_\simeq(Q, \{a\}) = Q$. In this case, it can be proved that $P \equiv_\mathrm{vs} Q$. Nevertheless, $\mathrm{Tr}_\simeq(P, \{a\}) \not\equiv_\mathrm{vs} \mathrm{Tr}_\simeq(Q, \{a\})$. The lack of visible strong equivalence between the translations can be seen for example in the context $R = \{a^-.\ \}$ where $\mathrm{SM}(\mathrm{Tr}_\simeq(P, \{a\}) \cup R) = \{a, a^-\}$ while $\mathrm{SM}(\mathrm{Tr}_\simeq(Q, \{a\}) \cup R) = \emptyset$. ■

The use of negation substitution translations when proving visible strong equivalence via Theorem 4.10 becomes truly useful once the translated programs are free of any negation. In this specific case, Corollary 4.2 on positive programs guarantees that an inspection of hidden minimal models is sufficient to establish visible strong equivalence. This proof strategy is captured in the following result, which is obtained as a corollary of the mentioned results, Theorem 4.10 and Corollary 4.2.

Proposition 4.12. *For programs P and Q with $\mathrm{At_v}(P) = \mathrm{At_v}(Q)$ that are positive except for possibly any negative body atoms included in a set $A \subseteq \mathrm{At_v}(P)$ of visible atoms, such that $A^- \cap (\mathrm{At}(P) \cup \mathrm{At}(Q)) = \emptyset$, it holds that $\mathrm{MM_h}(\mathrm{Tr}_\simeq(P, A)) =_\mathrm{v} \mathrm{MM_h}(\mathrm{Tr}_\simeq(Q, A))$ implies $P \equiv_\mathrm{vs} Q$.*

The precondition of this proposition amounts to the requirement that programs P and Q must be without choice rules and without negative conditions based on hidden atoms.

To end this section, a few notes are presented on the usability of the main result of the section, Proposition 4.12. On the one hand, the comparison of hidden minimal models as opposed to VSE-models yields an improved level of convenience. On the other hand, the result holds only in the stated direction and is therefore incomplete, and its converse does not hold. This fact is

inherited from Theorem 4.10. As a consequence, the proposition is only useful in proving visible strong equivalence between some types of programs. The lack of completeness is related to the fact that the complement atoms of the form a^- are not logically connected to the respective original atoms a. Although the atoms a^- substitute negative body conditions, they are fresh auxiliary atoms without defining rules. Therefore, the VSE-models $\langle X, Y\rangle$ of the translated programs $\mathrm{Tr}_{\simeq}(\cdot, A)$ generally involve models X or Y where atoms a and their complements a^- are both either present or absent. Likewise, a hypothetical context program R may define them in arbitrary ways in terms of other visible atoms, with no regard to any functional dependency between a and a^-, as is seen in Example 4.11.

Whereas these reasons generally present an added challenge in proving the premises of Proposition 4.12, and may make it impossible, they are not a problem in certain interesting cases. Specifically, the lack of a logical connection between a complement atom a^- and the respective original atom a is of no concern when the original programs do not mention the atom a outside negative body conditions $\sim a$. Indeed, in such use cases, the translated programs are free of direct references to a, and therefore, no perplexing relations between a^- and a may arise. One may note that this condition is more likely to be satisfied by smaller than larger programs. In the extreme case, the condition is trivially satisfied by programs consisting of a single rule, as long as the rule contains no repeated occurrences of atoms with opposite polarities. This makes it particularly viable to apply the proposition to single rules and their translated or normalized versions, for example. Moreover, this kind of equivalence checking lends itself naturally to *modular* settings where two programs are split into unions of smaller programs, and these smaller programs are verified to be visibly strongly equivalent using the proposition. This type of rule-by-rule verification is further studied in Section 5.

5 APPLICATION IN PROGRAM TRANSFORMATIONS

In this section, we illustrate the use of visible strong equivalence in a practical setting. We present and analyze program transformations that refactor parts of programs while preserving the meaning of the programs. In this context, we regard a transformation correct if it maps input rules to visibly strongly equivalent sets of output rules. The transformations considered here are *normalizations* of extended rule types, and more specifically, of choice rules (2) and cardinality (3) rules as illustrative examples. We refer by normalization to the substitution of heterogeneous types of rules with semantically identical sets of normal rules, which typically involve some fresh auxiliary atoms that help reduce the number of used rules. Normalization is useful, for example, when translating answer-set programs into formalisms with no extended rule support such as classical logic [24], when debugging answer-set programs [39], when implementing extended rule support for new solvers [7], and when pursuing performance improvements on benchmarks [5, 6]. As discussed previously, strong equivalence [29] as well as relativized strong equivalence [45, 46] are prohibitively strict in this context given that they compare not only visible atoms, but undesirably also hidden atoms (recall Example 2.11). In contrast, visible strong equivalence suits the task. We demonstrate this first in the case of choice rules, for which we use VSE-models and the correspondence established in Theorem 3.21, and then for cardinality rules, for which we use hidden minimal models as justified by the results of Section 4.

As presented in Reference [24], a choice rule of the form (2) can be turned into a normal rule $d \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m$ where the head has been replaced by a new atom d denoting that the body of the rule is satisfied. Moreover, we need $2h$ normal rules

$$a_1 \leftarrow d, \sim\overline{a_1}. \qquad \overline{a_1} \leftarrow \sim a_1. \qquad \ldots \qquad a_h \leftarrow d, \sim\overline{a_h}. \qquad \overline{a_h} \leftarrow \sim a_h.$$

with new atoms $\overline{a_1}, \ldots, \overline{a_h}$ that denote the complements of $a_1, \ldots, a_h$, respectively.

Example 5.1. Consider a program P consisting of a single choice rule $\{a\} \leftarrow b, {\sim}c$ such that $\mathrm{At_v}(P) = \{a, b, c\}$ and $\mathrm{At_h}(P) = \emptyset$. The translation $Q = \{a \leftarrow d, {\sim}\overline{a}.\ d \leftarrow b, {\sim}c.\ \overline{a} \leftarrow {\sim}a.\ \}$ such that $\mathrm{At_v}(Q) = \mathrm{At_v}(P)$ and $\mathrm{At_h}(Q) = \{\overline{a}, d\}$. Any subset $Y \subseteq \mathrm{At}(P)$ satisfies P. The reduct P^Y contains $a \leftarrow b$ iff $a \in Y$ and $c \notin Y$. Thus only $Y = \{a, b\}$ gives rise to exceptional VSE-models $\langle\emptyset, \{a, b\}\rangle$, $\langle\{a\}, \{a, b\}\rangle$, $\langle\{a, b\}, \{a, b\}\rangle$ whereas all other VSE-models based on $Y \neq \{a, b\}$ take the form $\langle X, Y\rangle$ with any $X \subseteq Y$. A summary of the VSE-models is given in the table below. Any model $Y \subseteq \mathrm{At}(P)$ of P (and P^Y) can be turned into a $\mathrm{At_h}(Q)$-minimal model Y' of Q (resp. Q^Y) by adding $d \in Y'$ iff $b \in Y$ and $c \notin Y$, and by adding $\overline{a} \in Y'$ iff $a \notin Y$. The respective $\mathrm{At_h}(Q)$-minimal models X' of $Q^{Y'}$ are listed in the final column of the table.

Y	P^Y	X	Y'	$Q^{Y'}$	X'
$\emptyset$		$\emptyset$	$\{\overline{a}\}$	$d \leftarrow b.\ \overline{a}.$	$\{\overline{a}\}$
$\{a\}$	$a \leftarrow b.$	$\emptyset \ldots \{a\}$	$\{a\}$	$a \leftarrow d.\ d \leftarrow b.$	$\emptyset \ldots \{a\}$
$\{b\}$		$\emptyset \ldots \{b\}$	$\{\overline{a}, b, d\}$	$d \leftarrow b.\ \overline{a}.$	$\{\overline{a}\}, \{\overline{a}, b, d\}$
$\{c\}$		$\emptyset \ldots \{c\}$	$\{\overline{a}, c\}$	$\overline{a}.$	$\{\overline{a}\}, \{\overline{a}, c\}$
$\{a, b\}$	$a \leftarrow b.$	$\emptyset, \{a\}, \{a, b\}$	$\{a, b, d\}$	$a \leftarrow d.\ d \leftarrow b.$	$\emptyset, \{a\}\{a, b, d\}$
$\{a, c\}$		$\emptyset \ldots \{a, c\}$	$\{a, c\}$	$a \leftarrow d.$	$\emptyset \ldots \{a, c\}$
$\{b, c\}$		$\emptyset \ldots \{b, c\}$	$\{\overline{a}, b, c\}$	$\overline{a}.$	$\{\overline{a}\} \ldots \{\overline{a}, b, c\}$
$\{a, b, c\}$		$\emptyset \ldots \{a, b, c\}$	$\{a, b, c\}$	$a \leftarrow d.$	$\emptyset \ldots \{a, b, c\}$

The sets $\pi_2(\mathrm{VSE}(P))$ and $\pi_2(\mathrm{VSE}(Q))$ are in one-to-one correspondence and coincide up to $\mathrm{At_v}(P) = \mathrm{At_v}(Q) = \{a, b, c\}$. It is also easy to check (6) for each pair Y and Y'. Thus $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$ and $P \equiv_{\mathrm{vs}} Q$ by Theorem 3.21. In other words, the choice rule and its translation behave identically in any context not referring to d and $\overline{a}$.

Finally, let us note that SE-models of P coincide with VSE-models. The relativized $\{a, b, c\}$-SE-models of Q can be obtained using the respective projections of X'. ■

Proposition 5.2. *Let P be a program consisting of a choice rule $\{A\} \leftarrow B, {\sim}C$ and Q its normalization in the way described above. Then,* $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$.

Proof. As regards signatures, we have $\mathrm{At_v}(P) = A \cup B \cup C$ and $\mathrm{At_h}(P) = \emptyset$, which reduces $\mathrm{VSE}(P)$ to $\mathrm{SE}(P)$. Then, $Y \models P$ holds for any $Y \subseteq \mathrm{At}(P)$ as choice rules are always satisfied by definition. However, we have $\mathrm{At_v}(Q) = \mathrm{At_v}(P)$ and $\mathrm{At_h}(Q) = \{d\} \cup \overline{A}$ where $\overline{A} = \{\overline{a} \mid a \in A\}$. Then, consider any $Y \subseteq \mathrm{At}(P)$ and its extension $Y' = f(Y) = Y \cup \{d \mid B \subseteq Y \text{ and } C \cap Y = \emptyset\} \cup \{\overline{a} \mid a \in A \setminus Y\}$ to an interpretation of Q. It is clear that $Y' \models d \leftarrow B, {\sim}C$ directly by the definition of Y'. Likewise, $Y' \models \overline{a} \leftarrow {\sim}a$ holds for any $a \in A$, as $a \notin Y'$ implies $a \notin Y$ and $\overline{a} \in Y'$. Moreover, assuming that $Y' \not\models a \leftarrow d, {\sim}\overline{a}$ implies that $\overline{a} \notin Y$ and $a \notin Y'$, which contradict by the definition of Y'. Hence $Y' \models Q$ and $Y' \models Q^{Y'}$ by Lemma 2.2.

Let us then suppose that Y' is not $\mathrm{At_h}(Q)$-minimal, i.e., there is $X' \models Q^{Y'}$ such that $X' \subset Y'$ and $X'_{\mathrm{v}} = Y'_{\mathrm{v}}$. (i) Assuming $d \in Y' \setminus X'$ implies $d \notin X'$, $d \in Y'$, and $Y' \models B \cup {\sim}C$ by the definition of Y'. Thus $Y' \models {\sim}C$ so that $d \leftarrow B \in Q^{Y'}$. However $Y' \models B$ and $X' \models B$ as $X'_{\mathrm{v}} = Y'_{\mathrm{v}}$. Since $d \notin X'$ we have $X' \not\models Q^{Y'}$, a contradiction. (ii) Assuming that $\overline{a} \in Y' \setminus X'$ gives us $\overline{a} \notin X'$ but $\overline{a} \in Y'$. The latter implies $a \notin Y'$ by the definition of Y'. Thus $\overline{a}$ appears as a fact in $Q^{Y'}$ so that $X' \not\models Q^{Y'}$ a contradiction. By (i) and (ii), we conclude that Y' is $\mathrm{At_h}(Q)$-minimal.

Thus $\pi_2(\mathrm{VSE}(P)) =_{\mathrm{v}} \pi_2(\mathrm{VSE}(Q))$ via f defined above. To establish (6), let us consider any $Y \subseteq \mathrm{At}(P)$, $Y' = f(Y)$, and X_{v} for which $\langle X, Y\rangle \in \mathrm{VSE}(P) = \mathrm{SE}(P)$, i.e., $X \subseteq Y$ and $X \models P^Y$. Note that $X = X_{\mathrm{v}}$ is trivially $\mathrm{At_h}(P)$-minimal as $\mathrm{At_h}(P) = \emptyset$ by definition. The reducts of P and Q with respect to Y and Y' are

$$\begin{aligned} P^Y &= \{a \leftarrow B \mid a \in A \cap Y \text{ and } C \cap Y = \emptyset\} \text{ and} \\ Q^{Y'} &= \{a \leftarrow d \mid a \in A \cap Y\} \cup \{d \leftarrow B \mid C \cap Y = \emptyset\} \cup \{\overline{a} \mid a \in A \setminus Y\} \end{aligned}$$

by the definition of $Y' = f(Y)$. Then, define an interpretation of Q by setting

$$X' = X \cup \{d \mid B \subseteq X \text{ and } C \cap Y = \emptyset\} \cup \{\overline{a} \mid a \in A \setminus Y\},$$

which is contained in $Y' = f(Y)$ as $X \subseteq Y$. The inspection of $X' \models Q^{Y'}$ leads to three cases. (i) Assuming $X' \not\models a \leftarrow d$ for some $a \in A \cap Y$ implies that $a \notin X'$ and $d \in X'$, i.e., $a \notin X$ and $B \subseteq X$, and $C \cap Y = \emptyset$. Since $a \in Y$ and $C \cap Y = \emptyset$, we know that $a \leftarrow B \in P^Y$ so that $X \not\models P^Y$, a contradiction. (ii) Suppose that $X' \not\models d \leftarrow B$ and $C \cap Y = \emptyset$. It follows that $d \notin X'$ and $B \subseteq X'$, i.e., $B \not\subseteq X$, as $C \cap Y = \emptyset$, and $B \subseteq X'_{\mathrm{v}} = X$, a contradiction. (iii) By definition, X' satisfies each fact $\overline{a}$ with $a \in A \setminus Y$. Thus $X' \models Q^{Y'}$ follows by (i)–(iii). To argue for the $\mathrm{At_h}(Q)$-minimality of X', let us consider $Z' \models Q^{Y'}$ such that $Z'_{\mathrm{v}} = X'_{\mathrm{v}} = X_{\mathrm{v}}$ and $Z'_{\mathrm{h}} \subset X'_{\mathrm{h}}$. Now d cannot be in the difference, because $d \in X'_{\mathrm{h}} \setminus Z'_{\mathrm{h}}$ implies $B \subseteq X$ and $C \cap Y = \emptyset$, i.e., $d \leftarrow B \in Q^{Y'}$, $Z' \models B$ as $Z'_{\mathrm{v}} = X_{\mathrm{v}}$, and $Z' \not\models Q^{Y'}$ as $d \notin Z'_{\mathrm{h}}$. Nor can the difference be explained by any $a \in A \setminus Y$ for which $Q^{Y'}$ includes $\overline{a}$ as a fact. Thus $\langle X', Y' \rangle \in \mathrm{VSE}(Q)$ and $X'_{\mathrm{v}} = X_{\mathrm{v}}$ is contained in the right-hand side of Equation (6).

Then, consider any X'_{v} such that $\langle X', Y' \rangle \in \mathrm{VSE}(Q)$. This makes X' an $\mathrm{At_h}(Q)$-minimal model of $Q^{Y'}$. Then, define $X = X' \cap \mathrm{At}(P) = X'_{\mathrm{v}}$ and assume that $X \not\models P^Y$. It follows that for some $a \in A \cap Y$, $a \notin X$, $B \subseteq X$, and $C \cap Y = \emptyset$. The reduct $Q^{Y'}$ contains $a \leftarrow d$ and $d \leftarrow B$. Since $B \subseteq X \subseteq X'$ and $X' \models Q^{Y'}$, we obtain $d \in X'$. But then $a \in X'$, as $X' \models Q^{Y'}$, and $a \in X$, contradiction. Thus $X \models P^Y$, $\langle X, Y \rangle \in \mathrm{SE}(P) = \mathrm{VSE}(P)$, and Equation (6) holds. □

Cardinality rules are much harder to normalize succinctly. In Reference [24], we present a transformation following an approach introduced in the context of satisfiability checking [11]. To represent Equation (3) in general, a total of $l \times (n + m - l + 1)$ new atoms $d_{i,j}$ are required. Here $1 \leq i \leq l$ is the count of satisfied literals and $l - i + 1 \leq j \leq n + m - i + 1$ points to the jth literal (either b_j or $\sim c_{j-n}$) in the body of Equation (3). The reading of $d_{i,j}$ is that the number of satisfied literals, from the jth up to the $(n + m)$th literal, is at least i. Thus, $d_{l,1}$ captures the intended condition for the head a of Equation (3). The rules are as follows:

$$\begin{aligned} &a \leftarrow d_{l,1}. && \\ &d_{i,j} \leftarrow d_{i,j+1}. && (1 \leq i \leq l \text{ and } l - i + 1 \leq j < n + m - i + 1) \\ &d_{i,j} \leftarrow d_{i-1,j+1},\ b_j. && (1 < i \leq l \text{ and } l - i + 1 \leq j \leq n) \\ &d_{i,j} \leftarrow d_{i-1,j+1},\ \sim c_{j-n}. && (1 < i \leq l \text{ and } \max(l, n + 1) \leq j \leq n + m - i + 1) \\ &d_{1,j} \leftarrow b_j. && (l \leq j \leq n) \\ &d_{1,j} \leftarrow \sim c_{j-n}. && (\max(l, n + 1) \leq j \leq n + m) \end{aligned}$$

The rules formalize a kind of a *counting grid* where vertical moves (up) increase i upon a satisfaction of a literal. Horizontal moves (left) are possible by default when a literal is not satisfied—decreasing j. The positive case is analyzed below, first via the example $l = 1$, $n = 2$, $m = 0$, and then via a lemma on the general correctness of the translation. Then, the case allowing negation $m >= 0$ is considered.

Example 5.3. Consider a positive program P consisting of $a \leftarrow 1 \leq \{b, c\}$ and its translation $Q = \{a \leftarrow d_{1,1}.\ d_{1,1} \leftarrow d_{1,2}.\ d_{1,1} \leftarrow b.\ d_{1,2} \leftarrow c.\ \}$. Let $\mathrm{At_v}(P) = \mathrm{At_v}(Q) = \{a, b, c\}$, $\mathrm{At_h}(P) = \emptyset$, and $\mathrm{At_h}(Q) = \{d_{1,1}, d_{1,2}\}$. There are five classical models $Y \subseteq \mathrm{At}(P)$ of P, which are also hidden minimal models due to the lack of hidden atoms in P. Each $Y \models P$ can be turned into a $\mathrm{At_h}(Q)$-minimal model Y' of Q by adding $d_{1,1} \in Y'$ when $b \in Y$ or $c \in Y$, and $d_{1,2} \in Y'$ when $c \in Y$.

Y	Y'
$\emptyset$	$\emptyset$
$\{a\}$	$\{a\}$
$\{a, b\}$	$\{a, b, d_{1,1}\}$
$\{a, c\}$	$\{a, c, d_{1,1}, d_{1,2}\}$
$\{a, b, c\}$	$\{a, b, c, d_{1,1}, d_{1,2}\}$

The bijective relationship of $Y \in \mathrm{MM_h}(P)$ with $Y' \in \mathrm{MM_h}(Q)$ is easy to inspect. Thus, $\mathrm{MM_h}(P) =_\mathrm{v} \mathrm{MM_h}(Q)$, and hence Corollary 4.2 gives $P \equiv_\mathrm{vs} Q$. ■

Lemma 5.4. *Let P be a program consisting of a positive cardinality rule $a \leftarrow l \leq \{B\}$ and Q its normalization in the way described above. Then,* $\mathrm{MM_h}(P) =_\mathrm{v} \mathrm{MM_h}(Q)$.

Proof. Suppose that the rule has the form of Equation (3) with no negative body literals, so that $m = 0$ and $C = \emptyset$, and define the relevant signatures by setting $\mathrm{At_v}(P) = \{a\} \cup B \cup C$, $\mathrm{At_h}(P) = \emptyset$, $\mathrm{At_v}(Q) = \mathrm{At_v}(P)$, and

$$\mathrm{At_h}(Q) = \{d_{i,j} \mid 1 \leq i \leq l \text{ and } l - i + 1 \leq j \leq n - i + 1\}.$$

By Corollary 4.2, it suffices to show that $\mathrm{MM_h}(P) =_\mathrm{v} \mathrm{MM_h}(Q)$. To this end, consider any $Y \models P$ and define an interpretation $Y' = f(Y)$ by adding to Y every $d_{i,j} \in \mathrm{At_h}(Q)$ for which the number of satisfied literals under Y, from the jth to the nth literal, is at least i. We prove by complete induction on the depth $d(j) = n + 1 - j$ that Y' satisfies the rules of Q whose head atom is $d_{i,j}$.

- For the base case $d(j) = 1$, consider $j = n$ and the satisfaction of $d_{1,n} \leftarrow b_n$. If Y satisfies b_n, then the number of literals from b_n to b_n that Y satisfies is trivially 1. Hence, $Y' \models d_{1,n}$ and thus $Y' \models d_{1,n} \leftarrow b_n$.
- Then, assume that $d(j) > 1$. There are three kinds of rules to consider, given that the rules with negation are not instantiated. (i) Assuming that $Y' \not\models d_{i,j} \leftarrow d_{i,j+1}$ implies that $d_{i,i} \notin Y'$ and $d_{i,j+1} \in Y'$. Since $d(j + 1) < d(j)$, we obtain by the definition of Y' and the inductive hypothesis that the number of literals satisfied by Y, from the $(j + 1)$th to the nth literal, is at least i. This clearly holds even if we consider the jth literal as well so that $d_{i,j} \in Y'$ by the definition of Y', a contradiction. (ii) The assumption $Y' \not\models d_{i,j} \leftarrow d_{i-1,j+1}, b_j$ leads to similar argumentation except that the number of satisfied literals under Y is $i - 1$ by the inductive hypothesis. Furthermore, $b_j \in Y'$ implies $b_j \in Y$ so that the number of satisfied literals is at least i and $d_{i,j} \in Y'$ by the definition of Y', a contradiction. Last, the rule type (iii) with $i = 1$ can be treated as in the base case.

It remains to consider the rule $a \leftarrow d_{l,1}$. If it were not satisfied by Y', then we would have $a \notin Y'$ and $d_{l,1} \in Y'$. The former implies $a \in Y$ whereas the latter indicates that the number of literals satisfied by Y in the body of (3) is at least l. But this contradicts the fact that $Y \models P$. Hence $Y' \models Q$.

To establish the $\mathrm{At_h}(Q)$-minimality of Y', let us consider $Z' \models Q$ such that $Z' \subseteq Y'$ and $Z'_\mathrm{v} = Y'_\mathrm{v} = Y$. Again, we use complete induction on $d(j)$ to show that $d_{i,j} \in Y'$ implies $d_{i,j} \in Z'$. The base case $d_{1,n} \in Y'$ implies by the definition of $Y' = f(Y)$ that $Y \models b_n$. Thus b_n is true under Z' as $Z'_\mathrm{v} = Y$ and $d_{1,n} \in Z'$ as $Z' \models Q$. For the induction step, we note that $d_{i,j} \in Y'$ implies that the number of satisfied literals under Y, from the jth to the nth literal, is at least i. The relevant rule of Q is $d_{i,j} \leftarrow d_{i,j+1}$ together with $d_{i,j} \leftarrow d_{i-1,j+1}, b_j$.[5] If Y satisfies at least i literals from the $(j + 1)$th to the nth, then $d_{i,j+1} \in Y'$ by the definition of Y' and $d_{i,j+1} \in Z'$ by the inductive hypothesis. It follows that $d_{i,j} \in Z'$ as $Z' \models d_{i,j} \leftarrow d_{i,j+1}$. Otherwise, the number of satisfied literals is $i - 1$

[5]The literal $d_{i-1,j+1}$ is not present in the latter rule if $i = 1$.

from the $(j+1)$th literal onward. Thus we necessarily have $Y \models b_j$. The same applies to Z', as $Z'_{\mathrm{v}} = Y$. But then $d_{i,j} \in Z'$, since Z' satisfies the respective rules. Thus, $Z' = Y'$ making Y' then $\mathrm{At_h}(Q)$-minimal.

However, consider an $\mathrm{At_h}(Q)$-minimal model Y' of Q. Then, define Y as the projection $Y = Y' \cap \mathrm{At}(P) = Y'_{\mathrm{v}}$. Again, we can show by induction on $d(j)$ that $d_{i,j} \in Y'$ iff the number of literals satisfied by Y, from the jth literal to the nth, is at least i. Assuming that $Y \not\models P$ implies $a \notin Y$ but the number of literals satisfied by Y in the body of Equation (3) is at least l. It follows that $d_{l,1} \in Y'$ and $a \notin Y'$ by the definition of Y, i.e., $Y' \not\models a \leftarrow d_{l,1}$, a contradiction. Hence $Y \models P$. Since $Y' = f(Y)$ uniquely extends Y to a $\mathrm{At_h}(Q)$-minimal model of Q, we have practically established that $\mathrm{MM_h}(P) =_{\mathrm{v}} \mathrm{MM_h}(Q)$ via f. The inverse of f can be defined for $Y' \in \mathrm{MM_h}(Q)$ by setting $f^{-1}(Y') = Y' \cap \mathrm{At}(P)$. □

Now that the correctness of the counting grid translation has been established for positive cardinality rules, the case allowing negation remains. Again, an example is analyzed first, here with $l = n = m = 1$, and then a general correctness result is given. This time, the example resorts to VSE-models to cope with negation. In contrast to this, the proof of the correctness result exploits the results of Section 4 to generalize Lemma 5.4 to the case with negation.

Example 5.5. Consider a program P consisting of $a \leftarrow 1 \leq \{b, {\sim}c\}$ and its translation $Q = \{a \leftarrow d_{1,1}.\ d_{1,1} \leftarrow d_{1,2}.\ d_{1,1} \leftarrow b.\ d_{1,2} \leftarrow {\sim}c.\ \}$. Let $\mathrm{At_v}(P) = \mathrm{At_v}(Q) = \{a, b, c\}$, $\mathrm{At_h}(P) = \emptyset$, and $\mathrm{At_h}(Q) = \{d_{1,1}, d_{1,2}\}$. There are five classical models $Y \subseteq \mathrm{At}(P)$ of P and the reduct P^Y contains $a \leftarrow 1 \leq \{b\}$ if $c \in Y$, and $a \leftarrow 0 \leq \{b\}$ if $c \notin Y$. Each $Y \models P$ can be turned into a $\mathrm{At_h}(Q)$-minimal model Y' of Q by adding $d_{1,2} \in Y'$ iff $c \notin Y$, and by adding $d_{1,1} \in Y'$ iff $b \in Y$ or $c \notin Y$. The reduct $Q^{Y'}$ contains the first three rules of Q and $d_{1,2}$ as a fact iff $c \notin Y'$. The respective $\mathrm{At_h}(Q)$-minimal models X' are reported below:

Y	P^Y	X	Y'	X'
$\{a\}$	$a \leftarrow 0 \leq \{b\}$	$\{a\}$	$\{a, d_{1,1}, d_{1,2}\}$	$\{a, d_{1,1}, d_{1,2}\}$
$\{c\}$	$a \leftarrow 1 \leq \{b\}$	$\emptyset \ldots \{c\}$	$\{c\}$	$\emptyset \ldots \{c\}$
$\{a, b\}$	$a \leftarrow 0 \leq \{b\}$	$\{a\}, \{a, b\}$	$\{a, b, d_{1,1}, d_{1,2}\}$	$\{a, d_{1,1}, d_{1,2}\}$, $\{a, b, d_{1,1}, d_{1,2}\}$
$\{a, c\}$	$a \leftarrow 1 \leq \{b\}$	$\emptyset \ldots \{a, c\}$	$\{a, c\}$	$\emptyset \ldots \{a, c\}$
$\{a, b, c\}$	$a \leftarrow 1 \leq \{b\}$	$\emptyset \ldots \{a, c\}$, $\{a, b\}, \{a, b, c\}$	$\{a, b, c, d_{1,1}\}$	$\emptyset \ldots \{a, c\}$, $\{a, b, d_{1,1}\}, \{a, b, c, d_{1,1}\}$

Again, the bijective relationship of $Y \in \pi_2(\mathrm{VSE}(P))$ with $Y' \in \pi_2(\mathrm{VSE}(Q))$ is easy to inspect and, furthermore, Equation (6) holds for each such pair of Y and $Y' = f(Y)$. Thus, $\mathrm{VSE}(P) \overset{\mathrm{v}}{=} \mathrm{VSE}(Q)$ and $P \equiv_{\mathrm{vs}} Q$ follows by Theorem 3.21. ■

Proposition 5.6. *Let P be a program consisting of a cardinality rule $a \leftarrow l \leq \{B, {\sim}C\}$ where $C \cap (\{a\} \cup B) = \emptyset$ and Q its normalization in the way described above. Then, $P \equiv_{\mathrm{vs}} Q$.*

Proof. Suppose that the rule has the form of Equation (3) and define the relevant signatures by setting $\mathrm{At_v}(P) = \{a\} \cup B \cup C$, $\mathrm{At_h}(P) = \emptyset$, $\mathrm{At_v}(Q) = \mathrm{At_v}(P)$, and

$$\mathrm{At_h}(Q) = \{d_{i,j} \mid 1 \leq i \leq l \text{ and } l - i + 1 \leq j \leq n + m - i + 1\}.$$

The negation substitution translation $\mathrm{Tr}_{\simeq}(P, C)$ of the cardinality rule in P consists of the single positive cardinality rule $a \leftarrow l \leq \{B, C^-\}$. Moreover, the negation substitution translation $\mathrm{Tr}_{\simeq}(Q, C)$ of the counting grid translation of P coincides with the counting grid translation of the negation substitution translation $\mathrm{Tr}_{\simeq}(P, C)$ of P. Since both negation substitution translations are positive and of the forms expected in the preconditions of Lemma 5.4, it follows

that $\mathrm{MM_h}(\mathrm{Tr}_{\simeq}(P,C)) =_{\mathrm{v}} \mathrm{MM_h}(\mathrm{Tr}_{\simeq}(Q,C))$. By Corollary 4.2, this further implies $\mathrm{Tr}_{\simeq}(P,C) \equiv_{\mathrm{vs}} \mathrm{Tr}_{\simeq}(Q,C)$. Since $C \subseteq \mathrm{At_v}(P)$ and $C^- \cap (\mathrm{At}(P) \cup \mathrm{At}(Q)) = \emptyset$, Theorem 4.10 gives $P \equiv_{\mathrm{vs}} Q$. □

Propositions 5.2 and 5.6 imply that the translations of choice and cardinality rules above can be safely used in any context which mutually respects the hidden atoms involved.

6 TRANSLATION-BASED VERIFICATION

The goal of this section is to develop a translation-based method to check whether $P \equiv_{\mathrm{vs}} Q$ holds for two programs P and Q. We will restrict ourselves to the case of SMODELS programs and use SMODELS-compatible solvers for actual computations. To realize this, we have to further constrain the class of programs under consideration, i.e., we assume that each program has *enough visible atoms* (the so-called EVA property [25]). This means that there is a unique stable model for *the hidden part of P relative to I_{v}*, which is a program obtained from P by partially evaluating the visible parts of rules and keeping the remaining hidden parts [25]. The evaluation process is in close analogy to how the usual ASP reduct is obtained by partially evaluating the negative parts of rules and keeping the remaining positive parts. Thus, each $\mathrm{At_h}(P)$-minimal model Y of P^Y becomes unique up to $\mathrm{At_v}(P)$. This reduces the complexity of verifying $\equiv_{\mathrm{vs}}$ to a level where actual SMODELS-compatible solvers can be used.

The overall translation for the verification task is devised in two steps. First, we translate an SMODELS program P into another program $\mathrm{Tr_{vs}}(P)$ that has the VSE-models of P as its stable models. The VSE-models of Q are analogously captured by $\mathrm{Tr_{vs}}(Q)$. After this we can apply an existing translation, $\mathrm{Tr_{eq}}(\cdot,\cdot)$, for the verification of weak equivalence [25] to finalize our approach: $P \equiv_{\mathrm{vs}} Q$ iff $\mathrm{Tr_{vs}}(P) \equiv \mathrm{Tr_{vs}}(Q)$ iff the symmetric translations $\mathrm{Tr_{eq}}(\mathrm{Tr_{vs}}(P), \mathrm{Tr_{vs}}(Q))$ and $\mathrm{Tr_{eq}}(\mathrm{Tr_{vs}}(Q), \mathrm{Tr_{vs}}(P))$ have no stable models. The below translation $\mathrm{Tr_{vs}}(P)$ introduces a new atom $a^\bullet$ for each atom $a \in \mathrm{At}(P)$. These atoms represent the unique *least model* $X = \mathrm{LM}(P^Y)$ given a $\mathrm{At_h}(P)$-minimal model Y of P^Y. We extend the notation to sets $S \subseteq \mathrm{At}(P)$ of atoms by setting $S^\bullet = \{a^\bullet \mid a \in S\}$.

DEFINITION 6.1. The translation $\mathrm{Tr_{vs}}(P)$ extends an SMODELS program P with

(1) choice rules $\{a\}$ and $\{a^\bullet\} \leftarrow a$ for each visible atom $a \in \mathrm{At_v}(P)$;
(2) a constraint $\leftarrow a^\bullet, {\sim}a$ for each visible atom $a \in \mathrm{At_v}(P)$;
(3) a rule $a^\bullet \leftarrow B^\bullet, {\sim}C$ for each normal rule $a \leftarrow B, {\sim}C$ in P;
(4) a rule $a^\bullet \leftarrow B^\bullet \cup \{a\}, {\sim}C$ for each choice rule $\{A\} \leftarrow B, {\sim}C$ in P and each head atom $a \in A$; and
(5) a rule $a^\bullet \leftarrow l \leq \{B^\bullet, {\sim}C\}$ for each cardinality rule $a \leftarrow l \leq \{B, {\sim}C\}$ in P;
(6) a rule $a^\bullet \leftarrow w \leq \{B^\bullet = W_{B^\bullet}, {\sim}C = W_C\}$ for each weight rule $a \leftarrow w \leq \{B = W_B, {\sim}C = W_C\}$ in P.

Let $\mathrm{At_v}(\mathrm{Tr_{vs}}(P)) = \mathrm{At_v}(P) \cup \mathrm{At_v}(P)^\bullet$ and $\mathrm{At_h}(\mathrm{Tr_{vs}}(P)) = \mathrm{At_h}(P) \cup \mathrm{At_h}(P)^\bullet$.

THEOREM 6.2. *Let P be an* SMODELS *program with the EVA property and $\mathrm{Tr_{vs}}(P)$ its translation to capture the VSE-models of P. Then, for any $X \subseteq Y \subseteq \mathrm{At}(P)$, $\langle X, Y\rangle \in \mathrm{VSE}(P)$ if and only if $M = Y \cup X^\bullet \in \mathrm{SM}(\mathrm{Tr_{vs}}(P))$.*

PROOF. Consider any $\langle X, Y\rangle$ with $X \subseteq Y \subseteq \mathrm{At}(P)$ and the respective interpretation $Y \cup X^\bullet \subseteq \mathrm{At}(\mathrm{Tr_{vs}}(P))$. The reduct $\mathrm{Tr_{vs}}(P)^{Y \cup X^\bullet}$ consists of P^Y, the fact a for each $a \in Y_{\mathrm{v}}$, the rule $a^\bullet \leftarrow a$ for each $a \in X_{\mathrm{v}}$, the constraint $\leftarrow a^\bullet$ for each $a \in \mathrm{At_v}(P) \setminus Y_{\mathrm{v}}$, and $(P^Y)^\bullet$. It follows that $Y \cup X^\bullet = \mathrm{LM}(\mathrm{Tr_{vs}}(P)^{Y \cup X^\bullet})$ iff Y is a $\mathrm{At_h}(P)$-minimal model of P^Y, $X \subseteq Y$, and X is a $\mathrm{At_h}(P)$-minimal model of P^Y. Hence the result. □

Table 1. CPU Time Taken and Numbers of Conflicts Encountered When Verifying $P \equiv_{\mathrm{vs}} Q$ for a Choice Rule P with n Literals and Its Translation Q

n		128	256	512	1,024	2,048	4,096	8,192	16,384
time (s)	(a)	0.0	0.2	1.4	8.7	60.7	440.3	3340.2	20152.2
	(b)	0.0	0.2	1.5	10.4	75.2	581.1	4387.2	23401.5
conflicts	(a)	216	259	514	1026	2051	4098	8195	16387
	(b)	289	545	1058	2040	3990	12140	25172	49925

(a) $\mathrm{Tr}_{\mathrm{eq}}(\mathrm{Tr}_{\mathrm{vs}}(P), \mathrm{Tr}_{\mathrm{vs}}(Q))$.
(b) $\mathrm{Tr}_{\mathrm{eq}}(\mathrm{Tr}_{\mathrm{vs}}(Q), \mathrm{Tr}_{\mathrm{vs}}(P))$.

Due to the EVA property, the verification of the key condition (6) gets simpler. Given $Y \in \pi_2(\mathrm{VSE}(P))$ and $Y' \in \pi_2(\mathrm{VSE}(P))$, $Y'_{\mathrm{v}} = Y_{\mathrm{v}}$ implies $Y' = Y$. By this observation, the translation-based method for weak/visible equivalence [25] implies the following.

Corollary 6.3. *Let P and Q be* smodels *programs with the EVA property. Then, $P \equiv_{\mathrm{vs}} Q$ iff $\mathrm{Tr}_{\mathrm{eq}}(\mathrm{Tr}_{\mathrm{vs}}(P), \mathrm{Tr}_{\mathrm{vs}}(Q))$ and $\mathrm{Tr}_{\mathrm{eq}}(\mathrm{Tr}_{\mathrm{vs}}(Q), \mathrm{Tr}_{\mathrm{vs}}(P))$ have no stable models.*

7 EXPERIMENTS

This section presents scalability experiments on automated verification of visible strong equivalence following the two step approach from Section 6. We recall that the approach relies on translation $\mathrm{Tr}_{\mathrm{vs}}(\cdot)$ to capture the sets of VSE-models of a program and translation $\mathrm{Tr}_{\mathrm{eq}}(\cdot, \cdot)$ to compare them. The translations are respectively implemented in the tools classic[6] and lpeq [25]. Using the translations, the visible strong equivalence $P \equiv_{\mathrm{vs}} Q$ of two programs P and Q is verified by checking that neither $\mathrm{Tr}_{\mathrm{eq}}(\mathrm{Tr}_{\mathrm{vs}}(P), \mathrm{Tr}_{\mathrm{vs}}(Q))$ nor $\mathrm{Tr}_{\mathrm{eq}}(\mathrm{Tr}_{\mathrm{vs}}(Q), \mathrm{Tr}_{\mathrm{vs}}(P))$ has stable models. This is done by running the state-of-the-art answer-set solver clasp [19] on Linux machines with Intel Xeon E5-4650 2.70-GHz processors.

In the course of the experiments, different representations of extended rules are verified to be equivalent, first for choice rules (2) and then for cardinality rules (3). The representations include the normalizations from Section 5 as implemented in the tool lp2normal [4, 24]. In this way, the formal correctness results from Section 5 and the quality of implementation are double checked for the explored parameter values. For cardinality rules, more advanced normalizations are also included from both lp2normal and another translation tool pbtranslate.[7]

Table 1 shows CPU times and the numbers of conflicts reported by clasp after verifying choice rules to be visibly strongly equivalent to their normalizations. The verification consists of checking unsatisfiability of the programs (a) $\mathrm{Tr}_{\mathrm{eq}}(\mathrm{Tr}_{\mathrm{vs}}(P), \mathrm{Tr}_{\mathrm{vs}}(Q))$ and (b) $\mathrm{Tr}_{\mathrm{eq}}(\mathrm{Tr}_{\mathrm{vs}}(Q), \mathrm{Tr}_{\mathrm{vs}}(P))$. These symmetric tasks amount to ensuring that each VSE-model of P corresponds to one of Q and vice versa. Each run was repeated five times and the results averaged. From the results it can be seen that verification performance scales tremendously well for choice rules. Only choice rules with hundreds of literals provide any significant challenge. Moreover, even huge choice rules with $n = 4{,}096$ literals can be proven visibly strongly equivalent to their normalizations in a matter of minutes and choice rules with $n = 16{,}384$ literals within half a day. This level of scalability is particularly positive in light of the fact that model enumeration based verification would be infeasible on this scale even for weak equivalence, given the exponentially growing number of models. In terms of time, verification performance scales superlinearly: doubling input size more than doubles run time. In terms of conflicts, scaling is linear and close to linear for the respective translation

[6]Published under http://research.ics.aalto.fi/software/asp/.
[7]Published under https://github.com/jbomanson/pbtranslate.

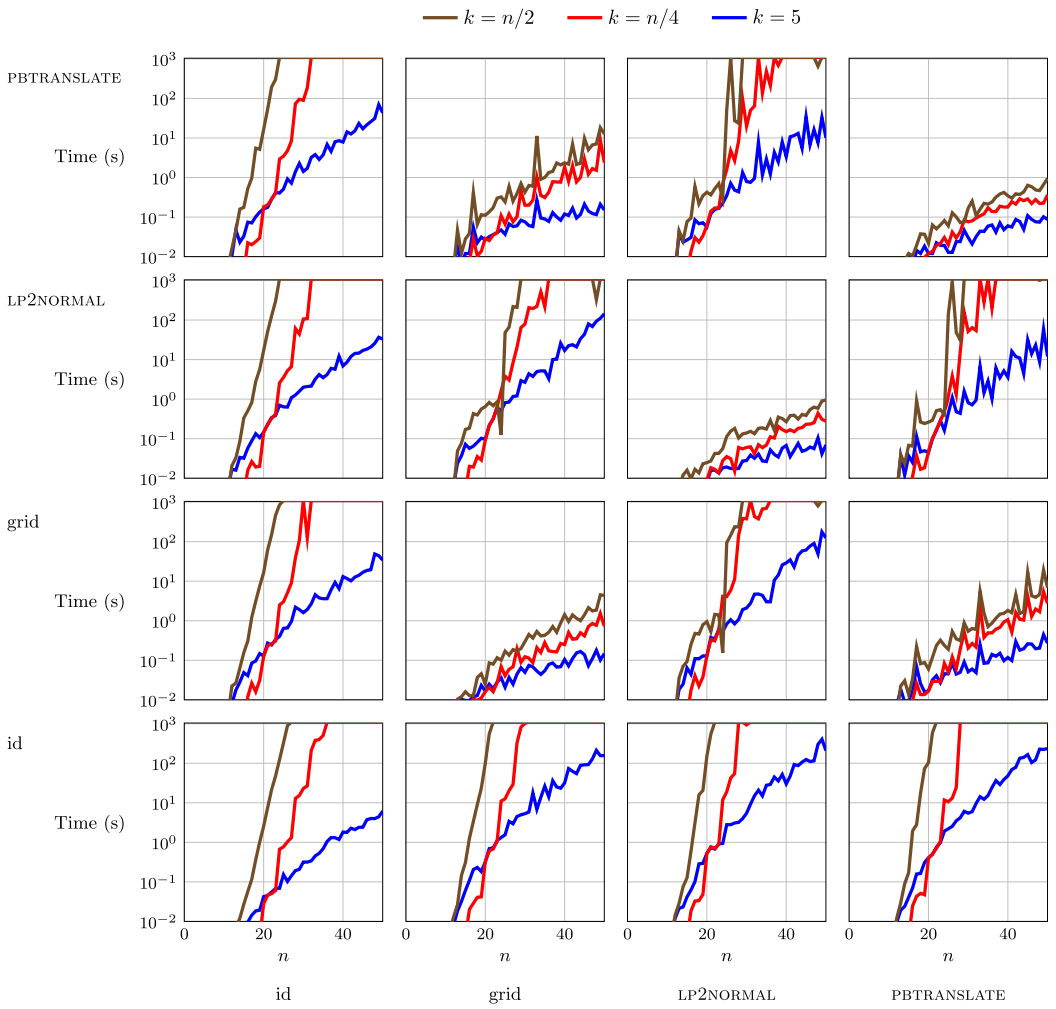

Fig. 2. The CPU time taken to verify visible strong equivalence $P \equiv_{\text{vs}} Q$ between programs representing cardinality rules C of different sizes as programs produced with the translations: id, grid, LP2NORMAL, and PBTRANSLATE. In the layout, rows and columns indicate which translations produced P and Q, respectively. For each pair P and Q, the verification process consists of unsatisfiability checks of programs of the form $\text{Tr}_{\text{eq}}(\text{Tr}_{\text{vs}}(P), \text{Tr}_{\text{vs}}(Q))$. The shown CPU times indicate averages over five identical checks of this type for each parameter combination (n, k).

directions (a) and (b). This suggests that the solver is able to prove unsatisfiability of (a) and (b) without performing much combinatorially challenging search. The results are asymmetric as direction (a) is easier, indicating that it is easier to verify that VSE-models of a choice rule correspond to VSE models of its normalization than vice versa.

Figures 2 and 3 illustrate the CPU time consumed when verifying different cardinality rule representations against one another. Between the illustrations, Figure 2 gives regular plots on cardinality rules with $n \leq 50$ input literals and bounds $k \in \{5, n/4, n/2\}$, whereas Figure 3 gives contour plots on cardinality rules with $n \leq 30$ literals and bounds $k \leq n$. In the results, the label id stands for the identity translation, grid for the counting grid translation from Reference [24] as

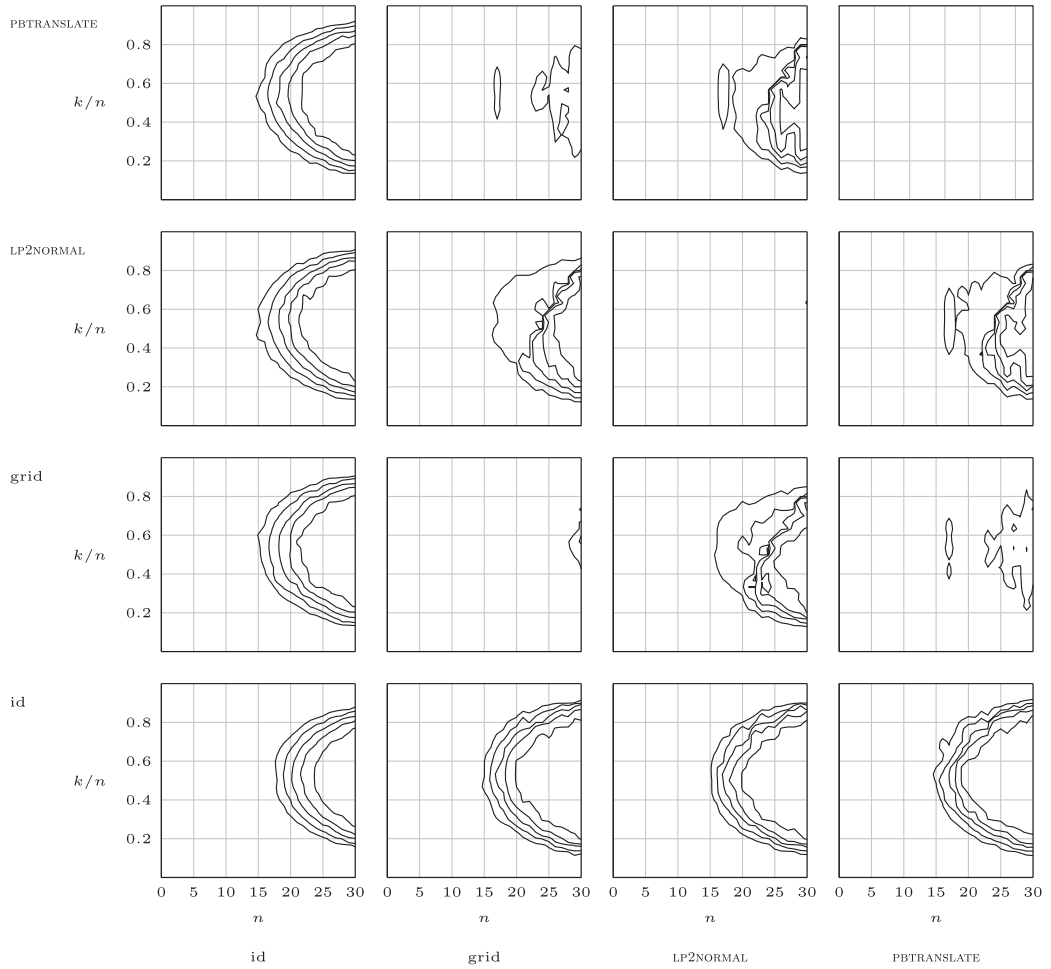

Fig. 3. The CPU time taken to verify visible strong equivalence $P \equiv_{\text{vs}} Q$ as in Figure 2, but illustrated here with contour plots and a different range of parameters. Namely, the horizontal and vertical dimensions in these contour plots reflect the number n and fraction k/n, where n and k are the number of input literals and the bound of each translated cardinality rule, respectively. Moreover, the contour lines mark CPU times of 0.25, 1, 4, 16, 64 s, from left to right. Essentially, the larger the "mountain," the higher the CPU time. All illustrated times have been averaged over five identical runs for each parameter combination $(n, k/n)$. In each plot, parameter combinations taking less than 0.25 seconds form the largest area, outside the mountain. Furthermore, parameter combinations taking more than 64 seconds form a plateau on top of the mountain of several of the plots. These plateaus are a reflection of the used time limit; indeed, given a higher limit and more contour lines, the mountains would grow higher.

analyzed in Section 5, LP2NORMAL for the default translation of LP2NORMAL, and PBTRANSLATE for the default translation of PBTRANSLATE. The default for LP2NORMAL heuristically and recursively mixes a number of translations [4]. The default for PBTRANSLATE uses small precomputed sorters and Batcher's odd-even merge sorters [2] with structure chosen via dynamic programming. More precise descriptions of these translations is out of the scope of this article. Indeed, the translations are complicated and involve miscellaneous implementation details not presented in research

papers. For this reason, they are particularly interesting in these experiments as their inclusion helps obtain a picture of verification performance in a realistic, practical setting.

The results reveal that verification difficulty clearly increases as the number of literals increases and as the bound k approaches $n/2$, i.e., as the fraction k/n approaches 0.5. Moreover, it is generally tougher to verify that the VSE-models of a cardinality rule match VSE-models of its normalizations than vice versa. Beyond these trends, the results vary significantly between the different pairs of compared cardinality rule representations. The most challenging verification tasks are those involving the cardinality rule itself, as "produced" by the identity translation id. These cases correspond to the column of plots on the left and the row of plots on the bottom, which are notably uniform. Among these cases, the one where id is verified against itself is the easiest, but only by a small margin.

The overall easiest verification tasks are those between pairs of actual normalizations, which exclude the identity translation id. These cases are represented by the square of nine plots on the top right. The improvement in performance over the cases with id is tremendous. The cases on the diagonal where normalizations are verified against themselves are trivial. The cases with grid against PBTRANSLATE are solved withing fractions of a second for nearly all parameter combinations in both verification directions. The cases with LP2NORMAL against the other two normalizations are likewise easier than the cases with id, but curiously more challenging than the other cases comparing normalizations against normalizations. Additional investigation into the matter revealed that the difficulty largely disappears if LP2NORMAL is configured to behave a bit more similarly to PBTRANSLATE by avoiding a defining feature of Parberry's sorting networks [37] that is implemented and used by default in LP2NORMAL, but not PBTRANSLATE. This is in line with the intuition that structural similarity between normalizations eases verifiability.

In general, these results on cardinality rules speak strongly in favor of a practical verification strategy for normalization schemes, in which schemes are verified against some already trusted scheme, as opposed to the extended rule being normalized. Moreover, in a setting where a number of normalization schemes are to be verified, the overall computation time can be optimized by pairing up selected representations for comparison against one another such that visible strong equivalence of all of the representations is established via transitivity while some difficult cases are hopefully avoided. For example, in this experimental setting, the overall verification time would be the lowest if the consecutive pairs of normalizations in the following order were verified: id, grid, PBTRANSLATE, LP2NORMAL. This would prove the correctness of all of the normalizations for the considered parameters while avoiding all but one of the most difficult cases.

8 RELATED WORK

The general framework of *correspondence frames* [16] expresses a broad class of equivalence checking problems. The framework is parameterized by a class of context programs and an arbitrary comparison relation. By appropriate choices of these parameters, the framework yields the problems of checking strong equivalence, uniform equivalence, and relativized strong equivalence, for example. Due to the generality of the framework, even VSE checking can be cast as an instance of the framework, as shown in Section 3.5. Despite this connection between correspondence frames and VSE, the results in the study of Eiter et al. [16] are focused on another special case of the framework. That special case is called relativized strong equivalence with projection, which is parameterized by a set A like its non-projective variant, but also by a further parameter set B used for answer-set projection. This relation is similar to VSE, and is implied by VSE when given appropriate parameters, but treats projections differently. Namely, the relation does not require a one-to-one correspondence of answer sets, but instead requires for programs P and Q in a context R with $\mathrm{At}(R) \subseteq A$ that $\mathrm{SM}(P \cup R)\,|_B = \mathrm{SM}(Q \cup R)\,|_B$. Thus, the requirement is independent of the

numbers of answer set copies in contrast to VSE, which does depend on the numbers. This dependence is meaningful, for example, when a disjunctive rule $a \mid b$, or the analogous normal rules $a \leftarrow {\sim}b$ and $a \leftarrow {\sim}b$, with $a, b \in A \setminus B$, are added to a program with no other references to a or b. Indeed, such an addition preserves relativized strong equivalence with projection, but breaks VSE. Due to this difference, VSE reflects standard ASP solving technology more strictly, as solvers such as CLASP do print answer set copies by default. While CLASP also gives the user the option to project answer sets, preprocessing transformations that preserve VSE also preserve that choice with the user. However, transformations that preserve only relativized strong equivalence with projection but not VSE require the use of such an option in order to preserve printed answer sets, thus taking the choice away from the user. The differences to VSE are also particularly evident in certain probabilistic applications [1, 3] and in corner cases: setting $B = \emptyset$ means checking whether answer sets exist on an equal basis whereas for $\equiv_{\mathrm{vs}}$, setting $\mathrm{At_v}(P) = \emptyset$ implies model counting. It is also worth pointing out that the two approaches coincide in cases where answer set copies can not possibly arise as an issue, such as in the case of SMODELS programs possessing the EVA property as used in Section 6. That is, in these cases the stronger notion of VSE can be established via relativized strong equivalence with projection. In the general case, however, knowledge of the visible strong equivalence $P \equiv_{\mathrm{vs}} Q$ brings more information about P and Q than only knowing that P and Q are strongly equivalent relative to A and projected onto B (subject to $\mathrm{At_v}(P) = A = B = \mathrm{At_v}(Q)$).

The model-based characterization of relativized strong equivalence (RSE) from [45] is closely related to the characterization in this article. The respective definitions of A-SE-models $\langle X, Y\rangle$ and VSE-models $\langle X, Y\rangle$ are close to each other: The main difference is in VSE-models having a more streamlined X-part. As recalled in Definition 2.12, A-SE-models distinguish *total* cases $X = Y$ from *non-total* cases $X \subset Y$ and project the latter onto a set A of (visible) atoms. In contrast, VSE-models from Definition 3.3 make the roles of X and Y more analogous by not projecting either of them. This reflects the idea that $\mathrm{At_h}(P)$-minimal models act as natural representatives of classical models in the presence of hidden atoms. The lack of projection in VSE-models is accounted for by concealing their hidden parts when checking the correspondence (6). This difference stems from how RSE and VSE insist on the stable models of $P \cup R$ and $Q \cup R$ to match via $=$ and $\overset{\mathrm{v}}{=}$, respectively.

As regards concrete practical applications to program transformations, the preservation of strong and *uniform equivalence* [12] under various transformations is addressed in Reference [13]. Therein, a number of rule-level transformations from the literature, such as TAUT, RED$^-$, NONMIN, WGPPE, and CONTRA, are considered. The fruitful use of the original strong and uniform equivalence notions in this context is enabled by the fact that these simple transformations do not involve auxiliary atoms. Transformations that do involve auxiliary atoms can be checked for the preservation of RSE with projection via general translations into Quantified Boolean Formulas (QBFs) [42]. In being a translation-based verification approach, this is similar to the approach from Section 6, which reduces VSE checking given the EVA property into a pair of ASP unsatisfiability checks. The approaches differ in the computational complexities of the equivalence checking problems concerned: whereas RSE checking with projection is Π_4^P-complete [42] and is thus out of the reach of ASP solvers, VSE checking given EVA is instead within their reach. Despite the high complexity of RSE checking, the problem can be tackled with the mentioned translation into QBF, which is implemented in the *correspondence checking tool* CCT [34] that is designed to be coupled with a QBF solver. Indeed, practical use of the approach has been demonstrated in verifying student solutions to logic programming assignments [35].

9 CONCLUSIONS

In this article, we harness the qualities of strong equivalence [29] and visible equivalence [23] and propose *visible strong equivalence* as a generalization of both. An associated characterization

based on VSE-models suggests that the construction is successful. We believe that VSE will have both theoretical and practical value due to the provided formal results and the fact that it models encapsulation the same way standard answer-set solvers do. Indeed, as illustrated in Section 5, VSE can be used to prove translations introducing new auxiliary atoms correct in a very strict sense (formalized by $\equiv_{\mathrm{v}}$ and $\overset{\mathrm{v}}{=}$ in the article). Moreover, the implementation described in Section 6 provides a basis for the systematic verification and debugging of rule-level translators. This is successfully showcased in scalability experiments in Section 7. The findings support a verification strategy where outputs of translations are checked against one another as opposed to their inputs, whenever feasible. The implementation relies on the so-called EVA property, which is a technical restriction necessary to trade off computational complexity so that a polynomial translation is achievable and answer-set solvers themselves can be used for checking the property. The outputs of grounders, such as LPARSE and GRINGO, have this property by default unless enough atoms are intentionally hidden to create a hidden program part with non-deterministic behavior that is not apparent from the visible part. In the experiments, for instance, this limitation posed no issues.

As regards future work, there is a call for a complexity analysis of $\equiv_{\mathrm{vs}}$ in terms of #P-oracles. Moreover, the close interconnection of *uniform equivalence* [12] and strong equivalence, as studied in Reference [14], suggests an analogous extension to be worth considering for our approach using sets of facts rather than arbitrary programs as contexts.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for thorough and informative comments.

REFERENCES

[1] A. Abdelbar. 1998. An algorithm for finding MAPs for belief networks through cost-based abduction. *Artif. Intell.* 104, 1–2 (1998), 331–338.

[2] K. E. Batcher. 1968. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference*. ACM, 307–314.

[3] H. Beaver and I. Niemelä. 1999. Finding MAPs for belief networks using rule-based constraint programming. *Arpakannus (Special Issue on Networks)* 1 (1999), 8–11.

[4] J. Bomanson. 2017. lp2normal - A normalization tool for extended logic programs. In *Proceedings of the 14th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*, M. Balduccini and T. Janhunen (Eds.), Lecture Notes in Computer Science, Vol. 10377. Springer, 222–228. DOI: https://doi.org/10.1007/978-3-319-61660-5_20

[5] J. Bomanson, M. Gebser, and T. Janhunen. 2014. Improving the normalization of weight rules in answer set programs. In *Proceedings of the 14th European Conference on Logics in Artificial Intelligence (JELIA'14) (LNAI)*, Vol. 8761. Springer, 166–180.

[6] J. Bomanson and T. Janhunen. 2013. Normalizing cardinality rules using merging and sorting constructions. In *Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*, Lecture Notes in Computer Science, Vol. 8148. Springer, 187–199.

[7] J. Bomanson, T. Janhunen, and A. Weinzierl. 2019. Enhancing lazy grounding with lazy normalization in answer-set programming. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*. AAAI Press, 2694–2702. DOI: https://doi.org/10.1609/aaai.v33i01.33012694

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

[9] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub. 2012. ASP-CORE-2 Input Language Format.

[10] M. Dao-Tran, T. Eiter, M. Fink, and T. Krennwallner. 2009. Modular nonmonotonic logic programming revisited. In *Proceedings of the 25th International Conference on Logic Programming (ICLP'09)*, Lecture Notes in Computer Science, Vol. 5649. Springer, 145–159.

[11] N. Eén and N. Sörensson. 2006. Translating Pseudo-Boolean constraints into SAT. *J. Satis. Bool. Model. Comput.* 2, 1–4 (2006), 1–26.

[12] T. Eiter and M. Fink. 2003. Uniform equivalence of logic programs under the stable model semantics. In *Proceedings of the 19th International Conference on Logic Programming (ICLP'03)*, Lecture Notes in Computer Science, Vol. 2916. Springer, 224–238.
[13] T. Eiter, M. Fink, H. Tompits, and S. Woltran. 2004. Simplifying logic programs under uniform and strong equivalence. In *Proceedings of the 7th International Conference Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, Lecture Notes in Computer Science, Vol. 2923. Springer, 87–99.
[14] T. Eiter, M. Fink, and S. Woltran. 2007. Semantical characterizations and complexity of equivalences in answer set programming. *ACM Trans. Comput. Logic* 8, 3 (2007), Article 17.
[15] T. Eiter and G. Gottlob. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.* 15, 3–4 (1995), 289–323.
[16] T. Eiter, H. Tompits, and S. Woltran. 2005. On solution correspondences in answer-set programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*. 97–102.
[17] M. Fink. 2011. A general framework for equivalences in Answer-Set Programming by countermodels in the logic of Here-and-There. *Theory Pract. Logic Program.* 11, 2–3 (2011), 171–202.
[18] M. Gebser, R. Kaminski, A. König, and T. Schaub. 2011. Advances in *gringo* Series 3. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, Lecture Notes in Computer Science, Vol. 6645. Springer, 345–351.
[19] M. Gebser, B. Kaufmann, and T. Schaub. 2012. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.* 187 (2012), 52–89.
[20] M. Gelfond and V. Lifschitz. 1988. The stable model semantics for logic programming. In *Proceedings of the 6th International Conference on Logic Programming (ICLP'88)*. 1070–1080.
[21] M. Gelfond and V. Lifschitz. 1990. Logic programs with classical negation. In *Proceedings of the 7th International Conference on Logic Programming (ICLP'90)*. 579–597.
[22] M. Gelfond and V. Lifschitz. 1991. Classical negation in logic programs and disjunctive databases. *New Gener. Comput.* 9 (1991), 365–385.
[23] T. Janhunen. 2006. Some (in)translatability results for normal logic programs and propositional theories. *J. Appl. Non-Classic. Logics* 16, 1–2 (June 2006), 35–86.
[24] T. Janhunen and I. Niemelä. 2011. Compact translations of non-disjunctive answer set programs to propositional clauses. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning—Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, Lecture Notes in Computer Science, Vol. 6565. Springer, 111–130.
[25] T. Janhunen and E. Oikarinen. 2007. Automated verification of weak equivalence within the Smodels system. *Theory Pract. Logic Program.* 7, 6 (2007), 697–744.
[26] T. Janhunen, E. Oikarinen, H. Tompits, and S. Woltran. 2009. Modularity aspects of disjunctive stable models. *J. Artif. Intell. Res.* 35 (2009), 813–857.
[27] V. Lifschitz. 1985. Computing circumscription. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 121–127.
[28] V. Lifschitz. 1999. Answer set planning. In *Proceedings of the 16th International Conference on Logic Programming (ICLP'99)*. 23–37.
[29] V. Lifschitz, D. Pearce, and A. Valverde. 2001. Strongly equivalent logic programs. *ACM Trans. Comput. Logic* 2, 4 (2001), 526–541.
[30] V. Lifschitz and H. Turner. 1994. Splitting a logic program. In *Proceedings of the 11th International Conference on Logic Programming (ICLP'94)*. MIT Press, 23–37.
[31] L. Liu and M. Truszczynski. 2006. Properties and applications of programs with monotone and convex constraints. *J. Artif. Intell. Res.* 27 (2006), 299–334.
[32] J. McCarthy. 1986. Applications of circumscription to formalizing commonsense knowledge. *Artif. Intell.* 28 (1986), 89–116.
[33] I. Niemelä. 2008. Answer set programming without unstratified negation. In *Proceedings of the 24th International Conference on Logic Programming (ICLP'08)*, M. G. de la Banda and E. Pontelli (Eds.), Lecture Notes in Computer Science, Vol. 5366. Springer, 88–92. DOI: https://doi.org/10.1007/978-3-540-89982-2_15
[34] Johannes Oetsch, Martina Seidl, Hans Tompits, and Stefan Woltran. 2006. ccT: A tool for checking advanced correspondence problems in answer-set programming. In *Proceedings of the FLOC-Workshop on Search and Logic (LaSh'06)*.
[35] Johannes Oetsch, Martina Seidl, Hans Tompits, and Stefan Woltran. 2009. ccT on stage: Generalised uniform equivalence testing for verifying student assignment solutions. In *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, Lecture Notes in Computer Science, Vol. 5753. Springer, 382–395.
[36] E. Oikarinen and T. Janhunen. 2008. Achieving compositionality of the stable model semantics for smodels programs. *Theory Pract. Logic Program.* 8, 5–6 (2008), 717–761.

[37] I. Parberry. 1991. On the computational complexity of optimal sorting network verification. In *Proceedings of Parallel Architectures and Languages Europe (PARLE'91), Volume I: Parallel Architectures and Algorithms*, E. H. L. Aarts, J. van Leeuwen, and M. Rem (Eds.), Lecture Notes in Computer Science, Vol. 505. Springer, 252–269. DOI: https://doi.org/10.1007/BFb0035109

[38] D. Pearce. 2006. Equilibrium logic. *Ann. Math. Artif. Intell.* 47, 1–2 (2006), 3–41.

[39] A. Polleres, M. Frühstück, G. Schenner, and G. Friedrich. 2013. Debugging non-ground ASP programs with choice rules, cardinality and weight constraints. In *Proceedings of the 12th International Conference Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*, P. Cabalar and Tran Cao Son (Eds.), Lecture Notes in Computer Science, Vol. 8148. 452–464. DOI: https://doi.org/10.1007/978-3-642-40564-8_45

[40] J. Pührer and H. Tompits. 2009. Casting away disjunction and negation under a generalisation of strong equivalence with projection. In *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*. 264–276.

[41] P. Simons, I. Niemelä, and T. Soininen. 2002. Extending and implementing the stable model semantics. *Artif. Intell.* 138, 1–2 (2002), 181–234.

[42] Hans Tompits and Stefan Woltran. 2005. Towards implementations for advanced equivalence checking in answer-set programming. In *Proceedings of the 21st International Conference on Logic Programming (ICLP'05)*, M. Gabbrielli and G. Gupta (Eds.), Lecture Notes in Computer Science, Vol. 3668. Springer, 189–203. DOI: https://doi.org/10.1007/11562931_16

[43] H. Turner. 2001. Strong equivalence for logic programs and default theories (made easy). In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01) ()*, Lecture Notes in Computer Science, Vol. 2173. Springer, 81–92.

[44] H. Turner. 2003. Strong equivalence made easy: Nested expressions and weight constraints. *Theory Pract. Logic Program.* 3, 4–5 (2003), 609–622.

[45] S. Woltran. 2004. Characterizations for relativized notions of equivalence in answer set programming. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA'04)*, Vol. 3229. Springer, 161–173.

[46] S. Woltran. 2008. A common view on strong, uniform, and other notions of equivalence in answer-set programming. *Theory Pract. Logic Program.* 8, 2 (2008), 217–234.

Received June 2019; revised March 2020; accepted July 2020