On the Properties of Atom Definability and Well-Supportedness in Logic Programming

Pedro Cabalar, Jorge Fandinno, Luis Fariñas del Cerro, David Pearce, Agustín Valverde

On the Properties of Atom Definability and Well-Supportedness in Logic Programming

Pedro Cabalar[1(✉)], Jorge Fandinno[1,2], Luis Fariñas[2], David Pearce[3], and Agustín Valverde[4]

[1] Universidade da Coruña, A Coruña, Spain
{cabalar,jorge.fandino}@udc.es, jorge.fandinno@irit.fr
[2] University of Toulouse IRIT, CNRS, Toulouse, France
farinas@irit.fr
[3] Universidad Politécnica de Madrid, Madrid, Spain
david.pearce@upm.es
[4] Universidad de Málaga, Málaga, Spain
a_valverde@ctima.uma.es

Abstract. We analyse alternative extensions of stable models for non-disjunctive logic programs with arbitrary Boolean formulas in the body, and examine two semantic properties. The first property, we call *atom definability*, allows one to replace any expression in rule bodies by an auxiliary atom defined by a single rule. The second property, *well-supportedness*, was introduced by Fages and dictates that it must be possible to establish a derivation ordering for all true atoms in a stable model so that self-supportedness is not allowed. We start from a generic fixpoint definition for well-supportedness that deals with: (1) a monotonic basis, for which we consider the whole range of intermediate logics; and (2), an assumption function, that determines which type of negated formulas can be added as defaults. Assuming that we take the strongest underlying logic in such a case, we show that only Equilibrium Logic satisfies both atom definability and strict well-suportedness.

1 Introduction

Almost 30 years ago, the introduction of the *stable models* [1] semantics for normal logic programs constituted the first general semantics for default negation that was defined on any normal logic program, without limitations on the syntactic dependences among atoms and rules. Since then, many extensions of stable models have been proposed in the literature to cope with more and more general syntactic fragments that went beyond normal logic programs. If we exclusively

Partially supported by Xunta de Galicia (projects GPC ED431B 2016/035 and 2016-2019 ED431G/01 for CITIC center) and ERDF; by the Centre International de Mathématiques et d'Informatique de Toulouse (CIMI), contract ANR-11-LABEX-0040-CIMI within program ANR-11-IDEX-0002-02; by UPM RP151046021 and by Spanish MINECO project TIN2015-70266-C2-1-P.

DOI: 10.1007/978-3-319-65340-2 51

focus on propositional connectives, rule heads were soon extended to include disjunction [2] and negative literals [3]. Going a step forward, [4] introduced a type of rule $B \rightarrow H$ where both the body B and the head H could be a so-called *nested expression*, that is, a Boolean formula allowing conjunction, disjunction and negation, but not the implication symbol, which could not be nested. The first extension of stable models to arbitrary propositional formulas, including nested implications, was actually provided with the previous definition of *Equilibrium Logic* [5] which, as proved in [6], is a conservative extension of nested expressions and, as shown in [7], can be alternatively described in terms of a formula reduct. Although Equilibrium Logic constitutes nowadays one of the most successful and better studied logical characterisations for *Answer Set Programming* (ASP), other approaches have been proposed trying to overcome some features on which no agreement seems to have been reached so far. For instance, one of those properties pursued by some authors is that stable models of a program should be *minimal* with respect to the set of their true atoms. Although this holds for disjunctive logic programs in all ASP semantics, the first proposals for negation in the head (or double negation in the body) [3] already violated minimality, this being also the case of Equilibrium Logic, which is a conservative extension. For instance, a common way to represent a choice rule in Equilibrium logic is using the expression:

$$\neg\neg p \rightarrow p \tag{1}$$

with double negation or, alternatively, its strongly equivalent disjunctive form $p \vee \neg p$ that uses negation in the head. The equilibrium models of (1) are $\emptyset$ and $\{p\}$, which is not minimal. In an attempt to guarantee minimality for programs with aggregates, Faber et al. [8] (FLP) came out with a new semantics that was generalised to arbitrary propositional formulas in [9] while keeping the minimality criterion. For instance, the unique FLP-stable model of (1) is $\emptyset$.

Apart from minimality, another property that has been recently considered by Shen et al. in [10] is the extension of Fages' *well-supportedness* [11], originally defined for normal logic programs, to rules with a more general syntax like, for instance, allowing Boolean formulas in the head or the body. Intuitively, a model M is said to be *well-supported* if its true atoms can be assigned a derivation ordering (via modus ponens) from the positive part of the program, while the interpretation of negated atoms is fixed with respect to M, acting like an assumption *a priori*. Fages proved that well-supported models coincide with stable models for normal logic programs, but did not specify how to extrapolate well-supportedness to other syntactic classes. For instance, consider rule (1) again and model $M = \{p\}$. If we consider that $\neg\neg p$ belongs to the "positive" part of the program, then it should be included in the derivation ordering, as any regular atom. However, doing so, there is no way to obtain p in a well-supported manner, since we would have to assign $\neg\neg p$ some level strictly smaller than p and find a different rule to justify $\neg\neg p$, something that does not exist. On the other hand, if $\neg\neg p$ is seen as a "negated" formula (as happens with negated atoms), then it should behave as an assumption and its truth should be fixed

with respect to M *a priori* as well. For $M = \{p\}$, $\neg\neg p$ would directly hold, and so, rule (1) would just behave as a fact for p, making it true.

In this paper, we provide a general definition of *well-supportedness* for programs with a head atom and a Boolean formula in the body. This definition is parametrized in two ways: (1) the type of formulas that can be used as "assumptions," that is, whose truth is fixed with respect to some model M; and (2), the monotonic logic that defines satisfaction of a rule body before applying the rule to derive a new conclusion. For (1), we study three cases: negated atoms, negated literals, and negated arbitrary formulas. For (2), we analyse the whole range of intermediate logics, from intuitionistic to classical logic, both included. In the paper, we prove that a group of variants collapse either into Equilibrium Logic or Clark's completion. To compare the different alternatives, we analyse one more property we call *atom definability*. This property asserts that if we replace occurrences of a formula φ in one or more rule bodies by a new auxiliary atom a, and we define this atom with an additional rule $\varphi \rightarrow a$, then we should get a strongly equivalent program (modulo the original alphabet). As we will see, this is important since semantics satisfying atom definability immediately provide a way to unfold programs with double negation into regular, normal logic programs. We show that, among the analysed variants, only those collapsing to Equilibrium Logic or to Clark's completion satisfy atom definability.

2 Auxiliary Atoms and Atom Definability

In this section we introduce the property of *atom definability* and motivate its importance for one of most powerful representational features of ASP: the definition of *auxiliary atoms* or *predicates*. Auxiliary atoms constitute a fundamental part of the widespread, commonly accepted, specification methodology for problem solving in ASP called *Generate, Define and Test* (GDT) that we will illustrate with a well-known example.

Example 1 (Hamiltonian cycles). Given a graph with nodes N and edges $E \subseteq N \times N$ find cyclic paths that visit each node exactly once.

> INPUT: Facts $\{node(X) \mid X \in N\}$ and $\{edge(X,Y) \mid \langle X,Y\rangle \in E\}$.
> OUTPUT: Facts $in(X,Y)$, edges forming a cyclic path that traverses all nodes.

In what follows, we represent logic program rules as implications $B \rightarrow H$, B being the rule *body* and H the rule *head*. We also use $\wedge$ and $\neg$ instead of commas and *not*, respectively. When using a expression with variables we assume it is an abbreviation of the conjunction of its possible ground instantiations. We also assume finite domains, leaving the infinite case for the future extension to first-order. A possible ASP representation of this problem would be:

$$edge(X,Y) \rightarrow 0\ \{in(X,Y)\}\ 1 \tag{2}$$
$$in(X,Y) \wedge in(X,Z) \wedge Y \neq Z \rightarrow \bot \tag{3}$$
$$in(X,Y) \wedge in(Z,Y) \wedge X \neq Z \rightarrow \bot \tag{4}$$
$$node(X) \wedge node(Y) \wedge \neg reach(X,Y) \rightarrow \bot \tag{5}$$
$$in(X,Y) \rightarrow reach(X,Y) \tag{6}$$
$$in(X,Z) \wedge reach(Z,Y) \rightarrow reach(X,Y) \tag{7}$$

The GDT methodology identifies three main groups of rules:

G = non-deterministic choices that *generate* potential solutions. In our case, we have the *choice rule* (2) so that, for each edge $edge(X,Y)$ in the graph, we may freely decide to include 0 or 1 instances of fact $in(X,Y)$ in our solution.

T = constraints that rule out undesired solutions (the *test* part). In the example, rules (3), (4), (5) check that we generate linear paths and that any pair of nodes are mutually reachable.

D = *definition* of auxiliary predicates when features for **G** and **T** cannot be directly represented in the ASP language. In the example, rules (6) and (7) define the *auxiliary predicate* $reach(X,Y)$, the transitive closure of $in(X,Y)$.

Although choice rules like (2) are already included in the standard input language *ASP Core 2.0* [12] (used for the ASP solvers competition), their semantics is actually defined in terms of auxiliary predicates. In the past, before the introduction of choices, a common way to represent (2) was:

$$edge(X,Y) \wedge \neg out(X,Y) \rightarrow in(X,Y) \tag{8}$$
$$edge(X,Y) \wedge \neg in(X,Y) \rightarrow out(X,Y) \tag{9}$$

using another auxiliary predicate $out(X,Y)$. An important observation, sometimes underestimated, is that these auxiliary predicates *are not a relevant part* of the problem definition. In Example 1, this problem definition involves input predicates *node* and *edge* plus the output predicate *in* describing the result. Predicates *out* and *reach* are representational resources used internally and are not to be included in the final result, as their extent is *irrelevant* for the problem solution. Think, for instance, that $out(X,Y)$ eventually collects the edges that are not $in(X,Y)$, so it does not provide new information and its use is merely technical. Moreover, if we had to compare two different ASP encodings of the Hamiltonian cycle problem, it seems obvious that predicates *out* and *reach* should not be part of the language. In fact, all ASP solvers provide some option to hide irrelevant predicates.

In the previous example, we saw a pair of features (the transitive closure and the choice rule) whose semantics could be directly defined in terms of auxiliary atoms. Of course, when doing so, *correctness* is not an issue, since the application of auxiliary atoms is done by definition. However, one may wonder what happens when we want to use auxiliary predicates to capture the meaning of some expression or formula that is not an ASP extension, but is part of the basic language from normal logic programs. Can we trust that the replacement

is correct? To illustrate this idea, consider the following common situation. We introduced a large graph instance for which we expect to find some Hamiltonian cycle, but the execution of the ASP solver yields no solution. In order to identify which constraint might have been applied, we decide to replace (5) by:

$$unreach(X,Y) \rightarrow \bot \tag{10}$$

$$node(X) \wedge node(Y) \wedge \neg reach(X,Y) \rightarrow unreach(X,Y) \tag{11}$$

i.e., the constraint body is now captured by an auxiliary predicate $unreach(X,Y)$ that keeps track of pairs of disconnected nodes. We momentarily remove (10) and find a pair of nodes in the graph for which some edge was missing by mistake. Then, we decide to keep (10), (11) for repeating this debugging technique. Now, can we safely replace (5) by (10)–(11) in any context?

This question is directly related to the formal property of *strong equivalence* [6]. Let V be some *vocabulary* or set of atoms, and $\mathcal{L}_V$ a syntactic language, with signature V, for which stable models are defined. Moreover, let $\mathrm{SM}(\Gamma)$ denote the set of stable models for some $\Gamma \subseteq \mathcal{L}_V$. We say that two theories Γ, Γ' are *strongly equivalent*, written $\Gamma \cong \Gamma'$, iff $\mathrm{SM}(\Gamma \cup \Delta) = \mathrm{SM}(\Gamma' \cup \Delta)$ for any arbitrary theory $\Delta \subseteq \mathcal{L}_V$. That is, Γ and Γ' provide the same results even when joined with any arbitrary common context Δ. This definition assumes that Γ, Γ' and Δ deal with the same common signature V. However, as we discussed before, auxiliary atoms should be kept hidden inside Γ and Γ' and not used for comparison. To cope with different vocabularies, we further specialise to one of the variants considered in [13] recently named *projective strong equivalence* in [14]. Suppose that the vocabularies of Γ and Γ' are, respectively, $V \cup U$ and $V \cup U'$, where U and U' represent hidden local atoms. We write now $\mathrm{SM}_V(\Gamma)$ to stand for the set of stable models of Γ restricted to vocabulary V, that is $\mathrm{SM}_V(\Gamma) \stackrel{\text{df}}{=} \{I \cap V \mid I \in \mathrm{SM}(\Gamma)\}$. Then, two theories Γ, Γ' satisfy *projective strong equivalence with respect to vocabulary* V (are *V-strongly equivalent*, for short), written $\Gamma \cong_V \Gamma'$ iff $\mathrm{SM}_V(\Gamma \cup \Delta) = \mathrm{SM}_V(\Gamma' \cup \Delta)$ for any theory $\Delta \subseteq \mathcal{L}_V$.

Using this formal concept, our example amounts to asking whether the programs $\Gamma = \{(5)\}$ and $\Gamma' = \{(10), (11)\}$ are V-strongly equivalent for any vocabulary V not containing $unreach(X,Y)$. Since (11) defines predicate *unreach*, and the latter cannot be defined anywhere else in the program, we obviously expect an affirmative answer to this question. We can even generalise this property in the following way. We say that a syntactic language $\mathcal{L}_V$ for vocabulary V is *implicational* if it contains, at least, the implication symbol $\rightarrow$. A *program* $\Gamma \subseteq \mathcal{L}_V$ from an implicational language $\mathcal{L}_V$ is a set of implications (*rules*) so that, for each rule $(\alpha \rightarrow \beta) \in \Gamma$ the formulas α (the *body*) and β (the *head*) do not contain implications[1] in their turn. Given a program Γ, let $\Gamma[\varphi/a]$ denote any theory resulting from arbitrarily replacing some occurrences of formula φ in the rule bodies of Γ by an atom a.

[1] We allow the exception $\varphi \rightarrow \bot$ since, as we will see later, this corresponds to $\neg\varphi$ in intermediate logics.

Definition 1 (Atom definability). *We say that a semantics for an implicational language $\mathcal{L}_V$ satisfies* atom definability *iff for any program $\Gamma \subseteq \mathcal{L}_V$, any subformula φ occurring in one or more bodies of Γ and any fresh atom $a \notin V$:*

$$\Gamma \cong_V \Gamma[\varphi/a] \cup \{\varphi \to a\} \qquad \square$$

In our example, we have replaced each ground instance of body formula $\varphi = node(c) \wedge node(d) \wedge \neg reach(c,d)$ in (5) by a new ground atom $a = unreach(c,d)$, (10) being the result $\Gamma[\varphi/a]$ of these replacements. On the other hand, it is easy to see that (11) corresponds to the new rule $\varphi \to a$. Thus, these replacements would be V-strongly equivalent to the original formula if we chose a semantics satisfying atom definability. In the general case, it seems clear that this is an interesting property that one would wish to guarantee, as it is behind the intuitive use of auxiliary predicates. However, the consequences of such a property may also affect the admissible semantics for other extensions going beyond normal or disjunctive logic programs. For instance, suppose that bodies with double negation were introduced in ASP for the first time and that no previous semantics for this extension were available. We could still see each doubly negated atom $\neg\neg p$ as an expression $\neg\varphi$ where $\varphi = (\neg p)$. Then, atom definability should allow us simply to replace $\neg\neg p$ by $\neg a$ providing that a is a fresh atom and we include a rule $\varphi \to a$ in the program. This means that atom definability immediately provides a method to remove double negation. For instance, take (1) again under this new reading: $\neg \underbrace{\neg p}_{\varphi} \to p$. Atom definability guarantees that:

$$\neg \ a \to p \tag{12}$$

$$\underbrace{\neg p}_{\varphi} \to a \tag{13}$$

is strongly equivalent to (1) relative to any original signature not containing a. In particular, as the stable models of (12)–(13) are $\{p\}$ and $\{a\}$, atom definability implies that the stable models of (1) *must be* the result of filtering out atom a, i.e., $\{p\}$ and $\emptyset$. In other words, any argument against obtaining $\{p\}$ and $\emptyset$ as stable models of (1) becomes an argument against obtaining $\{p\}$ and $\{a\}$ as regular stable models for the normal logic program (12)–(13), under the reasonable assumption that definition of auxiliary atoms works "as expected".

3 Formal Preliminaries

We recall some basic preliminaries and definitions that will be used in the rest of the paper. Here, we will restrict attention to propositional formulas, leaving first-order extensions for future work. Propositional formulas are built in the usual way over a *vocabulary* or set V of atoms plus connectives $\wedge$, $\vee$, $\to$ and $\bot$. We regard $\neg\varphi$ is an abbreviation of $\varphi \to \bot$, that $\top$ stands for $\neg\bot$ and that $\varphi \leftrightarrow \psi$ stands for $(\varphi \to \psi) \wedge (\psi \to \varphi)$. A *literal* is an atom p (*positive* literal)

or its negation $\neg p$ (*negative* literal). Given a conjunction of literals B, we write B^+ and B^- to respectively stand for the conjunctions of positive and negative literals in B (empty conjunctions correspond to $\top$). As expected, a *negated literal* can be either $\neg p$ or $\neg\neg p$. Note that, in intermediate logics, $\neg\neg p$ does not need to be equivalent to p whereas operator $\rightarrow$ is independent from $\wedge$ and $\vee$ and cannot be defined in terms of the latter. We say that an occurrence of formula φ in Γ is *positive* iff φ is in the scope of an even number of implication antecedents in Γ. We also say that occurrence φ is *negated* in Γ iff φ is in the scope of negation in Γ, that is, it is in the antecedent of some implication with $\bot$ as consequent. Note that φ can be both positive and negated in Γ: for instance, p is positive and negated in $(p \rightarrow q) \rightarrow \bot$, but q is just negated. A *Boolean formula* (also known as *nested expression* [4]), is a propositional formula exclusively formed with operators $\wedge, \vee, \neg$ and $\bot$. In other words, Boolean formulas do not contain $\rightarrow$ except in negations $\varphi \rightarrow \bot$, that is, $\neg\varphi$.

Let $\mathbf{L}$ be a propositional logic and let $M \models_{\mathbf{L}} \varphi$ represent its satisfaction relation for an interpretation M and formula φ. M is said to be a *model* of a theory Γ, written $M \models_{\mathbf{L}} \Gamma$, iff it satisfies all formulas in Γ. As usual, we say that Γ *entails* a formula ψ, written $\Gamma \models_{\mathbf{L}} \psi$, iff all models of Γ satisfy ψ. Similarly, φ is a *tautology*, written $\models_{\mathbf{L}} \varphi$, if any interpretation is a model of φ. We write **CL** to stand for Classical Logic. As usual, a *classical interpretation* M is just a set of atoms $M \subseteq V$. We write **IL** for Intuitionistic Logic and briefly recall its semantics. A *frame* is a pair $\langle W, \leq \rangle$ where W is a set of points or 'worlds' and $\leq$ is a partial order on W. An interpretation has the form $\langle W, \leq, v \rangle$ where $v : W \rightarrow 2^V$ assigns a set of true atoms to each world, satisfying $v(w) \subseteq v(w')$ for all pairs of worlds $w \leq w'$. We define when $M = \langle W, \leq, v \rangle$ satisfies a formula φ at some world w, written $M, w \models_{\mathbf{IL}} \varphi$, in the following recursive way:

- $M, w \models p$ iff $p \in v(w)$ for any atom $p \in V$
- $M, w \not\models \bot$
- $M, w \models \alpha \wedge \beta$ iff $M, w \models \alpha$ and $M, w \models \beta$
- $M, w \models \alpha \vee \beta$ iff $M, w \models \alpha$ or $M, w \models \beta$
- $M, w \models \alpha \rightarrow \beta$ iff for all $w' \geq w$, $M, w' \not\models \alpha$ or $M, w' \models \beta$

Intuitionistic logic **IL** is strictly weaker than classical logic **CL**, $\mathbf{IL} \subset \mathbf{CL}$, since many classical tautologies (such as $p \vee \neg p$) are not tautologies in **IL**. By an *intermediate* logic we mean any logic $\mathbf{L}$ lying between **IL** and **CL**, $\mathbf{IL} \subseteq \mathbf{L} \subseteq \mathbf{CL}$. The strongest (non-classical) intermediate logic is known as the logic of *Here-and-There*, **HT** and is defined by frames with two worlds $W = \{h, t\}$ (respectively called *here* and *there*) fixing $h \leq t$. An **HT** model can be represented as a pair $\langle H, T \rangle$ with $H \subseteq T$ corresponding to frame $\langle \{h, t\}, \leq, v \rangle$ where $v(h) = H$ and $v(t) = T$. An **HT** interpretation $M = \langle H, T \rangle$ is said to be an *equilibrium model* of a theory Γ iff $H = T$, $M \models_{\mathbf{HT}} \Gamma$ and there is no $H' \subset H$ such that $\langle H', T \rangle \models_{\mathbf{HT}} \Gamma$. *Equilibrium logic* is the logic induced by equilibrium models.

Theorem 1. *Equilibrium Logic satisfies the atom definability property (Definition 1). Moreover, this property holds even when allowing nested implications in Γ, given that the replaced occurrences of φ do not occur positively non-negated in Γ.*

The extension in Theorem 1 for nested implications does not hold if φ occurs positively non-negated in Γ. As an example, take the program Γ consisting of $((p \rightarrow q) \rightarrow p)$ and $(p \rightarrow q)$ whose only stable model is $\{p, q\}$. Assume that φ is the leftmost occurrence of p in the first formula, which occurs positively non-negated. Then, $\Gamma[\varphi/a] \cup \{\varphi \rightarrow a\}$ contains the rules $((a \rightarrow q) \rightarrow p)$, $(p \rightarrow q)$ and $(p \rightarrow a)$ yielding no stable model. The intuition for this limitation is that a positive, non-negated occurrence of a formula acts as a rule head in **HT**. In fact, $(p \rightarrow q) \rightarrow p$ is **HT**-equivalent to the pair of rules $\neg\neg q \rightarrow p$ and $\neg p \rightarrow \bot$.

Although equilibrium models are defined for arbitrary propositional theories, the syntactic fragment we will identify as *logic programs* in this paper will be more limited, since we are interested in extensions of normal programs for which we can still find a natural definition of well-supportedness. We define a *Boolean (logic) program* P to be a set of rules $B \rightarrow p$ where the body B is a Boolean formula and the head p is an atom. As usual, P is further said to be a *normal (logic) program* iff all rule bodies in P are conjunctions of literals. We assume the reader is familiar with normal programs and their stable model semantics [1]. As is well-known, equilibrium models coincide with stable models in the sense that an interpretation M is a *stable model* of a normal program P iff $\langle M, M\rangle$ is an equilibrium model of P, [5].

Clark's *completion* [15] of a normal program P, denoted as $\text{COMP}(P)$, corresponds to the union of P and the implications $p \rightarrow B_1 \vee \cdots \vee B_n$ for each atom $p \in V$ where $B_1, \ldots, B_n$ are the bodies of all rules $B_i \rightarrow p$ in P for that head atom. As usual, if no rules exist for p, then the empty disjunction corresponds to $\bot$. The intuitive reading of $\text{COMP}(P)$ is that each true atom in M must have some supporting rule $B_i \rightarrow p$ in P whose body is true in M, $M \models B_i$. We say that a classical interpretation M is a *supported* model of P iff $M \models_{\mathbf{CL}} \text{COMP}(P)$ and, by abuse of notation, we also write $\text{COMP}(P)$ to represent the supported models of P. For normal programs, it is well-known that $\text{SM}(P) \subseteq \text{COMP}(P)$ but the converse does not necessarily hold. The main difference relies on the behaviour of positive loops. For instance, take the program P_1:

$$q \wedge \neg r \rightarrow p \tag{14}$$

$$p \rightarrow q \tag{15}$$

Its completion is the conjunction of P_1 plus the implications $(r \rightarrow \bot)$, $(p \rightarrow q \wedge \neg r)$ and $(q \rightarrow p)$. The resulting theory is classically equivalent to $\neg r \wedge (p \leftrightarrow q)$ having two supported models $\emptyset$ and $\{p, q\}$ while only the former is stable. To overcome this difference, Fages [11] strengthened supported models as follows. A classical interpretation M is a *well-supported model* of a normal program P iff there exists a strict partial order $\prec$ on M such that, for every atom $p \in M$, there is a rule $(B_i \rightarrow p) \in P$ that satisfies: (i) $M \models B_i$ and (ii) $q \prec p$ for every positive literal q in B_i. In the example above, the supported model $M = \{p, q\}$ is not well-supported. To see why, note that the only support for p is (14) whose body holds in M. To be well-supported, we would need a strict order $\prec$ satisfying $q \prec p$ for the positive literal q in the body. However, the only support for q, in its turn, is (15) whose body also holds in M and we would also need its positive

literal to satisfy $p \prec q$. If we add fact p to program P_1, then the new program P_2 has a unique well-supported model $\{p, q\}$ where p is supported by the fact and q is supported by (15) with the order $p \prec q$. Fages proved that the stable models of a normal logic program coincide with its well-supported models.

4 Well-Supported Models of Boolean Programs

Extending the definition of supported models from normal to Boolean programs is straightforward: for each true atom p in M, we must still find some rule $B_i \rightarrow p$ in the program with true body $M \models B_i$ to support it. So, we add the formulas $p \rightarrow B_1 \vee \cdots \vee B_n$ collecting all bodies B_i for head p in the program – the fact that these bodies are Boolean formulas does not affect the definition in a substantial way. For instance, the completion of (1) would become $p \leftrightarrow \neg\neg p$ which is a classical tautology, its supported models being $\emptyset$ and $\{p\}$.

Theorem 2. *Supported models of Boolean programs satisfy atom definability.*

The extension of well-supportedness to Boolean bodies, however, is not so immediate, as it depends on the syntactic form of the rule body, treating negative and positive literals in a different way. Given a candidate model M, an interesting observation is that all negative literals are directly interpreted with respect to M, regardless of the derivation order $\prec$ we choose. Thus, we can simply add them to the program as a set of axioms $\Delta_M := \{\neg p \mid p \in V \setminus M\}$ we call *assumptions*. On the other hand, for finding a supporting rule $B \rightarrow p$ for p, all atoms in B^+ must be strictly smaller than p with respect to relation $\prec$. Let us define $M_{\prec p} := \{q \in M \mid q \prec p\}$, that is, all atoms in M strictly smaller than p. Using these ideas, we can rephrase the definition of well-supported model in a way that does not depend on the rule body syntax:

Proposition 1. *M is a well-supported model of a normal program P iff there exists a well-founded strict partial order $\prec$ on M such that, for each $p \in M$, there is a rule $(B \rightarrow p) \in P$ satisfying: $M_{\prec p} \cup \Delta_M \models_{\mathbf{CL}} B$.* □

The use of negated assumptions Δ_M shares some resemblance with McDermott and Doyle's [16] fixpoint definition of *expansion* E for non-monotonic modal logics: in that case, the epistemic negation $\neg L\varphi$ of any formula $\varphi \notin E$ can be added as assumption. As an example of Proposition 1, consider program P_3 consisting of $(b \wedge \neg c \rightarrow d)$ and fact b. Its unique well-supported model is $\{b, d\}$, associated to order $b \prec d$. It is easy to see that d is justified because the body of its rule $b \wedge \neg c$ is classically entailed by $M_{\prec d} = \{b\}$ and $\Delta_M = \{\neg c\}$. Now, Proposition 1 can be directly used to provide a definition of well-supported model for Boolean programs by simply generalising the form of rule bodies B from conjunctions of literals to Boolean formulas. Unfortunately, this direct extrapolation does not satisfy atom definability. Take (1) again and consider the interpretation $M = \{p\}$. As we only have one atom and this atom is true, $M_{\prec p} \cup \Delta_M = \emptyset$ while the only possible rule is not supported $\emptyset \not\models_{\mathbf{CL}} \neg\neg p$. However, as we explained in

Sect. 2, to respect atom definability, (1) should behave as the program (12)–(13) after removing atom a, so $\{p\}$ must be a stable model of both programs. This example apparently creates a false dilemma: either we choose well-supportedness or atom definability, but not both. We claim, however, that the apparent dilemma can be resolved by allowing the concept of well-supportedness to be parametrised in at least two different ways. A first, obvious way is to permit different logics to characterise the monotonic entailment relation in Proposition 1; so one would expect, for instance, that Equilibrium Logic corresponds to **HT** instead of **CL**. Different semantics may arise from considering other logics but, as we will show, if we focus on the whole family of intermediate logics, most variants collapse into a pair of non-monotonic alternatives, one of them being Equilibrium Logic. A second observation is that there is no reason *a priori* why the set of assumptions Δ_M should be restricted to negated atoms. As mentioned, in non-monotonic modal logics, assumptions may involve negations of more general formulas. Given a class of formulas $\mathcal{C} \subseteq \mathcal{L}_V$, we define the corresponding set of assumptions with respect to a classical interpretation M as $\Delta^{\mathcal{C}}_M := \{\neg\varphi \mid \varphi \in \mathcal{C},\ M \not\models_{\mathbf{CL}} \varphi\}$ that is, we collect the negation of all formulas of class $\mathcal{C}$ not satisfied by M. We are particularly interested in three classes: the set of atoms, the set of literals and the set of propositional formulas, respectively denoted with the superscripts at, lit and for. Thus, Δ_M used before corresponds now to Δ^{at}_M. This leads us to the following general definition of well-supported model.

Definition 2 (Well-supported model). *Given a logic* **L** *and a class of assumption formulas* $\mathcal{C}$*, a set of atoms* M *is a* $\mathbf{L}^{\mathcal{C}}$-well-supported model *(for short,* $\mathbf{L}^{\mathcal{C}}$-model*) of a Boolean program* P *iff there exists a strict partial order* $\prec$ *on* M *such that, for each* $p \in M$*, there is a rule* $(B \rightarrow p) \in P$ *satisfying* $M_{\prec p} \cup \Delta^{\mathcal{C}}_M \models_{\mathbf{L}} B$. □

Under this new notation, [10] corresponds now to $\mathbf{CL}^{at}$-models, that is, we use classical entailment of rule bodies and take negated atoms as assumptions. If we consider the class of literals $\mathcal{C} = lit$ as assumptions, then we obtain the following characterisations of supported and equilibrium models.

Theorem 3. *If* P *is a Boolean program and* M *a classical interpretation:*

(i) M *is a supported model of* P *iff* M *is a* $\mathbf{CL}^{lit}$*-model of* P*.*
(ii) $\langle M, M\rangle$ *is an equilibrium model of* P *iff it is a* $\mathbf{HT}^{lit}$*-model of* P*.* □

Property (i) means that we can see Clark's completion (supportedness) as a degenerate case of well-supportedness. This is because Δ^{lit}_M has M as its unique classical model, so the other part of the well-supportedness condition $M_{\prec p}$ has no effect at all. Property (ii), however, has a different reading. It means that equilibrium models are well-supported if we take negated literals as assumptions and use **HT** entailment to interpret them. Remember that $\neg\neg p$ is not **HT**-equivalent to p. Definition 2 gives us a new reading of their meanings: $\neg\neg p$ corresponds to assuming that p will not eventually become false, while p must be derived from rules under some derivation order $\prec$. Therefore, Equilibrium Logic

simultaneously satisfies well-supportedness (in a non-degenerate way) besides atom definability. What happens with the rest of variants that can be obtained from Definition 2? These variants are not completely unrelated. For instance, well-supported models for $\mathbf{L}^{\mathcal{C}}$ are preserved for stronger logics or for more general assumption classes, as stated below.

Proposition 2. *Let M be a $\mathbf{L}^{\mathcal{C}}$-model of a Boolean program P. Then, for any logic $\mathbf{M} \supseteq \mathbf{L}$ and any $\mathcal{D} \supseteq \mathcal{C}$, M is also an $\mathbf{M}^{\mathcal{D}}$-model of P.* □

As we showed that $\mathbf{CL}^{at}$-models do not satisfy atom definability because $\{p\}$ is not a $\mathbf{CL}^{at}$-model of (1), by the proposition above, $\{p\}$ will not be a well-supported model of (1) in any weaker logic either, and so:

Corollary 1. *For any $\mathbf{L} \subseteq \mathbf{CL}$, $\mathbf{L}^{at}$-models do not satisfy atom definability.* □

The next result shows that, at least for intermediate logics, the remaining combinations for monotonic logics and where $\mathcal{C}$ includes at least the set of literals *lit*, eventually collapse into supported or equilibrium models.

Theorem 4. *Let P be a Boolean program and $\mathcal{C} \supseteq lit$. Then:*

(i) M is a $\mathbf{CL}^{\mathcal{C}}$-model of P iff M is a supported model of P.
(ii) For any intermediate logic $\mathbf{L} \subset \mathbf{CL}$:
M is a $\mathbf{L}^{\mathcal{C}}$-model of P iff $\langle M, M\rangle$ is an equilibrium model of P. □

That is, we obtain the same result as in Theorem 3, even if we use any non-classical intermediate logic $\mathbf{L}$ from $\mathbf{IL}$ to $\mathbf{HT}$. Again, (i) is not surprising since, for classical logic, the set Δ_M^{lit} fixes a unique model M and the same will happen for any $\mathcal{C} \supseteq lit$. So, strictly speaking, supported models are not well-supported, since they admit any arbitrary partial order relation $\prec$. This means that the only non-monotonic candidate from Definition 2 among intermediate logics that satisfies strict well-supportedness and atom definability is Equilibrium Logic.

5 Related Work and Conclusions

We have examined two properties, well-supportedness and atom definability, that we suggest might be taken as *desiderata* for a sound methodology for generalised logic programming based on the concept of stable model. Given certain assumptions and a range of possible underlying logics, it turns out that essentially only Equilibrium Logic satisfies both conditions. This may be seen as a new and strong argument in its favour.[2]

A related approach to generalising well-supportedness for Boolean programs is pursued in [10], proposing a modification of the so-called FLP-semantics of [8]. Though the approach we have taken here is related, it is less restrictive than that of [10], since assumptions there are restricted to negated atoms, Δ_M^{at} in our

[2] Many other properties of Equilibrium Logic, studied elsewhere, also speak in its favour, e.g. not least the characterisation of strong equivalence, [6].

notation, and logical inference is classical, based on $\models_{\mathbf{CL}}$. A fuller analysis and discussion of [10] is left for future work. However, we can already remark that the semantics proposed in [10] does not satisfy atom definability.

We plan to extend this analysis to other semantics for Boolean programs such as [17] and the ones studied in [18]. An important topic for future study is to provide a full, first-order logical account of these desiderata.

Acknowledgements. We are very thankful to the anonymous reviewers for their helpful comments and suggestions to improve the paper, especially for pointing out example after Theorem 1 which led to a more accurate reformulation.

References

1. Gelfond, M., Lifschitz, V.: The stable models semantics for logic programming. In: Proceedings of the 5th International Conference on Logic Programming, pp. 1070–1080 (1988)
2. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**, 365–385 (1991)
3. Inoue, K., Sakama, C.: Negation as failure in the head. J. Log. Program. **35**(1), 39–78 (1998)
4. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Ann. Math. Artif. Intell. **25**, 369–389 (1999)
5. Pearce, D.: A new logical characterisation of stable models and answer sets. In: Dix, J., Pereira, L.M., Przymusinski, T.C. (eds.) NMELP 1996. LNCS, vol. 1216, pp. 57–70. Springer, Heidelberg (1997). doi:10.1007/BFb0023801
6. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Log. **2**(4), 526–541 (2001)
7. Ferraris, P.: Answer sets for propositional theories. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS, vol. 3662, pp. 119–131. Springer, Heidelberg (2005). doi:10.1007/11546207_10
8. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: semantics and complexity. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS, vol. 3229, pp. 200–212. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30227-8_19
9. Truszczyński, M.: Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs. Artif. Intell. **174**(16), 1285–1306 (2010)
10. Shen, Y.D., Wang, K., Eiter, T., Fink, M., Redl, C., Krennwallner, T., Deng, J.: FLP answer set semantics without circular justifications for general logic programs. Artif. Intell. **213**, 1–41 (2014)
11. Fages, F.: Consistency of Clark's completion and existence of stable models. J. Methods Log. Comput. Sci. **1**(1), 51–60 (1994)
12. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: ASP-Core-2 input language format (2013). https://www.mat.unical.it/aspcomp2013/ASPStandardization
13. Eiter, T., Tompits, H., Woltran, S.: On solution correspondences in answer-set programming. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, Scotland, UK, pp. 97–102. Professional Book Center (2005)

14. Aguado, F., Cabalar, P., Fandinno, J., Pearce, D., Pérez, G., Vidal, C.: Forgetting auxiliary atoms in forks. In: Proceedings of the 10th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2017) (2017)
15. Clark, K.L.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Databases, pp. 293–322. Plenum Press, New York (1978)
16. McDermott, D.V., Doyle, J.: Non-monotonic logic I. Artif. Intell. **13**(1–2), 41–72 (1980)
17. Tasharrofi, S.: A rational extension of stable model semantics to the full propositional language. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI 2013, pp. 1118–1124. AAAI Press (2013)
18. Alviano, M., Faber, W.: Stable model semantics of abstract dialectical frameworks revisited: a logic programming perspective. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 2684–2690 (2015)