Concrete Results on Abstract Rules

Markus Krötzsch, Despoina Magka, and Ian Horrocks

Department of Computer Science, University of Oxford, UK

Abstract. There are many different notions of "rule" in the literature. A key feature and main intuition of any such notion is that rules can be "applied" to derive conclusions from certain premises. More formally, a rule is viewed as a function that, when invoked on a set of known facts, can produce new facts. In this paper, we show that this extreme simplification is still sufficient to obtain a number of useful results in concrete cases. We define abstract rules as a certain kind of functions, provide them with a semantics in terms of (abstract) stable models, and explain how concrete normal logic programming rules can be viewed as abstract rules in a variety of ways. We further analyse dependencies between abstract rules to recognise classes of logic programs for which stable models are guaranteed to be unique.

1 Introduction

A large variety of different types of "rules" are considered in logic programming, knowledge representation, production rule systems, and databases. While many rules have a common background in predicate logic, there are still important differences between, say, normal logic programs [13], existential rules [3], and database dependencies [1]. It is, however, highly desirable to transfer concrete results and insights between these domains.

This goal is best illustrated by considering a concrete example. In a recent publication, the authors analyse *nonmonotonic existential rules* under a stable model semantics [14]. The work identifies syntactic conditions to guarantee finiteness and uniqueness of stable models, and shows how this can be applied to improve the performance of reasoning over real-world data. To fully exploit these ideas, this approach can be further extended in at least two ways: (1) other types of logic rules, e.g., normal logic programs, could be considered; (2) extend the scope of the approach for relevant special forms of programs, e.g., for equality and datatype reasoning.

In terms of (2), the authors already extended their results by considering integrity constraints [14]. Unfortunately, even this relatively small change required laborious extensions of all previous correctness proofs. While the structure of arguments remains similar, each individual step now needs to take constraints into account. Following this pedestrian approach, repeated effort is required for each modification in the underlying language. With the continued elaboration of rule-based languages, important ideas often remain confined to one sub-area and are rather reinvented than being transferred. For instance, the notion of rule dependency that was extended to nonmonotonic existential rules in [14] has first been proposed for conceptual graph rules in 2004 [2] and rediscovered for databases in 2008 [4].

To address this issue, we propose to take a more abstract view on "rules" that can be instantiated in many different cases. Our main intuition is that many kinds of rules can be "applied" in certain sense to derive conclusions from premises. We formalise this by viewing a rule as a function that, when invoked on a set of known facts, can produce new facts. This is a rather natural view on rules. Our main contribution is to show that this extreme simplification is still sufficient to derive interesting results that are easy to instantiate in concrete cases. Our contributions are organised as follows.

- We define *abstract rules* and provide them with a semantics in terms of *abstract stable models* that agrees with the standard notion of stable model semantics in concrete cases (Section 2).
- We present *operations for constructing abstract rules* (Section 3) and establish several strong equivalence results to show that these operations preserve our semantics (Section 4).
- We reformulate the condition of *R-stratification* from [14] for abstract rules (Section 5) and show that this condition leads to a *unique stable model* even in the abstract case (Section 6).
- We apply our results to *equality reasoning* in normal logic programs and propose a new approach for stratifying programs with equality using constraints (Section 7).

Proofs that are not included here can be found in an accompanying report [10].

2 Abstract Rules and Models

We consider a countable set B of basic logical expressions, which we think of as facts that might be true or false in a given situation; subsets $F \subseteq \mathsf{B}$ will therefore be called *sets of (abstract) facts*. B contains a distinguished element $\bot$, which denotes a contradiction.

Definition 1. *An* abstract rule *r is a function $r : 2^{\mathsf{B}} \to 2^{\mathsf{B}}$ with the following properties.*

- *r is* extensive*: $r(F) \supseteq F$ for every $F \subseteq \mathsf{B}$*
- *r is* compact *(or* finitary*): for every $F \subseteq \mathsf{B}$ and $f \in r(F)$, there are finite sets $B^+, B^- \subseteq \mathsf{B}$ with $B^+ \subseteq F$ and $B^- \cap F = \emptyset$, such that $f \in r(F')$ for every set F' with $B^+ \subseteq F'$ and $B^- \cap F' = \emptyset$.*

If $f \in r(F)$, we say that r derives *f from F. A pair of minimal sets $\langle B^+, B^- \rangle$ that witnesses the compactness property for the derivation $f \in r(F)$ is called an* abstract body *of r with respect to f and F.*

An abstract program *is a countable set of abstract rules, denoted P, possibly with subscripts or primes. Uncountable rule sets are not considered herein.*

A rule is *monotone* if $F_1 \subseteq F_2$ implies $r(F_1) \subseteq r(F_2)$ for all sets of facts $F_1, F_2 \subseteq \mathsf{B}$, but this is not required by our definition. For monotone rules, every derivation has a body $\langle B^+, B^- \rangle$ with $B^- = \emptyset$, as one would expect.

Example 1. A propositional logic programming rule is an expression of form $H \leftarrow B_1, \ldots, B_n, \mathbf{not}\, B_{n+1}, \ldots, \mathbf{not}\, B_m$, where $B_1, \ldots, B_m$ and H are propositional letters. It can be viewed as an abstract rule r: let B be the set of propositional letters and define $r(F) := F \cup \{H\}$ if $B_1, \ldots, B_n \in F$ and $B_{n+1}, \ldots, B_m \notin F$, and $r(F) := F$ otherwise. An abstract body is given by the sets $B^+ = \{B_1, \ldots, B_n\}$ and $B^- = \{B_{n+1}, \ldots, B_m\}$.

Note that this approach does not define a one-to-one correspondence of concrete rules and abstract rules. For example, the rules $A \leftarrow A$ or $A \leftarrow A, B$ both lead to the same abstract rule with $r(F) = F$. Thus our abstraction cannot capture any logic programming semantics where the presence of such rules is relevant.

Example 2. A normal logic programming rule is an expression of form $H \leftarrow B_1, \ldots, B_n$, **not** $B_{n+1}, \ldots,$ **not** B_m, where $B_1, \ldots, B_m$ and H are predicate-logic atoms over some logical signature. It can be viewed as an abstract rule r: let B be the set of all ground atoms over the signature (the so-called *Herbrand base*), let G_r be the set of all ground instantiations of r, and define $r(F) := \bigcup_{r_g \in G_r} r_g(F)$, where $r_g(F)$ is defined as for propositional rules in Example 1. An abstract body for a ground fact H_g is obtained as the abstract body for any ground instance r_g that can be used to derive H_g from F.

Example 2 illustrates that abstract bodies may not be unique, since several ground instantiations of a logic programming rule may have the same head but different bodies.

Example 3. Consider the Herbrand base $\mathsf{B} = \{n(a), n(s(a)), n(s(s(a))), \ldots\}$, and let $\mathsf{B}^{\geq 1}$ denote the set $\mathsf{B} \setminus \{n(a)\}$. Consider the following functions:

$$r_1(F) := F \cup \{n(a)\} \quad \text{if } \mathsf{B}^{\geq 1} \subseteq F; \qquad r_1(F) := F \quad \text{otherwise} \tag{1}$$

$$r_2(F) := F \cup \{n(a)\} \quad \text{if } \mathsf{B}^{\geq 1} \not\subseteq F; \qquad r_2(F) := F \quad \text{otherwise} \tag{2}$$

$$r_3(F) := F \cup \{n(a)\} \quad \text{if } F \text{ is finite}; \qquad r_3(F) := F \quad \text{otherwise} \tag{3}$$

Each of these functions is extensive. Function r_1 is not an abstract rule: it is monotone but depends on an infinite set $\mathsf{B}^{\geq 1}$ of premises, hence is not compact. Function r_2 is an abstract rule: for any fact $f \in \mathsf{B}^{\geq 1}$ such that $f \notin F$, the sets $B^+ = \emptyset$ and $B^- = \{f\}$ form a body of the derivation $n(a) \in r_2(F)$. A concrete representation of this rule along the lines of Example 2 is $n(a) \leftarrow$ **not** $n(s(X))$, although this may not be syntactically allowed in all logic programming approaches, since X occurs in negated atoms only. Function r_3 is not an abstract rule: it is nonmonotonic but there is no finite negative body B^- for any derivation (for finite F, we would get $B^+ = \emptyset$ and $B^- = \mathsf{B} \setminus F$).

We can define the consequence operator T_P for abstract rules as usual.

Definition 2. *For a set of rules P and a set of facts F, we define $T_P(F) := \bigcup_{r \in P} r(F)$. Moreover, we set $T_P^0(F) := F$, $T_P^{i+1}(F) := T_P(T_P^i(F))$, and $T_P^\infty(F) := \bigcup_{i \geq 0} T_P^i(F)$.*

Since we do not assume rules to be monotone, different orders of rule applications might lead to very different sets of derived sets of facts, and in particular $T_P^\infty(F)$ may not capture the desired semantics of the program. The next definition describes types of derived sets of facts that are more suitable for defining the semantics of abstract rules.

Definition 3. *Consider an abstract program P and a set of facts F_0. A set of facts F is* well-supported *for P and F_0 if there is a well-founded partial order $\prec$ on F such that, for every fact $f \in F \setminus F_0$, there is a rule $r_f \in P$ with body $\langle B_f^+, B_f^- \rangle$ for f and F, and $f' \prec f$ for all $f' \in B_f^+$. We assume that the choice of r_f and $\langle B_f^+, B_f^- \rangle$ is part of each well-supported set (there might be other choices, leading to other well-supported sets).*

A set of facts F is an abstract model *for a set of rules P and a set of input facts F_0 if $\bot \notin F$, $F_0 \subseteq F$, and $r(F) \subseteq F$ for every rule $r \in P$. An* abstract stable model *is a well-supported abstract model.*

For the case of normal logic programming rules, our definition of well-supported model agrees with that of Fages [7]. Our terminology is justified by Fages's result that well-supported models are exactly the (classical) stable models [7, Theorem 2.1]. We obtain the following theorem as a corollary.

Theorem 1. *The abstract stable models of a normal logic program P, viewed as a set of abstract rules as in Example 2, are exactly the classical stable models of P.*

Like in the classical case, T_P can be used to compute models, which, however, may not be well-supported.

Proposition 1. *$T_P^\infty(F)$ is an abstract model for P and F.*

Proof. Suppose for a contradiction that the claim does not hold. Then there is a rule $r \in P$ and a fact $f \in r(T_P^\infty(F))$ with $f \notin T_P^\infty(F)$. By compactness, there is a finite set $F' \subseteq T_P^\infty(F)$ such that $f \in r(F'')$ for every $F' \subseteq F'' \subseteq T_P^\infty(F)$. By construction of $T_P^\infty(F)$, there is some i such that $F' \subseteq T_P^i(F)$. But then $f \in r(T_P^i(F))$ by compactness, and hence $f \in T_P^{i+1}(F) \subseteq T_P^\infty(F)$—a contradiction. □

The well-founded order $\prec$ in Definition 3 leaves a lot of flexibility for establishing that a set is well-supported. Due to compactness, however, it is generally enough to order facts by a finite rank that can be expressed as a natural number.

Proposition 2. *If F is well-supported for P and F_0, then this is witnessed by an order $\prec$ such that $\langle F, \prec\rangle$ has an order-preserving injection into the natural numbers $\langle \mathbb{N}, <\rangle$.*

Proof. Since F is well-supported, there is an order $\prec_0$ as in Definition 3. For each fact $f \in F \setminus F_0$, there is a rule $r_f \in P$ and a body $\langle B_f^+, B_f^-\rangle$ as in Definition 3. The order $\prec_1$ on F is the transitive closure of the set $\{f' \prec_1 f \mid f' \in B_f^+\}$. Clearly, $\prec_1 \subseteq \prec_0$, so $\prec_1$ is well-founded. By definition, $\prec_1$ can be used to show that F is well-supported, using the same choice of rules r_f and bodies B_f^+ as for $\prec_0$.

We claim that for every fact $f \in F$, the set $f{\downarrow} := \{f' \mid f' \prec_1 f\}$ is finite ($*$). Indeed, by construction, $f{\downarrow} = \bigcup_{f' \in B_f^+} f'{\downarrow}$. The claim follows by well-founded induction: if $f'{\downarrow}$ is finite for all $f' \prec_1 f$, then $f'{\downarrow}$ is finite for all $f' \in B_f^+$, and thus $f{\downarrow}$ is finite, too.

To construct a total well-founded order $\prec$ as required in the claim, we re-order the elements of F as a sequence. Since F is countable, there is an injective mapping $\iota : F \to \mathbb{N}$ from F to natural numbers. We recursively construct a (possibly infinite) sequence $f_1, f_2, \ldots$ of facts from F as follows. To select f_i, consider the set of $\prec_1$-minimal elements M_i in the set $F \setminus \{f_1, \ldots, f_{i-1}\}$. If $M_i = \emptyset$ then there is no f_i and the construction terminates with a finite sequence. Otherwise, define $f_i \in M_i$ to be the ι-smallest element of M_i, i.e., $\iota(f_i) \leq \iota(f)$ for all $f \in M_i$.

We claim that the constructed sequence $S = \{f_1, f_2, \ldots\}$ contains exactly the elements of F. For a contradiction, suppose that there is an element $f \in F \setminus S$. By ($*$), the set $\{f\} \cup f{\downarrow}$ is finite. Consider the set $M := \{f\} \cup f{\downarrow} \setminus S$. By our assumptions, $f \in M$, so M is finite but not empty. Thus there is a $\prec_1$-minimal element f' in M, which is also a $\prec_1$-minimal element of $F \setminus S$. As there are only finitely many elements in F that are $\prec_1$-smaller than f', there is a finite index j such that f' is a $\prec_1$-minimal element of

$F \setminus \{f_1, \ldots, f_j\}$. There are at most $\iota(f') - 1$ many elements that can be added to S after f_j, before f' must also be added. Thus $f' \in S$, which contradicts our assumptions.

We define the order $\prec$ on F by setting $f_i \prec f_j$ for all $i < j$. This makes $\prec$ a suborder of $\langle \mathbb{N}, < \rangle$, and thus well-founded. Moreover, $\prec_1 \subseteq \prec$, so $\prec$ can be used to show that F is well-supported. □

3 Constructing Abstract Rules

The examples given so far mainly show that abstract rules can capture normal logic programs. In this section, we show that they are significantly more general, even on a base set of facts B that is the Herbrand base of a predicate logic signature. For this purpose, we introduce various operations for constructing new abstract rules from existing ones, and show that these operation preserve stable model semantics (Section 4). The basic operations we consider are union, composition and saturation of rules.

Definition 4. *Let P be a program. The* union $\bigcup P$ *of P is defined by setting $(\bigcup P)(F) := \bigcup_{r \in P} r(F)$ if $P \neq \emptyset$. For $P = \emptyset$, we define $(\bigcup \emptyset)(F) := F$.*

The intersection $\bigcap P$ *of P is defined as $(\bigcap P)(F) := \bigcap_{r \in P} r(F)$ if $P \neq \emptyset$. For $P = \emptyset$, we define $(\bigcap \emptyset)(F) := \mathsf{B}$.*

Example 4. The abstract rule induced by a normal logic programming rule as in Example 2 is the infinite union of the abstract rules obtained from its ground instantiations. Likewise, the one-step T_P operator of Definition 2 is the abstract rule $\bigcup P$. Intersections of rules are the abstract counterpart to conjunctions in rule bodies. For example, the intersection of the rules $q \leftarrow p_1$ and $q \leftarrow p_2, \mathbf{not}\ p_3$ can be expressed as $q \leftarrow p_1, p_2, \mathbf{not}\ p_3$.

Intersections of abstract rules do not always result in abstract rules. For example, the function in (1), which is not compact, can be viewed as the intersection of the infinite set of all rules $n(a) \leftarrow n(s^i(a))$ with $i \geq 1$. However, abstract rules are closed under infinite unions and finite intersections, as shown next.

Theorem 2. *The union $\bigcup P$ of an abstract program P is an abstract rule. If P is finite, then the intersection $\bigcap P$ is also an abstract rule.*

Proof. First consider $\bigcup P$. For all $r \in P$ we find $F \subseteq r(F)$ by extensiveness; hence $F \subseteq \bigcup_{r \in P} r(F) = \bigcup P(F)$ and $\bigcup P$ is extensive. If $P = \emptyset$ then $(\bigcup P)(F) = F$, so for any derivation $f \in (\bigcup P)(F)$ the sets $B^+ = \{f\}$ and $B^- = \emptyset$ show compactness of $(\bigcup P)$. If $P \neq \emptyset$ then, for any fact $f \in (\bigcup P)(F)$, there is a rule $r \in P$ such that $f \in r(F)$; this implies the existence of suitable sets B^+ and B^- to show compactness of $\bigcup P$.

Now assume that P is finite and consider $\bigcap P$. Extensiveness of $\bigcap P$ is again immediate from the extensiveness of rules in P. If $P = \emptyset$ then $(\bigcap P)(F) = \mathsf{B}$, so for any derivation $f \in (\bigcap P)(F)$ the sets $B^+ = B^- = \emptyset$ show compactness of $(\bigcap P)$. If $P \neq \emptyset$ then, for any fact $f \in (\bigcap P)(F)$ and any rule $r_i \in P$, we find sets B_i^+ and B_i^- by compactness of the derivation $f \in r_i(F)$. Thus, the sets $\bigcup_{r_i \in P} B_i^+$ and $\bigcup_{r_i \in P} B_i^-$ show compactness of $(\bigcap P)$. In particular, these sets are finite since P is. □

Another interesting type of operations is based on functional composition.

Definition 5. *The* composition $r_2 \circ r_1$ *of* r_1 *and* r_2 *is the function with* $(r_2 \circ r_1)(F) := r_2(r_1(F))$. *The* n-iteration r^n *of a rule* r *is the* n*-fold composition with itself, i.e.,* r^0 *is the identity function and* $r^{i+1} = r \circ r^i$. *The* saturation r^∞ *of* r *is the union of all its* n*-iterations, i.e.,* $r^\infty := \bigcup\{r^i \mid i \geq 0\}$.

Example 5. Iterations of the T_P operator of Definition 2 are equivalent to iterations of abstract rules: $T_P^i = (\bigcup P)^i$ and $T_P^\infty = (\bigcup P)^\infty$.

Example 6. In general, composition does not preserve compactness. Consider the rules $r_1 : q \leftarrow p(X)$ and $r_2 : r \leftarrow \textbf{not}\ q$ over the infinite base $\mathsf{B} = \{q, r, p(a), p(s(a)), \ldots\}$. The composition $r_2 \circ r_1$ can be described as

$$(r_2 \circ r_1)(F) = r_1(F) \cup \begin{cases} \{r\} & \text{if } p(s^n(a)) \notin F \text{ for all } n \geq 1 \\ \emptyset & \text{otherwise.} \end{cases}$$

Thus, for every choice of finite sets $\langle B^+, B^- \rangle$, there is a set F' with $B^+ \subseteq F'$ and $B^- \cap F' = \emptyset$ such that $r \notin (r_2 \circ r_1)(F')$. The function $r_2 \circ r_1$ is not compact.

Theorem 3. *Let* r_1 *and* r_2 *be abstract rules. If* r_2 *is monotone, then* $r_2 \circ r_1$, r_2^n *for all* $n \geq 0$, *and* r_2^∞ *are abstract rules.*

Proof. Consider $r_2 \circ r_1$. Extensiveness of r_1 and r_2 yields $F \subseteq r_1(F) \subseteq r_2(r_1(F)) = (r_2 \circ r_1)(F)$. For compactness, consider some $f \in (r_2 \circ r_1)(F)$. By compactness of r_2, we find finite sets B_2^+ and B_2^- for deriving f from $r_1(F)$. Since r_2 is monotone, we can assume without loss of generality that $B_2^- = \emptyset$. As B_2^+ is finite, it has the form $\{f_1, \ldots, f_m\}$. For each $f_i \in B_2^+$, there are sets B_{1i}^+ and B_{1i}^- that show compactness of the derivation $f_i \in r_1(F)$. The sets $B^+ = \bigcup_{i=1}^m B_{1i}^+$ and $B^- = \bigcup_{i=1}^m B_{1i}^-$ show the compactness of the derivation $f \in (r_2 \circ r_1)(F)$.

The claim for r^n follows by induction: the result is clear for r^0, and the induction step follows from the result for composition. The claim for saturation follows by combining the results for n-iteration and Theorem 2. □

4 Strong Equivalence of Abstract Programs

Logic programs P_1 and P_2 are strongly equivalent if the programs $P \cup P_1$ and $P \cup P_2$ have exactly the same stable models for any program P and set of facts F_0 [12,16]. We apply the same definition to abstract logic programs.

Theorem 4. *Every abstract logic program* P *is strongly equivalent to* $\{\bigcup P\}$.

Proof. To simplify the proof, we use the following auxiliary definition. An abstract program P_1 *is subsumed by* an abstract program P_2, written $P_1 \sqsubseteq P_2$, if the following holds: for every rule $r_1 \in P_1$ and derivation $f \in r_1(F)$ with a body $\langle B^+, B^- \rangle$, there is a rule $r_2 \in P_2$ and derivation $f \in r_2(F)$ for which $\langle B^+, B^- \rangle$ is also a body. Clearly, $P \sqsubseteq \{\bigcup P\}$ and $\{\bigcup P\} \sqsubseteq P$.

To complete the proof, we show some general properties of subsumption. Consider a set of facts F_0 and abstract programs P_1 and P_2.

1. If $P_1 \sqsubseteq P_2$, then every well-supported set for P_1, F_0 is well-supported for P_2, F_0.
2. If $P_2 \sqsubseteq P_1$, then every model of P_1, F_0 is a model of P_2, F_0.
3. If $P_1 \sqsubseteq P_2$ and $P_2 \sqsubseteq P_1$, then P_1 and P_2 are strongly equivalent.

The overall claim thus is an immediate consequence of the last item.

Assume that $P_1 \sqsubseteq P_2$ and that F is well-supported for P_1, F_0 using the order $\prec$. Then for every $f \in F$ there is a rule $r_1 \in P_1$ such that $f \in r_1(F)$ has a body $\langle B^+, B^- \rangle$ in F with $f' \prec f$ for all $f' \in B^+$. Since $P_1 \sqsubseteq P_2$, there is a rule $r_2 \in P_2$ with $f \in r_2(F)$ and the same body.

Assume that $P_2 \sqsubseteq P_1$ and that F is a model for P_1, F_0. Suppose for a contradiction that F is not a model of P_2, F_0. Then there is a rule $r_2 \in P_2$ and a fact $f \in r_2(F) \setminus F$. By $P_2 \sqsubseteq P_1$, there is a rule $r_1 \in P_1$ with $f \in r_1(F)$. This contradicts the assumptions that F is a model of P_1, F_0.

Assume that $P_1 \sqsubseteq P_2$ and $P_2 \sqsubseteq P_1$, and let P be an arbitrary abstract program. Clearly, $P \cup P_1 \sqsubseteq P \cup P_2$ and $P \cup P_2 \sqsubseteq P \cup P_1$. Thus, by the first two properties, every stable model of $P \cup P_1$ and F_0 is also a stable model of $P \cup P_2$ and F_0, and vice versa. □

It is easy to see that intersection $\bigcap P$ does not lead to strong equivalence. However, we can establish relevant results for composition, iteration, and saturation. The proof of the following result uses Proposition 2 to construct the well-founded order that is needed to show that a model is stable.

Proposition 3. *For monotone rules r_1 and r_2, $\{r_1, r_2\}$ is strongly equivalent to $\{r_2 \circ r_1\}$.*

Theorem 5. *For a monotone rule r, all of the programs $\{r\}$, $\{r^n\}$ for $n \geq 2$, and $\{r^\infty\}$ are pairwise strongly equivalent.*

Proof. The strong equivalence of $\{r\}$ and $\{r^n\}$ for any $n \geq 2$ is shown by induction. By Proposition 3, $\{r^{n+1}\}$ is strongly equivalent to $\{r, r^n\}$. By induction $\{r^n\}$ is strongly equivalent to $\{r\}$, so that $\{r, r^n\}$ is strongly equivalent to $\{r, r\} = \{r\}$ as required.

For the limit $\{r^\infty\}$, note that $\{r^\infty\} = \bigcup\{r^n \mid n \geq 1\}$ is strongly equivalent to $\{r^n \mid n \geq 1\}$ by Theorem 4. The result follows as each $\{r^n\}$ is strongly equivalent to $\{r\}$. □

5 Reliances and Stratifications

Stable models can not always be computed by applying rules in a bottom-up fashion. Due to nonmonotonicity, a rule that was applicable initially may no longer be applicable after further facts have been derived. Conversely, it can also happen that one rule is applicable only after another rule has been applied. Both types of relationships between rules are useful to gain insights about the stable models of a given program and to guide the computation of stable models.

We are interested in two types of dependencies, which we call *reliances* to avoid confusion with existing notions: negative reliance (the application of a rule may inhibit the application of another rule) and positive reliance (the application of a rule may enable the application of another rule). In both cases we ask whether this interaction of rules can occur during a normal derivation, i.e., when considering some set of (already derived) facts. We could just consider arbitrary sets of facts here, but we can obtain

stronger results if we restrict attention to fact sets which can actually occur during the derivation of a stable model. For the next definition, recall that the notation r_f and $\langle B_f^+, B_f^- \rangle$ was introduced for well-supported sets in Definition 3.

Definition 6. *Given a rule r and finite sets B^+ and B^-, we say that f* follows from *$\langle B^+, B^- \rangle$ by r if $f \in r(F)$ for every set $F \subseteq \mathsf{B}$ with $B^+ \subseteq F$ and $B^- \cap F = \emptyset$.*

Let $\mathscr{D} \subseteq 2^{\mathsf{B}}$ be a set of sets of facts that are admissible as input. *A set $F \subseteq \mathsf{B}$ is* derivable *from $\mathscr{D}$ and P if there is a set $F_0 \in \mathscr{D}$ such that F is well-supported for P and F_0, and for all $f \in F \setminus F_0$ we have: $f' \in r(F')$ for all f' that follow from $\langle B_f^+, B_f^- \rangle$ by r_f.*

Intuitively, derivable sets are well-supported sets that contain all the facts that must certainly follow when from the rule applications that establish well-supportedness. The use of $\mathscr{D}$ allows us to consider all sets of facts as admissible inputs (if $\mathscr{D} = 2^{\mathsf{B}}$) or to restrict attention to a single input F_0 (if $\mathscr{D} = \{F_0\}$). A common restriction in Datalog rules is that some "intensional" predicate symbols are not allowed in the input, while in existential rules one does not allow function symbols in input facts, although (skolem) functions may occur in derivations. When irrelevant or clear from the context, we speak of derivable sets without mentioning $\mathscr{D}$ and P explicitly.

Definition 7. *A rule r_2* positively relies on *a rule r_1, written $r_1 \xrightarrow{+} r_2$, if there is a derivable set of facts F with $\bot \notin F$ such that there is a fact $f_2 \in F$ with $r_{f_2} = r_2$, and a fact $f_1 \in B_{f_2}^+$ with $r_{f_1} = r_1$.*

A rule r_2 negatively relies on *a rule r_1, written $r_1 \xrightarrow{-} r_2$, if there is a derivable set of facts F, a derivation $f_2 \in r_2(F)$ with body $\langle B_2^+, B_2^- \rangle$, and a derivation $f_1 \in r_1(F) \cap B_2^-$ with body $\langle B_1^+, B_1^- \rangle$, such that $\bot$ does not follow from $\langle B_1^+, B_1^- \rangle$ by r_1.*

In both cases, $\bot$ is taken into account to exclude situations where the application of r_1 leads to an inconsistency.

In practice, it may not always be possible to compute $\xrightarrow{+}$ and $\xrightarrow{-}$ exactly. For example, it may be difficult to determine if a certain set is derivable (based on a given choice of $\mathscr{D}$). However, all of our results remain correct when working with larger relations instead of $\xrightarrow{+}$ and $\xrightarrow{-}$. Therefore, a practical algorithm may overestimate reliances without putting correctness at risk.

Example 7. A very simple overestimation of reliances on normal logic programs is related to the classical notion of stratification. Consider logic programming rules r_1 and r_2. We write $r_1 \rightsquigarrow^+ r_2$ if a predicate symbol that occurs in the head of r_1 occurs in a non-negated body atom of r_2, and $r_1 \rightsquigarrow^- r_2$ if a head predicate symbol of r_1 occurs in a negated body atom of r_2. It is easy to see that $\xrightarrow{+} \subseteq \rightsquigarrow^+$ and $\xrightarrow{-} \subseteq \rightsquigarrow^-$.

Example 8. A more elaborate notion of reliance was recently developed for *existential rules* by the authors [14]. Existential rules are first skolemised, which leads to normal logic programs where each function symbol occurs in the head of exactly one rule and in no rule bodies. Functions are not allowed in input fact sets either. In this special case, one can find all positive and negative reliances by considering only sets of facts F that contain no function symbols. It has been shown that, for programs that do not use $\bot$, checking $r_1 \xrightarrow{+} r_2$ is NP-complete, while $r_1 \xrightarrow{-} r_2$ can be checked in polynomial time [14]. This is an exact computation of the relations of Definition 7 for this specific case, not an overestimation.

Definition 8. *Consider a program P and a (finite or countably infinite) sequence of disjoint sets $\mathbf{P} = \langle P_1, P_2, \ldots \rangle$ with $\bigcup_{P_i \in \mathbf{P}} P_i = P$. $\mathbf{P}$ is an* R-stratification *of P if, for all rules $r_1 \in P_i$ and $r_2 \in P_j$,*

- *if $r_1 \xrightarrow{+} r_2$ then $i \leq j$;*
- *if $r_1 \xrightarrow{-} r_2$ then $i < j$.*

If P has an R-stratification then it is called R-stratifiable*.*

It should be noted how reliances interact with unions of rules. Any R-stratification of $P \cup \{\bigcup P'\}$ gives rise to an R-stratification of $P \cup P'$, while the converse is not true in general. This is analogous to the relationship of classical stratification (considering normal rules as unions of their groundings as in Example 4) to *local stratification* (considering stratification on the infinitely many ground instances [15]). It also illustrates that our approach can capture (and extend) both of these ideas.

6 Computing Stable Models of Stratified Rule Sets

We now show that R-stratified abstract programs have at most one stable model, which can be obtained by deterministic computation. For programs that have a finite stratification, this leads to a semi-decision procedure for entailment, provided that the given abstract rules are computable functions.

Definition 9. *Given a stratification $\mathbf{P} = \langle P_1, P_2, \ldots \rangle$ of P, we define*

$$S^0_{\mathbf{P}}(F) := F, \qquad S^{i+1}_{\mathbf{P}}(F) := T^{\infty}_{P_{i+1}}(S^i_{\mathbf{P}}(F)), \qquad S^{\infty}_{\mathbf{P}}(F) := \bigcup_{P_i \in \mathbf{P}} S^i_{\mathbf{P}}(F).$$

For the remainder of this section, let P denote an R-stratified program with R-stratification $\mathbf{P} = \langle P_1, P_2, \ldots \rangle$, and let F denote an admissible set of facts, i.e., $F \in \mathscr{D}$. We use the abbreviations $P^m_1 := \bigcup_{i=1}^m P_i$, $P^0_1 := \emptyset$, and $S^i_P := S^i_P(F)$. The main result that we will show in this section is the following.

Theorem 6. *If $\bot \notin S^{\infty}_P$ then S^{∞}_P is the unique stable model of P and F. Otherwise, P and F do not have a stable model.*

Lemma 1. *For every $P_j \in \mathbf{P}$ and $\ell \geq 0$, if $\bot \notin T^{\ell}_{P_j}(S^{j-1}_P)$ then $T^{\ell}_{P_j}(S^{j-1}_P)$ is derivable.*

Proof. For any $i \geq 1$ and $k \geq 0$, we use the abbreviation $T^k_i := T^k_{P_i}(S^{i-1}_P)$. We define a well-founded partial order $\prec$ on S^{∞}_P by setting $f_1 \prec f_2$ for facts $f_1, f_2 \in S^{\infty}_P$ iff there are numbers $i, k \geq 0$ such that $f_1 \in T^k_{i+1}$ and $f_2 \notin T^k_{i+1}$.

We proceed by induction over the derivation steps, i.e., we assume that T^k_i is derivable for all i, k such that $i < j$, or $i = j$ and $k < \ell$. Consider an arbitrary fact $f \in T^{\ell}_j$. Let $P_i \in \mathbf{P}$ and $k \geq 0$ be such that f was first derived in T^k_i. If $k = 0$, then $i = 1$ and $f \in F$ (since all facts in T^0_i for $i > 0$ do already occur in an earlier iteration T^m_{i-1} for some $m \geq 1$); in this case, f clearly satisfies the conditions of derivability.

If $k > 0$, then there is a rule $r \in P_i$ such that $f \in r(T^{k-1}_i)$. Let $\langle B^+, B^- \rangle$ be a body for this derivation with respect to T^{k-1}_i. We claim that $\langle B^+, B^- \rangle$ is also a body for $f \in r(T^{\ell}_j)$.

Clearly, $B^+ \subseteq T_i^{k-1} \subseteq T_j^\ell$, and thus $\hat{f} \prec f$ for every $\hat{f} \in B^+$. Moreover, we show that $B^- \cap T_j^\ell = \emptyset$. By definition, $B^- \cap T_i^k = \emptyset$. Now suppose for a contradiction that $B^- \cap T_j^\ell \neq \emptyset$. Then there is a rule $r' \in P_{i'}$ and a number k' such that $B^- \cap r'(T_{i'}^{k'}) \neq \emptyset$, where either $i < i'$, or $i = i'$ and $k \leq k'$, and also $i' < j$, or $i' = j$ and $k' < \ell$. By induction hypothesis, $T_{i'}^{k'}$ is derivable. Since $\bot \notin T_j^\ell$, we also have $\bot \notin r'(T_{i'}^{k'})$. Hence $r' \dashrightarrow r$; together with $i \leq i'$ this contradicts the assumed stratification. Hence $\langle B^+, B^- \rangle$ is a body for $f \in r(T_j^\ell)$. This establishes the conditions for well-supportedness of f. The remaining conditions for derivability are immediate by construction, since $r(T_i^{k-1}) \subseteq T_j^\ell$. □

Lemma 2. *Consider numbers $i \leq j$ with $P_i, P_j \in \mathbf{P}$, and a rule $r \in P_i$. If $\bot \notin S_P^j$ then $r(S_P^j) \subseteq S_P^j$.*

Proof. By Proposition 1, $r(S_P^i) \subseteq S_P^i$. Now consider $j > i$. Suppose for a contradiction that $r(S_P^i) \not\subseteq S_P^j$. There is $k > i$ and $\ell \geq 0$ with $T_k^\ell := T_{P_k}^\ell(S_P^{k-1})$ such that $r(T_k^\ell) \subseteq S_P^j$ and $r(T_k^{\ell+1}) \not\subseteq S_P^j$. Thus, there is a fact $f \in r(T_k^{\ell+1}) \setminus S_P^j$. Let $\langle B^+, B^- \rangle$ be a body for this derivation.We have $B^+ \not\subseteq T_k^\ell$. Thus there is a fact $f' \in B^+ \setminus T_k^\ell$ that is derived by a rule $r' \in P_k$ from T_k^ℓ. By Lemma 1, $T_k^{\ell+1}$ is derivable, where $r_{f'} = r'$. Since $\bot \notin S_P^j$ and $T_k^{\ell+1} \subseteq S_P^j$, also $\bot \notin T_k^{\ell+1}$. Hence $r' \stackrel{+}{\dashrightarrow} r$. Together with $i < j$ this contradicts the assumed stratification. □

Proposition 4. *If $\bot \notin S_P^\infty$ then S_P^∞ is a stable model of P and F.*

Proof. For every $r \in P$ and every derivation $f \in r(S_P^\infty)$, there is a body $\langle B^+, B^- \rangle$ by compactness. Since B^+ is finite, there is some $n \geq 0$ such that $B^+ \subseteq S_P^n$. By Lemma 2, $f \in S_P^n$. Hence S_P^∞ is a model of P and F.

For every $n \geq 0$, S_P^n is well-supported by Lemma 1. Let $\prec_n$ be an according well-founded order that is a suborder of $\langle \mathbb{N}, < \rangle$, which exists by Proposition 2. We construct a suitable order $\prec$ to show well-supportedness of S_P^∞ as follows. For every $n \geq 1$, let L_n be the set $S_P^n \setminus S_P^{n-1}$, ordered by the well-founded order $\prec_n$ (restricted from S_P^n to L_n). We now define $\prec$ to be the transitive closure of the following set:

$$\{f_1 \prec f_2 \mid f_1, f_2 \in L_n, f_1 \prec_n f_2\} \cup \{f_1 \prec f_2 \mid f_1 \in L_n, f_2 \in L_m, n < m\}.$$

This order is well-founded (it can clearly be embedded into the ordinal ω^2, since every $\prec_i$ can be embedded into ω). If $f_1, f_2 \in S_P^n$ and $f_1 \prec_n f_2$, then $f_1 \prec f_2$. Therefore, the bodies used to show well-foundedness of a fact $f \in L_n$ can be used to show well-foundedness of $f \in S_P^\infty$. □

The final ingredient to the proof of Theorem 6 is the following lemma. In the classical case, an analogous result was shown by taking advantage of the Gelfond-Lifschitz reduct of P [14]. Since this is not available for abstract rules, we need to take a very different approach, using an induction over the sets of facts in M for which well-foundedness is established by a rule from stratum P_k or below.

Lemma 3. *If M is a stable model of P and F, then $S_P^\infty = M$.*

Proof (of Theorem 6). If $\bot \notin S_P^\infty$ then by Proposition 4, S_P^∞ is a stable model of P and F. Together with Lemma 3 this implies that S_P^∞ is the unique stable model of P and F.

If $\bot \in S_P^\infty$ suppose for a contradiction that M is a stable model of P and F. Then by Lemma 3, $S_P^\infty = M$, which contradicts the fact that $\bot \notin M$. □

7 Stratifying Programs with Equality

In this section, we apply the previous results to show how a normal logic program with equality may be stratified. Classical logic programming engines support syntactic (term) equality that is easy to handle: it may only occur in the body of a rule. In contrast, *equality generating dependencies* in databases are rules that may infer new equalities between domain elements [1]. Inferred equality also plays a major role in ontology languages, which can be processed with answer set programming engines [6]. Fortunately, the special characteristics of equality can be fully expressed by logic programming rules, using the following well-known equality theory:

$$X \approx X \leftarrow \tag{4}$$

$$X \approx Y \leftarrow Y \approx X \tag{5}$$

$$X \approx Z \leftarrow X \approx Y, Y \approx Z \tag{6}$$

$$p(X_1,\ldots,Y,\ldots,X_n) \leftarrow X \approx Y, p(X_1,\ldots,X,\ldots,X_n) \tag{7}$$

where a rule of the form (7) is required for every n-ary predicate p (in a given program P), and every position of X within that predicate. We call this logic program $P_\approx$. While this approach allows logic programs to support equality without defining a special semantics, it has severe effects on stratification.

Example 9. Consider the program P that consists of the following rules

$$\mathsf{human}(X) \leftarrow \mathsf{biped}(X), \mathbf{not}\ \mathsf{bird}(X) \tag{8}$$

$$Y \approx Z \leftarrow \mathsf{human}(X), \mathsf{birthplace}(X,Y), \mathsf{birthplace}(X,Z) \tag{9}$$

together with a suitable equality theory $P_\approx$ for the predicates used therein. Rule $r_{(9)}$ states that each human has at most one birthplace. Let r_{bird} denote the version of rule (7) in $P_\approx$ for predicate bird. Now P cannot be R-stratified: if all set of ground facts are allowed as input, we have $r_{(8)} \xrightarrow{+} r_{(9)} \xrightarrow{+} r_{\mathsf{bird}} \xrightarrow{-} r_{(8)}$.

The previous example illustrates the fact that the equality theory leads to almost arbitrary reliances between otherwise unrelated rules, thus preventing stratification. This potential interaction is hardly desirable in this case, since no bird can ever be a birthplace. In [14] the authors have proposed the use of constraints to reduce the amount of reliances. We can obtain a similar effect using abstract rules.

Example 10. Consider the constraint $r_\bot : \bot \leftarrow \mathsf{bird}(X), \mathsf{birthplace}(Y,X)$ and the program P of Example 9. Define the program $P' := \{r_\bot \circ r \mid r \in P\}$, which immediately applies the constraint after each rule application. Instead of $r_{(9)} \xrightarrow{+} r_{\mathsf{bird}}$, we now find $(r_\bot \circ r_{(9)}) \not\xrightarrow{+} (r_\bot \circ r_{\mathsf{bird}})$ since $\bot$ is derived in all cases where the reliance could occur. Unfortunately, this approach still fails to make P' R-stratifiable, since we still find $(r_\bot \circ r_{(8)}) \xrightarrow{+} (r_\bot \circ r_{(9)}) \xrightarrow{+} (r_\bot \circ r_{(5)}) \xrightarrow{+} (r_\bot \circ r_{\mathsf{bird}}) \xrightarrow{-} (r_\bot \circ r_{(8)})$.

The symmetry rule (5) is used in the previous example to ensure that every reliance can be shown without violating the constraint. This is unfortunate since the program has only at most one unique stable model: in all situations where the chain of reliances of the example is mirrored by an actual chain of rule applications, the constraint $r_\bot$ must be violated. This problem can be overcome by incorporating equality reasoning into each rule application as follows.

Example 11. Let $P_\approx$ denote the equality theory for rules (8) and (9). Define a rule $\hat{r} := r_\bot \circ (\bigcup P_\approx)^\infty$, and rules $r_1 := \hat{r} \circ r_{(8)}$ and $r_2 := \hat{r} \circ r_{(9)}$. Note that these are indeed abstract rules by Theorem 3. Admissible input sets $\mathscr{D}$ are defined to be all models of $\hat{r}$. Then the program $\{r_1, r_2\}$ is R-stratified, and the only reliance is $r_1 \xrightarrow{+} r_2$. By Theorems 4 and 5, as well as Proposition 3, the stable models of $\{r_1, r_2\}$ are identical to the stable models of P' from Example 10. By Theorem 6, P' thus has a unique stable model whenever it is satisfiable.

The previous example outlines an interesting general approach to analyse the effects of equality. More important, however, is the fact that we have defined this extension and verified its key properties in a few lines. In contrast, the extension with constraints sketched in Example 10 originally required several pages of correctness proofs [14]. A major goal of our abstract framework is to extract the common ideas of such proofs, to provide an easy-to-use toolbox for establishing similar properties for many different scenarios and kinds of rules.

8 Conclusions

In this work, we proposed an abstract framework for studying logic programs, where rules are simply viewed as functions over an abstract set of derivable facts. We have shown that recent results on stratification and stable models can be lifted to this general case, and how these results can be instantiated to obtain a new approach for dealing with equality in logic programs. This result also takes advantage of a variety of semantic-preserving algebraic operations that we have introduced to construct abstract rules.

The purpose of this work is to demonstrate how abstract rules can serve as a powerful framework for establishing universal results about rule-based reasoning, which can readily be instantiated in concrete cases. There are numerous directions into which this research can be extended next. An obvious step is to define abstract notions of other semantics, such as the well-founded semantics, and to lift other relevant conditions, such as order consistency [8] and acyclicity [9,4,11]. In each case, new concrete applications of these results should be established, thus bridging gaps between different areas where rule languages are considered. Finally, even our basic notion of abstract rule may still be extended further, e.g., by allowing rules to retract facts. Moreover, our approach does not cover disjunctive rules, although these can often be simulated using nonmonotonic negation by rewriting $p \to q_1 \vee q_2$ as $p, \mathbf{not}\, q_1 \to q_2$ and $p, \mathbf{not}\, q_2 \to q_1$ [5].

Besides extensions of the abstract rules framework, it is also worthwhile to explore further application areas for these notions. All examples given herein are based on normal logic programs. Our treatment of equality shows that this already allows interesting applications, but it would also be interesting to consider different notions of rules. Possible candidates are rules that support datatype reasoning and data-related constraints.

Acknowledgements This work was supported by the Royal Society, the Seventh Framework Program (FP7) of the European Commission under Grant Agreement 318338, 'Optique', and the EPSRC projects ExODA, Score! and MaSI3.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley (1994)
2. Baget, J.F.: Improving the forward chaining algorithm for conceptual graphs rules. In: Dubois, D., Welty, C.A., Williams, M.A. (eds.) Proc. 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'04). pp. 407–414. AAAI Press (2004)
3. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artificial Intelligence 175(9–10), 1620–1654 (2011)
4. Deutsch, A., Nash, A., Remmel, J.B.: The chase revisited. In: Lenzerini, M., Lembo, D. (eds.) Proc. 27th Symposium on Principles of Database Systems (PODS'08). pp. 149–158. ACM (2008)
5. Eiter, T., Fink, M., Tompits, H., Woltran, S.: On eliminating disjunctions in stable logic programming. In: Dubois, D., Welty, C.A., Williams, M.A. (eds.) Proc. 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'04). pp. 447–458. AAAI Press (2004)
6. Eiter, T., Krennwallner, T., Schneider, P., Xiao, G.: Uniform evaluation of nonmonotonic DL-programs. In: Lukasiewicz, T., Sali, A. (eds.) Proc. 7th Int. Symposium on Foundations of Information and Knowledge Systems (FoIKS'12). LNCS, vol. 7153, pp. 1–22. Springer (2012)
7. Fages, F.: A new fixpoint semantics for general logic programs compared with the well-founded and the stable model semantics. New Generation Comput. 9(3/4), 425–444 (1991)
8. Fages, F.: Consistency of Clark's completion and existence of stable models. Meth. of Logic in CS 1(1), 51–60 (1994)
9. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theoretical Computer Science 336(1), 89–124 (2005)
10. Krötzsch, M., Magka, D., Horrocks, I.: Concrete results on abstract rules. Tech. rep., University of Oxford (2013), available from http://korrekt.org/page/Publications
11. Lierler, Y., Lifschitz, V.: One more decidable class of finitely ground programs. In: Hill, P.M., Warren, D.S. (eds.) Proc. 25th Int. Conf. on Logic Programming (ICLP'09). LNCS, vol. 5649, pp. 489–493. Springer (2009)
12. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Logic 2(4), 526–541 (2001)
13. Lloyd, J.W.: Foundations of Logic Programming. Springer (1988)
14. Magka, D., Krötzsch, M., Horrocks, I.: Computing stable models for nonmonotonic existential rules. In: Proc. 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI'13). AAAI Press/IJCAI (2013), to appear
15. Przymusinski, T.C.: On the declarative and procedural semantics of logic programs. J. Autom. Reasoning 5(2), 167–205 (1989)
16. Turner, H.: Strong equivalence made easy: nested expressions and weight constraints. TPLP 3(4–5), 609–622 (2003)