Resolution and Logic Programming in Algebraic Domains: Negation and Defaults

Pascal Hitzler

Artificial Intelligence Institute, Department of Computer Science
Dresden University of Technology, Dresden, Germany
phitzler@inf.tu-dresden.de, www.wv.inf.tu-dresden.de/∼pascal/

Technical Report WV–02–05

Abstract

W.C. Rounds and G.-Q. Zhang have recently proposed to study a form of disjunctive logic programming generalized to algebraic domains [RZ01]. This system allows reasoning with information which is hierarchically structured and forms a (suitable) domain. We extend this framework to include reasoning with *negative information*, i.e. the implicit or explicit *absence* of bits of information. These investigations will naturally lead to a form of default reasoning which is strongly related to logic programming with answer sets or stable models, which has recently created much interest amongst artificial intelligence researchers concerned with knowledge representation and reasoning.

Contents

1 Introduction

In [RZ01], Rounds and Zhang propose to study a form of clausal logic generalized to algebraic domains. In essence, they propose to interpret finite sets of compact elements as clauses, and develop a theory which links corresponding logical notions to topological notions on the domain. Amongst other things, they establish a sound and complete resolution rule and a form of disjunctive logic programming in domains, based on material implication. A corresponding semantic operator turns out to be Scott-continuous.

In this technical report, we will extend this paradigm to include reasoning with negation. We are motivated by the gain in expressiveness through the use of negation in artificial intelligence paradigms related to knowledge representation and nonmonotonic reasoning. We will in fact follow two approaches to negation, the first coming from classical logic, and the second from default logic. In classical (propositional) logic, negation can be understood as an involution on propositional variables. From the domain-theoretic point of view which we will adopt later, this corresponds to restrictions on the domains, i.e. we will allow only domains which provide an involution which can be understood as a negation. The second approach, using ideas from default logic, treats negation as a meta-logical supplement: The negation of an item is believed if there is no reason to believe the item itself. In the theory of logic programming, this point of view is strongly related to the treatment of negation as failure, and research in the theory of logic programming and nonmonotonic reasoning has lately led to the development of applications in the form of nonmonotonic reasoning systems known as *answer set programming* (cf. [Lif99, MT99] for accounts of this).

The paper is structured as follows. In Section 2 we review the most fundamental definitions for Rounds' and Zhang's *clausal logic in algebraic domains*, as laid out in [RZ01]. In Section 3 we will study a form of resolution for this framework as proposed in [RZ01]. In particular, we will provide a system consisting of three rules which is sound and complete with respect to resolution, but is simpler and easier to work with. The remainder of Section 3 will be devoted to establishing sufficient conditions which allow to carry over a main result on resolution from classical logic to algebraic domains: $T \models X$ if and only if $T \cup \{\neg X\} \vdash \emptyset$, cf. Theorem 3.12. It turns out that we need to impose severe restrictions on the domain in order to establish this, namely (1) a way of providing normal forms for clauses, and therefore for proofs (studied in Section 3.1), and (2) an involution which can be understood as "classical" negation (studied in Section 3.2). In Section 4, we move on to study logic programming in algebraic domains as proposed in [RZ01]. While the negation studied in Section 3 is implicitly given by the domain, we proceed in this chapter to adjoin a form of default negation to this programming paradigm. The exposition will naturally lead to establishing a form of well-founded semantics for these programs, as well as a notion of stable models, both very strongly related to their counterparts in the classical logic programming paradigm.

The emphasis of this technical report is to propose definitions and constructions, not to present deep results, although we have taken care to include just enough examples and results as to justify the constructions.

Acknowledgements. I would like to thank William C. Rounds, Anthony K. Seda, Pawel Waszkiewicz, and Guo-Qiang Zhang for helpful discussions on the general subject matter.

2 Preliminaries: Clausal Logic in Algebraic Domains

A *partially ordered set* is a pair $(D, \sqsubseteq)$, where D is a nonempty set and $\sqsubseteq$ is a relexive, antisymmetric, and transitive relation on D. A subset X of a partially ordered set is *directed* if for all $x, y \in X$ there is $z \in X$ with $x, y \sqsubseteq z$. An *ideal* is a directed and downward closed set. A *complete partial order*, *cpo* for short, is a partially ordered set $(D, \sqsubseteq)$ with a least element $\perp$, called the *bottom element* of $(D, \sqsubseteq)$, and such that every directed set in D has a least upper bound, or supremum, $\bigsqcup D$. An element $c \in D$ is said to be *compact* or *finite* if whenever $c \sqsubseteq \bigsqcup L$ with L directed, then there exists $e \in L$ with $c \sqsubseteq e$. The set of all compact elements of a cpo D is written as $\mathsf{K}(D)$. An *algebraic cpo* is a cpo such that every $e \in D$ is the directed supremum of all compact elements below it.

A set $U \subseteq D$ is said to be *Scott open*, or just *open*, if it is upward closed and for any directed $L \subseteq D$ we have $\bigsqcup L \in U$ if and only if $U \cap L \neq 0$. The *Scott topology* on D is the topology whose open sets are all Scott open sets. An open set is *compact open* if it is compact in the Scott topology. A *coherent algebraic cpo* is an algebraic cpo such that the intersection of any two compact open sets is compact open. We will not make use of many topological notions in the sequel. So let us just note that coherency of an algebraic cpo implies that the set of all minimal upper bounds of a finite number of compact elements is finite, i.e. if $c_1, \ldots, c_n$ are compact elements, then the set $\mathsf{mub}\{c_1, \ldots, c_n\}$ of minimal upper bounds of these elements is finite. Note that $\mathsf{mub}\,\emptyset = \{\perp\}$, where $\perp$ is the least element of D.

In the following, $(D, \sqsubseteq)$ will always be assumed to be a coherent algeraic cpo. We will also call these spaces *domains*.

Following [Joh82], an element $a \in D$ is called an *atom*, or an *atomic element*, if whenever $x \sqsubseteq a$ we have $x = a$ or $x = \perp$. The set of all atoms of a domain is denoted by $\mathsf{A}(D)$.

2.1 Definition Let D be a coherent algebraic cpo with set $\mathsf{K}(D)$ of compact elements. A *clause* is a finite subset of $\mathsf{K}(D)$. We denote the set of all clauses ove D by $\mathcal{C}(D)$. If X is a clause and $w \in D$, we write $w \models X$ if there exists $x \in X$ with $x \sqsubseteq w$, i.e. X contains an element below w.

A *theory* is a set of clauses, which may be empty. An element $w \in D$ is a *model* of a theory T, written $w \models T$, if $w \models X$ for all $X \in T$ or, equivalently, if every clause $X \in T$ contains an element below w.

A clause X is called a *logical consequence* of a theory T, written $T \models X$, if $w \models T$ implies $w \models X$. If $T = \{E\}$, then we write $E \models X$ for $\{E\} \models X$. Note that this holds if and only if for every $w \in E$ there is $x \in X$ with $x \sqsubseteq w$.

For two theories T and S, we say that $T \models S$ if $T \models X$ for all $X \in S$. We say that T and S are (*logically*) *equivalent*, written $T \sim S$, if $T \models S$ and $S \models T$. In order to avoid confusion, we will throughout denote the empty clause by $\{\}$, and the empty theory by $\emptyset$. A theory T is *closed* if $T \models X$ implies $X \in T$ for all clauses X. It is called *consistent* if $T \not\models \{\}$ or, equivalently, if there is w with $w \models T$.

A main technical result from [RZ01] shows that the set of all consistent closed theories over D, ordered by inclusion, is isomorphic to the collection of all non-empty Scott-compact saturated subsets of D, ordered by reverse inclusion. This result rests on the Hofmann-Mislove theorem, and we refer the reader to [RZ01] for details. It follows as a corollary that a theory is logically closed if and only if it is an ideal,[1] and also that a clause is a logical consequence of a theory T if and only if it is a logical consequence of a finite subset of T. The latter is a compactness theorem for clausal logic in algebraic domains.

2.2 Example In [RZ01], the following running example was given. Consider Kleene's strong three-valued logic in the propositional case[2], with the usual (knowledge)-ordering on the set $\mathbb{T} = \{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$ of truth values given by $\mathbf{u} \leq \mathbf{f}$ and $\mathbf{u} \leq \mathbf{t}$. This induces a pointwise ordering on the space $\mathbb{T}^{\mathcal{V}}$ of all interpretations (or *partial truth assignments*), where $\mathcal{V}$ is the (countably infinite) set of all propositional variables in the language under consideration. The partially ordered set $\mathbb{T}^{\mathcal{V}}$ is a coherent algebraic cpo[3]. Compact elements in $\mathbb{T}^{\mathcal{V}}$ are those interpretations which map all but a finite number of propositional variables to $\mathbf{u}$. We denote compact elements by strings such as $pq\overline{r}$, which indicates that p and q are mapped to $\mathbf{t}$ and r is mapped to $\mathbf{f}$.

We note that $\{e \mid e \models \phi\}$ is upward-closed for any formula ϕ. A clause in $\mathbb{T}^{\mathcal{V}}$ is a formula in disjunctive normal form, e.g. $\{pq\overline{r}, \overline{p}q, r\}$ translates to $(p \wedge q \wedge \neg r) \vee (\neg p \wedge q) \vee r$.

We also note that every compact element in $\mathbb{T}^{\mathcal{V}}$ can be uniquely expressed as the supremum of a finite number of atomic elements, and the set of all atomic elements is $\mathsf{A}\left(\mathbb{T}^{\mathcal{V}}\right) = \mathcal{V} \cup \{\overline{v} \mid v \in \mathcal{V}\}$. Furthermore, there exists a bijective function $^{-} : \mathsf{A}\left(\mathbb{T}^{\mathcal{V}}\right) \to \mathsf{A}\left(\mathbb{T}^{\mathcal{V}}\right) : p \to \overline{p}$ which extends naturally to a Scott-continuous involution onto all of $\mathbb{T}^{\mathcal{V}}$ via $\overline{p_1 \dots p_n} = \overline{p_1} \dots \overline{p_n}$. In the following, a clause over a domain D will be called an *atomic clause* if it is a finite subset of $\mathsf{A}(D)$. Atomic clauses on $\mathbb{T}^{\mathcal{V}}$ correspond to propositional clauses in the usual sense. Note that $p \not\uparrow \overline{p}$ for $p \in \mathsf{A}\left(\mathbb{T}^{\mathcal{V}}\right)$ and in general for all $c \in \mathsf{K}\left(\mathbb{T}^{\mathcal{V}}\right)$ we have $c \not\uparrow \overline{c}$.

3 Resolution in Algebraic Domains

In [RZ01], a sound and complete resolution rule, called *clausal hyperresolution*, was given as follows, where $\{X_1, \dots, X_n\}$ is a clause set and Y a clause, and $\mathsf{mub}\{a_1, \dots, a_n\}$ denotes the set of all minimal upper bounds of all the a_i's, which is a finite set of compact elements by algebraic coherence, i.e. a clause.

$$\frac{X_1 \quad X_2 \quad \dots \quad X_n; \qquad a_i \in X_i \text{ for } 1 \leq i \leq n; \qquad \mathsf{mub}\{a_1, \dots, a_n\} \models Y}{Y \cup \bigcup_{i=1}^{n} (X_i \setminus \{a_i\})} \qquad (\mathsf{hr})$$

This rule is sound in the following sense: Whenever $w \models X_i$ for all i, then for any admissible choice of the a_i and Y in the antecedent, we have $w \models Y \cup \bigcup_{i=1}^{n} (X_i \setminus \{a_i\})$.

For completeness, it is necessary to adjoin to the above clausal hyperresolution rule a special rule which allows the inference of any clause from the empty clause. We indicate this rule as follows.

$$\frac{\{\}; \qquad Y \in \mathcal{C}(D)}{Y} \qquad (\mathsf{spec})$$

[1] An ideal with respect to the *Smyth preorder* $\sqsubseteq^\sharp$, where $X \sqsubseteq^\sharp Y$ if and only if for every $y \in Y$ there exists some $x \in X$ with $x \sqsubseteq y$.

[2] Cf. [Fit85] for a discussion of this in the context of logic programming semantics and [Plo78] for a domain-theoretic context.

[3] In fact it is also consistently complete.

With this addition, given a theory T and a clause X with $T \models X$, we have that $T \vdash^* X$, where $\vdash^*$ stands for a finite number of applications of the clausal hyperresolution rule together with the special rule[4].

Furthermore, [RZ01, Remark 4.6] shows that binary hyperresolution, together with (spec), is already complete, i.e. the system consisting of the *binary clausal hyperresolution* rule

$$\frac{X_1 \quad X_2; \qquad a_1 \in X_1 \quad a_2 \in X_2; \qquad \mathsf{mub}\{a_1, a_2\} \models Y}{Y \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})} \qquad (\mathsf{bhr})$$

together with the special rule is sound and complete.

If the set $\{a_1, \dots, a_n\}$ is inconsistent, then $\mathsf{mub}\{a_1, \dots, a_n\} = \{\}$. Since $\{\} \models \{\}$, clausal hyperresolution generalizes the usual notion of resolution, given by the following rule.

$$\frac{X_1 \quad X_2; \qquad a_1 \in X_1 \quad a_2 \in X_2; \qquad a_1 \not\uparrow a_2}{(X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})} \qquad (\mathsf{r})$$

We note that the special rule (spec) can be understood as an instance of (ext) (see Footnote 4). Note that resolution (r) together with (ext) and (red) (see again Footnote 4) is not complete. In order to see this, we refer again to Example 2.2. Let $T = \{\{p\}, \{q\}\}$ and $X = \{pq\}$. Then $T \models X$ but there is no way to produce X from T using (r), (ext) and (red) alone. Indeed, it is easy to show by induction that any X which can be derived from T by using only (r), (ext) and (red), contains either p or q, which suffices.

It is our desire, however, to give a sound and complete system which is as simple as possible. Consider the following rule, which we call *simplified hyperresolution*. It is easy to see that it is an instance of (hr) and more general than (r).

$$\frac{X_1 \quad X_2; \qquad a_1 \in X_1 \quad a_2 \in X_2}{\mathsf{mub}\{a_1, a_2\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})} \qquad (\mathsf{shr})$$

3.1 Theorem The system consisting of (shr), (ext) and (red) is complete.

Proof: In order to show completeness, we derive (bhr) from (shr), (ext) and (red). Let X_1, X_2 be given with $a_1 \in X_1$ and $a_2 \in X_2$ with $a_1 \uparrow a_2$. Furthermore, let Y be a clause with $\mathsf{mub}\{a_1, a_2\} \models Y$. Let $\mathsf{mub}\{a_1, a_2\} = \{b_1, \dots, b_n\}$. Then for every b_i there exists $y_i \in Y$ with $y_i \sqsubseteq b_i$. Using (shr), from X_1 and X_2 we can derive $X_3 = \mathsf{mub}\{a_1, a_2\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})$,

[4]There seems to be a slight technical problem in the proof of [RZ01, Lemma 4.4], which states that $E \models X$ implies $E \vdash^* X$ for non-empty clause E and clause X. With notation from there, and working with the three-valued propositional setting from Example 2.2, we note that $E = \{pq\} \models \{p\} = X$, and $pq \in E \setminus X$, but $pq \not\sqsubseteq p$. The result, however, holds. First note that two special instances of the clausal hyperresolution rule are as follows, called the *reduction rule* and the *extension rule*.

$$\frac{X; \qquad \{a, y\} \subseteq X; \qquad y \sqsubseteq a}{X \setminus \{a\}} \quad (\mathsf{red}), \qquad \frac{X; \qquad y \in \mathsf{K}(D)}{\{y\} \cup X} \quad (\mathsf{ext})$$

Indeed, the first rule follows from (hr) since $a \in X$ and $\{a\} \models \{y\}$, while the latter rule follows since $\{a\} \models \{a, y\}$ for all $y \in \mathsf{K}(D)$. Now suppose E is a non-empty clause, and X a clause with $E \models X$. Then for all $f \in E$ there is $x \in X$ with $x \sqsubseteq f$. Using the second rule above, we can first extend E to $E \cup X$ and obtain $E \vdash^* E \cup X$. Then, using the first rule, we can remove all $f \in E \setminus X$ from $E \cup X$, obtaining $E \cup X \vdash^* X$, hence $E \vdash^* X$.

and with repeated application of (ext) and (red) we obtain from this $X_4 = \{y_1, \ldots, y_n\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})$. Finally, using (ext) repeatedly, we can add to X_4 all remaining elements from Y. The argumentation for $a_1 \not\uparrow a_2$ is similar. This completes the proof. ■

We note that a rule with weaker preconditions than (red) suffices, which we call the *weakening rule*:

$$\frac{X; \qquad a \in X; \qquad y \sqsubseteq a}{\{y\} \cup (X \setminus \{a\})} \quad \mathsf{(w)}$$

Indeed, (red) can be derived from (w) as follows. Let $\{a, y\} \subseteq X$ with $y \sqsubseteq a$. Then in particular $a \in X$, i.e. using (w) we can derive $\{y\} \cup (X \setminus \{a\})$ which is equal to $X \setminus \{a\}$ since y is already contained in X. On the other hand, (w) can be derived from (red) and (ext) as follows. Let $a \in X$ and $y \sqsubseteq a$. If $a = y$ then there is nothing to show, so assume $a \neq y$. Then $X \vdash X \cup \{y\}$ by the extension rule, so the reduction rule can be applied, yielding $(X \cup \{y\}) \setminus \{a\}$ as required.

The following technical result is inspired by [CZ00, Theorem 7].

3.2 Proposition For clauses $X_1, \ldots, X_n$ we have $\{X_1, \ldots, X_n\} \models X$ if and only if $\{\{a_1\}, \ldots, \{a_n\}\} \models X$ for all $(a_1, \ldots, a_n) \in X_1 \times \ldots \times X_n$.

Proof: Assume $\{X_1, \ldots, X_n\} \models X$ and let $a_i \in X_i$ be arbitrarily chosen for $i = 1, \ldots, n$. Then $\{a_i\}$ X_i for all $i = 1, \ldots n$ by (ext) and therefore $\{\{a_1\}, \ldots, \{a_n\}\} \models \{X_1, \ldots, X_n\} \models X$.

Conversely, assume that $\{\{a_1\}, \ldots, \{a_n\}\} \models X$ for all $(a_1, \ldots, a_n) \in X_1 \times \ldots \times X_n$ and let $w \in D$ with $w \models \{X_1, \ldots, X_n\}$, i.e. $w \models X_i$ for all $i = 1, \ldots, n$. Then for all $i = 1, \ldots, n$ there is $a_i \in X_i$ with $a_i \sqsubseteq w$. So for all $i = 1, \ldots, n$ choose a_i with $a_i \sqsubseteq w$. Then $w \models \{\{a_1\}, \ldots, \{a_n\}\}$ and by assumption we obtain $w \models X$. ■

We call the system consisting of the rules (red), (ext) and (shr) the RAD system, from *Resolution in Algebraic Domains*. For two theories T and S, we write $T \vdash^* S$ if $T \vdash^* A$ for each $A \in S$, and for clauses X and Y we write $X \vdash^* Y$, respectively $X \vdash^* T$, for $\{X\} \vdash^* Y$, respectively $\{X\} \vdash^* T$. The symbol $\vdash$ denotes derivation by a single application of one of the rules in RAD. With slight abuse of notation, for two theories T and S we allow to write $T \vdash S$ if $T \vdash X$ for some clause X and $S \subseteq T \cup \{X\}$.

We interpret the RAD rules in the setting of Example 2.2. We already know that clauses correspond to formulas in disjunctive normal form (DNF), and theories to sets of DNF formulas. The weakening rule acts on single clauses and replaces a conjunction contained in a DNF formula by a conjunction which contains a subset of the propositional variables contained in the original conjunction, e.g. $(p \wedge q) \vee r$ becomes $p \vee r$. The extension rule disjunctively extends a DNF formula by a further conjunction of propositional variables, e.g. $(p \wedge q) \vee r$ becomes $(p \wedge q) \vee r \vee (s \wedge q)$. The simplified hyperresolution rule finally takes two DNF formulas, deletes one conjunction from each of them, and forms a disjunction from the resulting formulas together with the conjunction of the deleted items, e.g. $(p \wedge q) \vee r$ and $\neg p \vee (s \wedge r)$ can be resolved to $(p \wedge q) \vee (r \wedge \neg p) \vee (s \wedge r)$.

A more abstract interpretation of the RAD system comes from a standard intuition underlying domain theory. Elements of the domain D are interpreted as pieces of information, and if $x \sqsubseteq y$, this represents that y contains more information than x. Compact elements are understood as items which are computationally accessible. From this point of view, RAD gives a calculus for reasoning about disjunctive information in computation, taking a clause, i.e. a finite set of computationally accessible information items as disjunctive knowledge about these items. The rules from RAD yield a system for deriving further knowledge from the given disjunctive information. The weakening rule states that we can replace an item by another one which contains less information. The extension rule states that we can always extend our knowledge disjunctively with further bits of information. Both rules decrease our knowledge. The simplified hyperresolution rule states that we can disjunctively merge two collections of disjunctive information, while strengthening our knowledge by replacing two of the items from the collections by an item which contains both pieces of information, and deleting the original items.

3.1 Atomic Domains

We simplify proof search via resolution by requiring stronger conditions on the domain. We will be guided by Example 2.2.

3.3 Definition An *atomic domain* is a coherent algebraic cpo D with the following property: For all $c \in \mathsf{K}(D)$, the set $\mathsf{A}(c) = \{p \in \mathsf{A}(D) \mid p \sqsubseteq c\}$ is finite and $c = \bigsqcup \mathsf{A}(c)$.

The domain $\mathbb{T}^{\mathcal{V}}$ from Example 2.2 is an example of an atomic domain. In the remainder of this section, D will always be an atomic domain.

We seek to represent a clause X by a finite set $\mathsf{A}(X)$ of atomic clauses which is logically equivalent to X. Given $X = \{a_1, \dots, a_n\}$, we define $\mathsf{A}(X)$ as follows.

$$\mathsf{A}(X) = \{\{b_1, \dots, b_n\} \mid b_i \in \mathsf{A}(a_i) \text{ for all } i = 1, \dots, n\}$$

3.4 Lemma Let $X = \{a_1, \dots, a_n\}$ be a clause. Let $X/a_1 = \{\{b, a_2, \dots, a_n\} \mid b \in \mathsf{A}(a_1)\}$. Then $X/a_1 \models X$.

Proof: Since $\bigsqcup \mathsf{A}(a_1) = a_1$, and therefore $\mathsf{mub}\,\mathsf{A}(a_1) \models \{a_1\}$, we obtain $X/a_1 \vdash^* X$ from (hr), and the assertion follows from the soundness of RAD. ■

3.5 Lemma For any clause X we have $\mathsf{A}(X) \sim X$.

Proof: Let $X = \{a_1, \dots, a_n\}$ and let $Y = \{b_1, \dots, b_n\} \in \mathsf{A}(X)$ with $b_i \in \mathsf{A}(a_i)$ for all i. Then $b_i \sqsubseteq a_i$ for all i and hence $X \vdash^* Y$ by repeated application of the weakening rule.

Conversely, we define for any compact element a and any set T of clauses: $T/a = \{Z \in T \mid a \notin Z\} \cup \{\{b\} \cup (Z \setminus \{a\}) \mid b \in \mathsf{A}(a), a \in Z \in T\}$. With notation from Lemma 3.4 and for any clause Z and $a \in Z$ we have $\{Z\}/a = Z/a$. So, from Lemma 3.4 we obtain that $T/a \models T$ for all sets of clauses T and $a \in \mathsf{K}(D)$. Now let $X = \{a_1, \dots, a_n\}$. Then $(\dots(X/a_1)/a_2 \dots)/a_n = \mathsf{A}(X)$ and consequently $\mathsf{A}(X) \models X$. ■

3.6 Lemma Let X be a clause and T a theory. Then $T \models X$ if and only if $T \models Y$ for all $Y \in \mathsf{A}(X)$.

Proof: Suppose $T \models X$ and let $Y \in \mathsf{A}(X)$. Then $\{X\} \vdash^* Y$ by Lemma 3.5. So $T \vdash^* Y$ and $T \models Y$.

Conversely, suppose $T \models Y$ for all $Y \in \mathsf{A}(X)$, i.e. $T \vdash^* Y$ for all $Y \in \mathsf{A}(X)$. But $\mathsf{A}(X) \vdash^* X$, so $T \vdash^* X$ and therefore $T \models X$. ∎

3.7 Theorem Let X be a clause and T a theory. Then $T \models X$ if and only if $T \vdash^* Y$ for all $Y \in \mathsf{A}(X)$.

Proof: This follows immediately from Lemma 3.6 and the completeness of RAD. ∎

In view of Theorem 3.7, it suffices to study $T \vdash^* X$ for theories T and atomic clauses X. We can actually obtain a stronger result, as follows, which provides some kind of normal forms of derivations. For a theory T, define $\mathsf{A}(T) = \{\mathsf{A}(X) \mid X \in T\}$.

3.8 Theorem Let D be an atomic domain, T be a theory, X be a clause and

$$T \vdash T_1 \vdash \cdots \vdash T_N \vdash X$$

be a derivation in RAD. Then there exists a derivation

$$\mathsf{A}(T) \vdash^* \mathsf{A}(T_1) \vdash^* \cdots \vdash^* \mathsf{A}(T_N) \vdash^* \mathsf{A}(X)$$

using only the *atomic extension rule*

$$\frac{X; \qquad y \in \mathsf{A}(D)}{\{y\} \cup X} \quad (\mathsf{axt})$$

and the *multiple atomic shift rule*

$$\frac{a_1 \in X_1 \quad \ldots \quad a_n \in X_n; \qquad \mathsf{mub}\{a_1, \ldots, a_n\} = \{x_1, \ldots, x_k\}, b_i \in \mathsf{A}(x_i) \text{ for all } i}{\{b_1, \ldots, b_k\} \cup \bigcup_{i-1}^{n} (X_i \setminus \{a_i\})} \quad (\mathsf{mas}).$$

Furthermore, all clauses occuring in the derivation are atomic.

Proof: Let X_1, X_2, X be clauses. We distinguish three cases, from which the assertion follows easily by induction on N.

1. $X_1 \vdash X$ using the reduction rule. First note that the following *atomic shift rule* is a special instance of the multiple atomic shift rule.

$$\frac{a_1 \in X_1 \quad a_2 \in X_2; \qquad a \in \mathsf{A}(x) \text{ for all } x \in \mathsf{mub}\{a_1, a_2\}}{\{a\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})} \quad (\mathsf{ash})$$

Indeed, (ash) follows from (mas) with $n = 2$ and $a = b_1 = \ldots = b_k$. Now let $a, y \in X_1$ with $y \sqsubseteq a$ and $X = X_1 \setminus \{a\} = \{y, x_1, \ldots, x_n\}$. Let $A \in \mathsf{A}(X)$, say $A = \{y', x'_1, \ldots, x'_n\}$ with $y' \in \mathsf{A}(y)$ and $x'_i \in \mathsf{A}(x_i)$ for all i. Without loss of generality we can assume that $\mathsf{A}(y) \subset \mathsf{A}(a)$, so there is $\{a'\} \cup A \in \mathsf{A}(X_1)$ for some $a' \in \mathsf{A}(a) \setminus \mathsf{A}(y)$. So we now have $a', y' \sqsubseteq a$ and $y' \sqsubseteq y$, i.e. $\{y', a', x'_1, \ldots, x'_n\} \in \mathsf{A}(X_1)$ and $\{y', y', x'_1, \ldots, x'_n\} = A \in \mathsf{A}(X_1)$. So $a' \in \{y', a', x'_1, \ldots, x'_n\}$, $y' \in \{y', y', x'_1, \ldots, x'_n\}$ and since $y' \sqsubseteq x$ for all $x \in \mathsf{mub}\{y', a'\}$ we can derive $\{y'\} \cup (\{y', a', x'_1, \ldots, x'_n\} \setminus \{a'\}) \cup (\{y', y', x'_1, \ldots, x'_n\} \setminus \{y'\}) = \{y', x'_1, \ldots, x'_n\} = A$ using the atomic shift rule.

2. $X_1 \vdash X$ using the extension rule, i.e. $X = X_1 \cup \{y\}$ for some y. Let $A \in \mathsf{A}(X)$. Then $A = \{y'\} \cup Y$ for some $y' \in \mathsf{A}(y)$ and $Y \in \mathsf{A}(X_1)$. Using the atomic extension rule we can derive $Y \vdash A$ and therefore $\mathsf{A}(X_1) \vdash A$ using the atomic extension rule only, which suffices.

3. $\{X_1, X_2\} \vdash X$ using the simplified hyperresolution rule. Let $a_1 \in X_1$, $a_2 \in X_2$ and $X = \mathsf{mub}\{a_1, a_2\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})$. Furthermore, let $M = \mathsf{mub}\{a_1, a_2\} = \{m_1, \ldots, m_k\}$ and let $A \in \mathsf{A}(X)$, i.e. $A = \{m'_1, \ldots, m'_k\} \cup B_1 \cup B_2$, where $m'_i \in \mathsf{A}(m_i)$ for all i, $B_1 \in \mathsf{A}(X_1 \setminus \{a_1\})$ and $B_2 \in \mathsf{A}(X_2 \setminus \{a_2\})$. Note that for all $a'_1 \in \mathsf{A}(a_1)$ we have that $B_1 \cup \{a'_1\} \in \mathsf{A}(X_1)$ and for all $a'_2 \in \mathsf{A}(a_2)$ we have that $B_2 \cup \{a'_2\} \in \mathsf{A}(X_2)$. Let $\mathsf{A}(a_1) = \{a'_1, \ldots, a'_{k_1}\}$ and $\mathsf{A}(a_2) = \{a'_{k_1+1}, \ldots, a'_{k_1+k_2}\}$. For $i = 1, \ldots, k_1$ let $Y_i = B_1 \cup \{a'_i\} \in \mathsf{A}(X_1)$ and for $i = k_1, \ldots, k_1 + k_2$ let $Y_i = B_2 \cup \{a'_i\} \in \mathsf{A}(X_2)$. Since $a_1 = \bigsqcup \mathsf{A}(a_1)$ and $a_2 = \bigsqcup \mathsf{A}(a_2)$ we have $\mathsf{mub}\left(\mathsf{A}(a_1) \cup \mathsf{A}(a_2)\right) = \mathsf{mub}\{a_1, a_2\} = \{m_1, \ldots, m_k\} = M$. From the multiple atomic shift rule we obtain

$$\frac{a_i \in Y_i \quad (i = 1, \ldots, k_1 + k_2) \qquad \mathsf{mub}\{a'_1, \ldots, a'_{k_1+k_2}\} = M, m'_j \in \mathsf{A}(m_j) \quad (j = 1, \ldots, k)}{\{m'_1, \ldots, m'_k\} \cup \bigcup (Y_i \setminus \{a_i\})}$$

Since $Y_i \setminus \{a'_i\} \subseteq B_1$ for $i = 1, \ldots, k_1$ and $Y_i \setminus \{a'_2\} \subseteq B_2$ for $i = k_1, \ldots, k_1 + k_2$, we obtain $\{m'_1, \ldots, m'_k\} \cup \bigcup (Y_i \setminus \{a_i\}) \subseteq A$ which suffices by the atomic extension rule. ■

Note that the atomic extension rule is a special case of the extension rule, and that the multiple atomic shift rule can be obtained as a subsequent application of first the hyperresolution rule (with $Y = \mathsf{mub}\{a_1, \ldots, a_n\}$) and then multiple instances of the reduction rule, hence both rules are sound.

3.9 Remark We note that Theorem 3.8 does not hold if (mas) is replaced by its binary version

$$\frac{a_1 \in X_1, a_2 \in X_2; \qquad \mathsf{mub}\{a_1, a_2\} = \{x_1, \ldots, x_k\}, b_i \in \mathsf{A}(x_i) \text{ for all } i}{\{b_1, \ldots, b_k\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})} \quad (\mathsf{bas}).$$

In order to see this, consider three atomic elements a_1, a_2, a_3 which are mutually consistent with supremum $\sup\{a_i, a_j\} = a_{ij}$, but do not have a common upper bound. Then $\{\{a_1\}, \{a_2\}, \{a_3\}\} \models \{\}$, but the empty clause $\{\}$ can not be derived from the theory $T = \{\{a_1\}, \{a_2\}, \{a_3\}\}$ using (axt) and (bas) alone. Indeed it is easy to show by induction that every clause which is derived from T using applications of (axt) and (bas) always contains one of the elements a_1, a_2 or a_3.

3.2 Domains with Negation

We introduce and investigate a notion of negation on domains, motivated by classical negation as in Example 2.2.

3.10 Definition An atomic domain is called an *atomic domain with negation* if there exists an involutive and Scott-continuous *negation function* $^- : D \to D$ with the following properties:

(i) $^-$ maps $\mathsf{A}(D)$ onto $\mathsf{A}(D)$.

(ii) For all $p, q \in \mathsf{A}(D)$ we have $p \not\uparrow q$ if and only if $q = \overline{p}$.

(iii) For every finite subset $A \subseteq \mathsf{A}(D)$ such that $p \uparrow q$ for all $p, q \in A$, the supremum $\bigsqcup A$ esists.

$\mathbb{T}^{\mathcal{V}}$ from Example 2.2 is an example of an atomic domain with negation.

3.11 Proposition Let D be an atomic domain with negation. Then for all $c \in \mathsf{K}(D)$ we have $\overline{c} = \bigsqcup\{\overline{a} \mid a \in \mathsf{A}(c)\}$.

Proof: Let $c \in \mathsf{K}(D)$. Then $c = \bigsqcup \mathsf{A}(c)$, hence $\mathsf{A}(c)$ is consistent. By (ii) of Definition 3.10, we obtain that every pair of elements from $\{\overline{a} \mid a \in \mathsf{A}(c)\}$ is consistent, and by (iii) the supremum $d = \bigsqcup\{\overline{a} \mid a \in \mathsf{A}(c)\}$ exists. From monotonicity of $^-$, we obtain first $d \sqsubseteq \overline{c}$, and then $\overline{d} \sqsubseteq \overline{\overline{c}} = c$. But, again by monotonicity of $^-$, we know that $\overline{d}$ is an upper bound of $\mathsf{A}(c)$, hence $c \sqsubseteq \overline{d}$, and consequently $c = \overline{d}$ and $\overline{c} = d = \bigsqcup\{\overline{a} \mid a \in \mathsf{A}(c)\}$ as required. ■

The following result allows one to replace the search for derivations by proof search, as in the classical form of resolution.

3.12 Theorem Let D be an atomic domain with negation. Let T be a theory and X be an atomic clause. Then $T \models X$ if and only if $T \cup \{\{\bar{a}\} \mid a \in X\} \vdash^* \{\}$.

Proof: Assume $T \models X$. Then $T \vdash^* X$ and $\{X\} \cup \{\{\bar{a}\} \mid a \in X\} \vdash^* \{\}$ follows easily by repeated application of the resolution rule (r).

Conversely, assume $T \cup \{\{\overline{a}\} \mid a \in X\} \vdash^* \{\}$, i.e. $T \cup \{\{\overline{a}\} \mid a \in X\} \models \{\}$. If $T \models \{\}$ then $T \vdash^* \{\} \vdash^* X$. So assume that $T \not\models \{\}$, i.e. there exists $w \in D$ with $w \models T$. We have to show that $w \models X$ for every such w. Since $w \models T$ but $w \not\models T \cup \{\{\overline{a}\} \mid a \in X\}$, we have that there is $a \in X$ with $\overline{a} \not\uparrow w$. Hence there exists $x \in \mathsf{A}(w)$ with $x \not\uparrow \overline{a}$. From the hypothesis we obtain $x = a$. Hence $a \sqsubseteq w$ and therefore, by the weakening rule, $w \vdash^* X$, i.e. $w \models X$. ■

On atomic domains with negation, we can therefore establish the following sound and complete proof principle.

3.13 Theorem Let T be a theory and X a clause. Consider $T' = \mathsf{A}(T)$. For every atomic clause $A \in \mathsf{A}(X)$ attempt to show $T' \cup \{\{\overline{a}\} \mid a \in A\} \vdash^* \{\}$ using (axt) and (mas). If this succeeds, then $T \models X$. Conversely, if $T \models X$ then there exists a derivation $T' \cup \{\{\overline{a}\} \mid a \in A\} \vdash^* \{\}$ for each $A \in \mathsf{A}(X)$ using only the above mentioned rules.

Proof: If $T' \cup \{\{\overline{a}\} \mid a \in A\} \vdash^* \{\}$, then by Theorem 3.8 the derivation can be carried out using only the mentioned rules and we obtain $T' \cup \{\{\overline{a}\} \mid a \in A\} \models \{\}$. By Theorem 3.12 we obtain $T' \models A$, so $T' \models A$ for all $A \in \mathsf{A}(X)$. By Lemma 3.6 this yields $T' \models X$ and finally we obtain $T \models X$ by application of Lemma 3.5, noting that $T' = \mathsf{A}(T) \sim T$.

Conversely, if $T \models X$ then we have $T' \models A$ for all $A \in \mathsf{A}(X)$, again by Lemmata 3.5 and 3.6. Theorem 3.12 then yields $T' \cup \{\{\overline{a}\} \mid a \in A\} \vdash^* \{\}$ for all $A \in \mathsf{A}(X)$, and finally from Theorem 3.8 we obtain that this derivation can be done using only the designated rules. ■

4 Logic Programming in Algebraic Domains

Following the lead from [RZ01], we now move on to study disjunctive logic programming in algebraic domains. Our aim is to extend this paradigm with a kind of default negation.

4.1 Definition A *disjunctive logic program* over D is a set P of *rules* of the form $Y \leftarrow X$, where X, Y are clauses over D. An element $e \in D$ is said to be a *model* of P if for every rule $Y \leftarrow X$ in P, if $e \models X$, then $e \models Y$. A clause Y is a *logical consequence* of P if every model of P satisfies Y. We write $\mathsf{cons}(P)$ for the set of all clauses which are logical consequences of P. If T is a theory, we write $\mathsf{cons}(T)$ for the set of all clauses which are logical consequences of T. Note that the notions of logical consequence are substantially different for theories and programs.

The following *propagation*[5] *rule*, denoted by $\mathsf{CP}(P)$, for given program P, was studied in [RZ01].

$$\frac{X_1 \quad \dots \quad X_n; \quad a_i \in X_i \quad (i = 1, \dots, n); \quad Y \leftarrow Z \in P; \quad \mathsf{mub}\{a_1, \dots, a_n\} \models Z}{Y \cup \bigcup_{i=1}^{n} (X_i \setminus \{a_i\})}$$

Applying this rule, we say that $Y \cup \bigcup_{i=1}^{n}(X_i \setminus \{a_i\})$ is a $\mathsf{CP}(P)$-consequence of a theory T if $X_1, \dots, X_n \in T$. The following operator is based on the notion of $\mathsf{CP}(P)$-consequence and acts on logically closed theories. Let T be a logically closed theory over D and let P be a program and define

$$\mathcal{T}_P(T) = \mathsf{cons}\left(\{Y \mid Y \text{ is a } \mathsf{CP}(P)\text{-consequence of } T\}\right).$$

In [RZ01], it was shown that $\mathcal{T}_P$ is a Scott-continuous function on the space of all logically closed theories, i.e. has a least fixed point $\mathsf{fix}(\mathcal{T}_P)$. It was also shown that $\mathsf{fix}(\mathcal{T}_P) = \mathsf{cons}(P)$, and that a simpler operator suffices: the *unary program-resolution operator* $\mathcal{U}_P$ on theories is defined as

$$\mathcal{U}_P(T) = \mathsf{cons}(\{Y \cup (X \setminus \{a\}) \mid X \in T, Y \leftarrow Z \in P, a \in \mathsf{min}(X), a \models Z\}),$$

where $\mathsf{min}(X)$ denotes the set of all minimal elements of X.

We obtain from [RZ01, Section 6], that $\mathcal{U}_P$ and $\mathcal{T}_P$ have the same least fixed points, namely $\mathsf{cons}(P)$.

The following technical result will make investigations concerning propagation easier.

4.2 Proposition Let P be a disjunctive logic program. Let Q be the set of all rules $Y \leftarrow \{d\}$ for which there is a rule $Y \leftarrow Z$ in P with $d \in Z$. Then $\mathcal{U}_P \equiv \mathcal{U}_Q$, and in particular $\mathsf{cons}(P) = \mathsf{cons}(Q)$.

Proof: The assertion follows immediately from the observation, that $a \models Z$ if and only if there exists $d \in Z$ with $a \models \{d\}$. ■

[5] This rule was called the *hyperresolution rule determined by* P in [RZ01]. We prefer the notion *propagation* since in our opinion resolution, when talking about programs, is better thought of as a process which yields the antecedent from a given consequent.

We shortly investigate the propagation rule for atomic domains with negation. If D is such a domain and $Y \leftarrow X$ is a rule over D, then let $\mathsf{clausal}(Y \leftarrow X)$ denote the set of clauses $\{Y \cup \{\overline{x}\} \mid x \in X\}$. If P is a program over D then let $\mathsf{clausal}(P)$ denote the union $\bigcup_{Y \leftarrow X \text{ in } P} \mathsf{clausal}(Y \leftarrow X)$. We call $\mathsf{clausal}(Y \leftarrow X)$, respectively, $\mathsf{clausal}(P)$ the *clausal form* of $Y \leftarrow X$, respectively, P.

4.3 Proposition Let D be an atomic domain with negation. Then $\mathsf{cons}(P) \subseteq \mathsf{cons}(\mathsf{clausal}(P))$.

Proof: Let W be a logical consequence of P and let $w \in D$ with $w \models \mathsf{clausal}(P)$. We have to show that $w \models W$. For this it suffices to show that w is a model of P. So let $Y \leftarrow X$ be a clause in P. Since $w \models \mathsf{clausal}(P)$ we have $w \models \mathsf{clausal}(Y \leftarrow X)$, i.e. $w \models Y \cup \{\overline{x}\}$ for all $x \in X$. Now assume $w \models X$. Then there is $x \in X$ with $x \sqsubseteq w$, so $\overline{x} \not\sqsubseteq w$. Since $w \models Y \cup \{\overline{x}\}$ there must be $y \in Y$ with $y \sqsubseteq w$, hence $w \models Y$. Since $Y \leftarrow X$ was chosen arbitrarily from P we have that w is a model of P as required. ■

We note that in the notation of Example 2.2 the program $P = \{r \leftarrow pq\}$ has model p, but p is not a model of $\mathsf{clausal}(P) = \{\{r, \overline{pq}\}\}$. So in general, a program and its clausal form do not share the same models. Furthermore, $\{r, \overline{pq}\}$ is a logical consequence of $\mathsf{clausal}(P)$, but not of P, so in general, a program and its clausal form do not share the same logical consequences. Another example for this is given by the program $P = \{\{p \leftarrow q\}, \{p \leftarrow \overline{q}\}\}$. In this case, we have $p \in \mathsf{cons}(\mathsf{clausal}(P))$, but $p \notin \mathsf{cons}(P)$. We see that even in the case of atomic domains with negation, propagation along $\leftarrow$ differs from implication.

4.1 Inference of Negative Information

Using the notation of Example 2.2, consider the program P consisting of the following rules. This program is in fact a propositional version of the well-known *even numbers* program, which can be found e.g. in [Fit94] or [HS0x].

$$\begin{aligned} \{p_0\} &\leftarrow \{\bot\} \\ \{p_{n+1}\} &\leftarrow \{\overline{p_n}\} \qquad \text{for all } n \in \mathbb{N}. \end{aligned}$$

Recall that $\mathsf{cons}(\emptyset) = \mathsf{cons}(\{\{\bot\}\})$, so we obtain $\mathsf{cons}(P) = \mathsf{fix}(\mathcal{T}_P) = \mathsf{cons}(\{\{p_0\}\})$. If we understand P as a logic program in the classical sense, however, then all major approaches to declarative semantics, e.g. the *Clark completion semantics* [Cla78] (also known as the *supported model semantics*), the *Fitting semantics* [Fit85] (also called *Kripke-Kleene semantics*), the *perfect model semantics* [Prz88], the *stable model semantics* [GL88] (also called *answer set semantics*, which is motivated by default logic), and the *well-founded semantics* [GRS91], agree on $M = \{\{p_n, \overline{p_{n+1}}\} \mid n \in \mathbb{N}\}$ as the intended model. We refer to [Sub99] for a very good and concise survey of these issues.

One way of justifying the latter model as the intended one would be the following: Since we obtain $\{p_0\}$ as a consequence (in some natural, naive sense) of the program, we are inclined to dismiss the possibility that $\{\overline{p_0}\}$ could hold, since it is inconsistent with the knowledge of $\{p_0\}$. So we infer "not $\{\overline{p_0}\}$", meaning that $\{\overline{p_0}\}$ can be dismissed as possible consequence. It follows that there is no way to justify $\{p_1\}$ as a consequence of the program. Common practice in nonmonotonic reasoning semantics is to therefore conclude "not $\{p_1\}$", and to identify this with $\{\overline{p_1}\}$, allowing to conclude $\{p_2\}$, and so on.

We attempt to lift this line of reasoning to the general setting of logic programs in algebraic domains. In place of the notion of negation, which is not available in the general setting, we can try to use inconsistency. From this perspective, and refering again to the above *even numbers* program, we can indeed dismiss $\{\overline{p_0}\}$ as a possible consequence from the observation that $\{p_0\}$ can be derived. Again we conclude that there is no way to justify $\{p_1\}$ as a consequence of the program, hence we obtain "not $\{p_1\}$", i.e. the absence of $\{p_1\}$ as a possible conclusion. In general algebraic domains, however, without a notion of negation, there may be many compact elements inconsistent with p_1. While in the case of the domain $\mathbb{T}^{\mathcal{V}}$ we can justify to derive $\overline{p_1}$ from the absence of provability of p_1, i.e. taking $\overline{p_1}$ as a kind of default, it is unclear, in the general case, which of the elements inconsistent with p_1 should be taken.

In the absence of an involutive notion of negation, we therefore should distinguish between two kinds of "negation", as follows. Assume that we believe in some items, i.e. compact elements of a domain, and that the collection of these items is consistent. We then say (1) that a compact element is *refuted by contradiction* if if is inconsistent with a compact element which belongs to our believe and (2) that a compact element is *refuted by default* if it is not believed, and not refuted by contradiction. Let us finally call a compact element *refuted*, if it is refuted by contradiction or refuted by default.

Let us again review the above *even numbers* program. We refuted $\overline{p_0}$ by contradiction, while we refuted p_1 by default, leading us to assuming $\overline{p_1}$, i.e. $\overline{p_1}$ was interpreted as the statement "p_1 is refuted". It relies entirely on the existence of an involutive negation, that we are able to identify "p_1 is refuted" with $\overline{p_1}$. For algebraic domains, we should be able to abstract from an involutive negation, and this is facilitated by the following definition.

4.4 Definition Let D be a coherent algebraic domain. An *extended clause* is a finite set $\{(c_1, N_1), \dots, (c_n, N_n)\}$ where for all $i \in \{1, \dots, n\}$ we have that N_i is a clause in D and $c_i \in \mathsf{K}(D)$. We call $(c, N) = (c, \{d_1, \dots, d_n\})$ an *extended precondition* and abreviate it by $(c; d_1, \dots, d_n)$, or by $c\neg d_1 \dots \neg d_n$. In the latter notation, we omit c if $c = \bot$ and $N \neq \emptyset$. If $N = \emptyset$ we abreviate (c, N) by c. Note that $(\bot, \emptyset)$ can be abbreviated to $\bot$, in which case $\bot$ may not be omitted. An extended clause $\{(c_1, N_1), \dots, (c_n, N_n)\}$ with $N_i = \emptyset$ for all i is called a *trivially extended clause*. A (*trivially*) *extended rule* is of the form $Y \leftarrow X$, where Y is a clause and X is a (trivially) extended clause. An (*extended disjunctive*) *program* consists of a set of extended rules.

We note that an extended disjunctive program which consists of trivially extended rules only, can be identified with a (non-extended) disjunctive logic program.

4.5 Example The following extended program P is a more suitable representation of the *even numbers* program above. We use again the notation from Example 2.2.

$$\begin{aligned} \{p_0\} &\leftarrow \{(\bot, \emptyset)\} \\ \{p_{n+1}\} &\leftarrow \{(\bot, \{p_n\})\} \qquad \text{for all } n \in \mathbb{N} \end{aligned}$$

In abbreviated form, this program may be written as

$$\begin{aligned}\{p_0\} &\leftarrow \{\bot\}\\ \{p_{n+1}\} &\leftarrow \{\neg p_n\} \qquad \text{for all } n\in\mathbb{N}.\end{aligned}$$

We now seek a reasonable notion of logical consequence of this extended program. Consider some candidate theory T which forms our belief. We next remove from the program all $\neg p_n$ for which p_n is not a logical consequence of T, i.e. we consider these p_n to be *refuted by default.* Then we remove all extended preconditions (c,N) for which there is $p\in N$ with $T\models\{p\}$. The remaining program is no longer extended, and we call it P/T. From P/T we can obtain its set of logical consequences, e.g. as $T'=\mathsf{fix}\left(\mathcal{T}_{P/T}\right)$. However, since T' is in general different from T, the set of elements which are refuted by default using T' is different from the set of elements refuted by default using T. But this means, that we are rather searching for a theory S with $S=\mathsf{fix}\left(\mathcal{T}_{P/S}\right)$, or in other words, if we define the operator $\mathcal{D}_P$ on theories (i.e. sets of clauses) by $\mathcal{D}_P(A)=\mathsf{fix}\left(\mathcal{T}_{P/A}\right)$, then we are searching for fixed points of the operator $\mathcal{D}_P$. It is in fact easy to see that the desired theory $\mathsf{cons}(\{\{p_{2n}\}\mid n\in\mathbb{N}\})$ is a fixed point of $\mathcal{D}_P$. It is indeed its unique fixed point, as we will see later, which is rather satisfactory.

The reader familiar with the stable model semantics for logic programming [GL88] may recognize the constructions made in Example 4.5: It is the original approach to stable models. This can be carried over to logic programs with disjunctions as consequents of their rules and containing two kinds of negation, namely classical negation and default negation. Such programs are called *extended disjunctive logic programs*, and we refer to [GL91] for the stable model semantics for these programs, which we will now lift to logic programming on coherent algebraic domains.

4.6 Definition Let D be a coherent algebraic domain, let P be an extended disjunctive program, and let T be a theory. We define P/T to be the (non-extended) program obtained by applying the following two transformations: (1) Delete from P all $\neg d$ for which d is not a logical consequence of T. (2) Delete all extended preconditions (c,N) for which there is $d\in N$ with $T\models\{d\}$. We define the *Gelfond-Lifschitz operator* or *default operator* $\mathcal{D}_P$ as a function on theories as $\mathcal{D}_P(T)=\mathsf{fix}\left(\mathcal{T}_{P/T}\right)$. A *stable model* of P is a fixed point of $\mathcal{D}_P$, i.e. a theory T such that $\mathcal{D}_P(T)=\mathsf{fix}\left(\mathcal{T}_{P/T}\right)=T$.

We obtain immediately from the definition that stable models are logically closed. Indeed, $\mathcal{D}_P$ maps logically closed theories to logically closed theories.

In order to justify our terminology, we have to explain what a *model* of an extended disjunctive program is.

4.7 Definition Consider a pair (T,S) of theories, which we call an *interpretation*, and let (c,N) be an extended precondition. We write $(T,S)\models(c,N)$ if $T\models\{c\}$ and for all $d\in N$ we have $S\not\models\{d\}$. If X is an extended clause, then we write $(T,S)\models X$ if $(T,S)\models C$ for some extended precondition C in X. The pair (T,S) is called a *model* of P if for every extended rule $Y\leftarrow X$ we have that $(T,S)\models X$ implies $T\models Y$. An interpretation (T,S) is called *consistent* if $\mathsf{cons}(T)\subseteq\mathsf{cons}(S)$. It is called *ideal* if $T=\mathsf{cons}(T)=\mathsf{cons}(S)$.

We can now identify every theory T with the ideal interpretation $(\mathsf{cons}(T), \mathsf{cons}(T))$. From this point of view, fixed points of the default operator are indeed models, as is easily verified.

The following technical result is analogous to Proposition 4.2.

4.8 Proposition Let P be an extended disjunctive program. Let Q be the set of all rules $Y \leftarrow \{(d,N)\}$ for which there is a rule $Y \leftarrow Z$ in P with $(d,N) \in Z$. Then P and Q have the same stable models.

Proof: Let T be a theory. By Proposition 4.2, the programs P/T and Q/T have the same set of logical consequences, which suffices. ■

Proposition 4.8 shows that it suffices to consider progams consisting of rules with single extended preconditions in the antecedent. For convenience, we will call such programs *singular*.

4.9 Proposition Let T and S be logically closed theories and $S \subseteq T$. Then $\mathcal{D}_P(T) \subseteq \mathcal{D}_P(S)$, i.e. $\mathcal{D}_P(T)$ is antitonic. In particular, $\mathcal{D}_P^2$ is monotonic with respect to set-inclusion on the set of all logically closed theories.

Proof: By Proposition 4.8, we can assume without loss of generality that P is singular. From Definition 4.6 we immediately obtain that $P/T \subseteq P/S$, and therefore that $\mathcal{U}_{P/T}(R) \subseteq \mathcal{U}_{P/S}(R)$ for all theories R. Consequently, $\mathsf{cons}(P/T) \subseteq \mathsf{cons}(P/S)$, i.e. $\mathcal{D}_P(T) \subseteq \mathcal{D}_P(S)$. ■

4.10 Remark The operator $\mathcal{D}_P^2$ is not in general Scott-continuous: Consider the program P consisting of the following rules, using the notation from Example 4.5.

$$
\begin{aligned}
\{p_0\} &\leftarrow \{\bot\} \\
\{p_{n+1}\} &\leftarrow \{\neg p_n\} \qquad \text{for all } n \in \mathbb{N} \\
\{q\} &\leftarrow \{\neg p_n \neg p_{n+1}\} \qquad \text{for all } n \in \mathbb{N} \\
\{r\} &\leftarrow \{\neg q\}
\end{aligned}
$$

We can now calculate

$$
\begin{aligned}
G_0 = \mathcal{D}_P(\mathsf{cons}(\{\bot\})) &= \mathsf{cons}(\{\{q\}, \{r\}, \{p_n\} \mid n \in \mathbb{N}\}) \\
L_1 = \mathcal{D}_P(G_0) &= \mathsf{cons}(\{\{p_0\}\}) \\
G_1 = \mathcal{D}_P(L_1) &= \mathsf{cons}(\{\{q\}, \{r\}, \{p_n\} \mid n \in \mathbb{N} \setminus \{1\}\}) \\
L_2 = \mathcal{D}_P(G_1) &= \mathsf{cons}(\{\{p_0\}, \{p_2\}\}) \\
&\vdots \\
G_n = \mathcal{D}_P(L_n) &= \mathsf{cons}(\{\{q\}, \{r\}, \{p_k\} \mid k \in \mathbb{N} \setminus \{1, \dots, 2n-1\}\}) \\
L_{n+1} = \mathcal{D}_P^2(L_n) = \mathcal{D}_P(G_n) &= \mathsf{cons}(\{\{p_0\}, \dots, \{p_{2n}\}\}),
\end{aligned}
$$

and we obtain

$$
\begin{aligned}
L_\omega = \bigcup_{n \in \mathbb{N}} L_n &= \mathsf{cons}(\{\{p_{2n}\} \mid n \in \mathbb{N}\}) \\
\mathcal{D}_P(L_\omega) &= \mathsf{cons}(\{\{r\}, \{p_{2n}\} \mid n \in \mathbb{N}\}) \\
\mathcal{D}_P^2(L_\omega) &= \mathsf{cons}(\{\{r\}, \{p_{2n}\} \mid n \in \mathbb{N}\}) \neq L_\omega,
\end{aligned}
$$

which shows that $\mathcal{D}_P^2$ is not Scott-continuous.

Although in general $\mathcal{D}_P^2$ fails to be Scott-continuous, we can make use of Proposition 4.9, which shows that it is monotonic, and that $\mathcal{D}_P$ is antitonic. So by the well-known Tarski fixed-point theorem, we obtain that $\mathcal{D}_P^2$ has a least fixed point, $L_P = \mathsf{lfp}\,(\mathcal{D}_P^2)$, and a greatest fixed point, $G_P = \mathsf{gfp}\,(\mathcal{D}_P^2)$.

4.11 Lemma $L_P = \mathcal{D}_P(G_P)$ and $G_P = \mathcal{D}_P(L_P)$.

Proof: We obtain $\mathcal{D}_P^2(\mathcal{D}_P(L_P)) = \mathcal{D}_P(\mathcal{D}_P^2(L_P)) = \mathcal{D}_P(L_P)$, i.e. $\mathcal{D}_P(L_P)$ is a fixed point of $\mathcal{D}_P^2$, hence $L_P \subseteq \mathcal{D}_P(L_P) \subseteq G_P$. Similarly, $L_P \subseteq \mathcal{D}_P(G_P) \subseteq G_P$. Since $L_P \subseteq G_P$ we get from Proposition 4.9 that $L_P \subseteq \mathcal{D}_P(G_P) \subseteq \mathcal{D}_P(L_P) \subseteq G_P$. Similarly, since $\mathcal{D}_P(L_P) \subseteq G_P$ we obtain $\mathcal{D}_P(G_P) \subseteq \mathcal{D}_P^2(L_P) = L_P \subseteq \mathcal{D}_P(G_P)$, so $\mathcal{D}_P(G_P) = L_P$, and $G_P = \mathcal{D}_P^2(G_P) = \mathcal{D}_P(L_P)$. ∎

4.12 Proposition (L_P, G_P) is a consistent model of P.

Proof: Consistency follows from $L_P \subseteq G_P$. It remains to show that $(L_P, G_P) = (L_P, \mathcal{D}_P(L_P))$ is a model of P. Assume without loss of generality that P is singular, and assume that $Y \leftarrow \{(c, N)\}$ is an extended rule in P with $(L_P, G_P) \models (c, N)$. Then $Y \leftarrow \{c\}$ is a rule in P/G_P, and $L_P \models \{c\}$, hence $\mathsf{fix}\left(\mathcal{T}_{P/G_P}\right) = \mathcal{D}_P(G_P) \models \{c\}$ and $L_P = \mathsf{fix}\left(\mathcal{T}_{P/G_P}\right) = \mathcal{D}_P(G_P) \models Y$ as required. ∎

We call (L_P, G_P) the *well-founded model* of P, borrowing terminology from nonmonotonic reasoning [Sub99].

4.13 Theorem For every stable model S we have $L_P \subseteq S \subseteq \mathcal{D}_P(L_P)$. Furthermore, if $L_P = \mathcal{D}_P(L_P)$ for some program P, i.e. if the well-founded model is ideal, then P has unique stable model L_P.

Proof: S is a fixed point of $\mathcal{D}_P$, hence a fixed point of $\mathcal{D}_P^2$ which suffices using Lemma 4.11. ∎

Considering again the program P from Remark 4.10, we notice that $\mathcal{D}_P^2(L_\omega) = \mathcal{D}_P(L_\omega)$ is the least fixed point of $\mathcal{D}_P^2$, and from Theorem 4.13 we obtain that $\mathcal{D}_P(L_\omega)$ is the unique stable model of P. Similar considerations hold for the *even numbers* program from Example 4.5.

4.2 Implicit and Explicit Knowledge

Extended disjunctive logic programming in algebraic domains enables us to represent knowledge in a variety of ways. Causal dependence may be encoded in the structure of the domain, i.e. implicitly, or explicitly by rules consituting a logic program. Likewise, negative information may be encoded implicitly in the domain, by facilitating inconsistency, or explicitly by using default negation. We give an example for this using a new representation of a classical problem.

4.14 Example We want to represent the following knowledge: (1) Tweety is a penguin. (2) Bob is a bird. (3) Birds fly or are penguins. (4) Birds always fly, unless the opposite can be shown. (5) Pengunins don't fly. (6) Penguins are birds.

We choose to represent (5) and (6) implicitly using the domain, and the remaining statements by a program. We first describe the domain D. Consider the set of items $A = \{p(T), p(B), b(T), b(B), f(T), f(B), n(T), n(B)\}$, where T stands for "Tweety", B stands for "Bob", $p(X)$ stands for "X is a penguin", $b(X)$ for "X is a bird", $f(X)$ for "X can fly", $n(X)$ for "X cannot fly". Now define D to be the set of all subsets c of A which satisfy the following conditions for all $X \in \{B, T\}$: (i) c does not contain both $f(X)$ and $n(X)$. (ii) c does not contain both $b(X)$ and $p(X)$. (iii) c does not contain both $p(X)$ and $n(X)$. (iv) c does not contain both $p(X)$ and $f(X)$.

For $c, d \in D$ let $c \leq d$ if and only if one of the following holds: (i) $c \subseteq d$, (ii) $c = (d \setminus p(X)) \cup b(X)$ for some $X \in \{B, T\}$, (iii) $c = (d \setminus p(X)) \cup n(X)$. We consider the domain $(D, \sqsubseteq)$, where $\sqsubseteq$ is the reflexive and transitive closure of $\leq$.

We note that in D, for all $X \in \{B, T\}$, sets containing both $n(X)$ and $f(X)$ are inconsistent, as are sets containing both $p(X)$ and $f(X)$. Now consider the following extended disjunctive program P

$$\begin{aligned}
\{p(T)\} &\leftarrow \{\bot\} \\
\{b(B)\} &\leftarrow \{\bot\} \\
\{f(X), p(X)\} &\leftarrow \{b(X)\} \qquad \text{for all } X \in \{B, T\} \\
\{f(X)\} &\leftarrow \{b(X) \neg n(X)\} \qquad \text{for all } X \in \{B, T\}
\end{aligned}$$

and the interpretation $S = \mathsf{cons}(\{p(T), b(T), n(T), b(B), f(B)\})$. The reader will easily verify that S is a stable model of P. In particular, we notice that in this model Tweety does not fly, but Bob does.

Let us now analyse how knowledge is represented in Example 4.14. The sentences (1) to (4) are certainly being represented by the clauses of the program P, while (5) and (6) are satisfactorily represented by the structure of the underlying domain. We can regard (5) and (6) as background knowledge, and thus obtain a conceptually clean way of distinguishing between background or domain knowledge, and the explicit knowledge given by the program rules. Likewise, negative knowledge is treated. "Flying" and "not flying" are opposite properties, and can not hold of a single object at the same time. This knowledge is encoded in the structure of the domain, by making them inconsistent. We had no need to endow the domain with an explicit negation function as in Section 3.2, which would provide an alternative, but in our opinion less concise way of treating Example 4.14. Default negation, however, was used explicitly in the program, and in the tradition of default logic was used to represent rules which "normally" hold, i.e. to which there may be exceptions.

5 Conclusions and Further Work

We introduced reasoning with negation to domain theory, in the form of logic programming in coherent algebraic domains. Many possible lines of investigation open up from our first observations, and we want to name just a few.

(1) Logic of domains. In the recent past, it became apparent that extended disjunctive logic programming, and its appropriate semantics, provides a very powerful tool for knowledge representation and reasoning, see eg. [Lif99, MT99]. It is therefore reasonable to expect that

the well-established research area concerned with the relationships and the interplay between logic and domain theory could profit from extensions along these lines. Generalized approaches to the well-founded and the stable semantics, as in [DMT00], could lead the way. For the approach presented here it would be fortunate if the restriction to algebraic domains could be disposed of, mainly because the interval domain fails to be algebraic.

(2) Domain-theory based logic programming. In classical logic programming, as implemented for example in Prolog, the use of negation is still unsatisfactory from a theoretical point of view, and it will probably remain so, since it can be argued that negation, as generally implemented in these systems, is not a clean declarative concept. Investigations on logic programming in algebraic domains may at some stage lead to a clean programming paradigm, including negation, which may be as powerful and applicable as modern Prolog systems. How this can be achieved is yet unclear, but first steps along these lines have already been performed, e.g. in [KRZ98]. Yet another line of research may be concerned with the machine learning paradigm known as *inductive logic programming* (ILP), see [MdR94], which still lacks a broad theoretical foundation. How this paradigm could be connected to domain theory proper is unclear, in particular since the *subsumption lattice*, which features prominently in ILP [NCdW97], fails to be a domain. As we have seen in Example 4.14 above, however, logic programming in algebraic domains provides a very natural way for a conceptually clean distinction between background knowledge and programs. For a domain-theory-based ILP paradigm one would attempt to encode the background knowledge in the domain and learn program rules.

(3) Theoretical foundations of answer set programming and deductive databases. Although there is a broad base of theoretical work on answer set programming ([Lif99, MT99]) and deductive databases ([Min97]), domain-theoretic foundations have, to our knowledge, not yet been studied for these paradigms — apart from some investigations concerning fixed-point semantics, e.g. [KKM93, HS99, DMT00, Hit01]. Extended disjunctive logic programming as presented in this report may provide an important link.

References

[Cla78] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[CZ00] T. Coquand and G.-Q. Zhang. Sequents, frames, and completeness. In *14th International Workshop on Computer Science Logic, Fischbachau, Germany, August 2000*, volume 1862 of *Lecture Notes in Computer Science*, pages 277–291. Springer, 2000.

[DMT00] M. Denecker, V.W. Marek, and M. Truszynski. Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In J. Minker, editor, *Logic-based Artificial Intelligence*, chapter 6, pages 127–144. Kluwer Academic Publishers, Boston, 2000.

[Fit85] M. Fitting. A Kripke-Kleene-semantics for general logic programs. *Journal of Logic Programming*, 2:295–312, 1985.

[Fit94] M. Fitting. Metric methods: Three examples and a theorem. *Journal of Logic Programming*, 21(3):113–127, 1994.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R.A. Kowalski and K.A. Bowen, editors, *Logic Programming. Proceedings of the 5th International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[GRS91] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[Hit01] P. Hitzler. *Generalized Metrics and Topology in Logic Programming Semantics*. PhD thesis, Department of Mathematics, National University of Ireland, University College Cork, 2001.

[HS99] P. Hitzler and A.K. Seda. Some issues concerning fixed points in computational logic: Quasi-metrics, multivalued mappings and the Knaster-Tarski theorem. In *Proceedings of the 14th Summer Conference on Topology and its Applications: Special Session on Topology in Computer Science, New York*, volume 24 of *Topology Proceedings*, pages 223–250, 1999.

[HS0x] P. Hitzler and A.K. Seda. Generalized metrics and uniquely determined logic programs. *Theoretical Computer Science*, 200x. To appear.

[Joh82] P. T. Johnstone. *Stone Spaces*. Number 3 in Cambridge studies in advanced mathematics. Cambridge University Press, 1982.

[KKM93] M.A. Khamsi, V. Kreinovich, and D. Misane. A new method of proving the existence of answer sets for disjunctive logic programs: A metric fixed-point theorem for multivalued mappings. In C. Baral and M. Gelfond, editors, *Proceedings of the Workshop on Logic Programming with Incomplete Information, Vancouver, B.C., Canada*, pages 58–73, 1993.

[KRZ98] E. Klavins, W. Rounds, and G.-Q. Zhang. Experimenting with power default reasoning. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, 1998.

[Lif99] V. Lifschitz. Answer set planning. In D. De Schreye, editor, *Logic Programming. Proceedings of the 1999 International Conference on Logic Programming*, pages 23–37, Cambridge, Massachusetts, 1999. MIT Press.

[MdR94] S. Muggleton and L. de Raedt. Inductive logic programming: Theory and applications. *Journal of Logic Programming*, 19–20:629–679, 1994.

[Min97] J. Minker. Logic and databases: Past, present, and future. *AI Magazine*, 18(3):21–47, 1997.

[MT99] V.M. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In K.R. Apt, V.W. Marek, M. Truszczyński, and D.S. Warren, editors, *The Logic Programming Paradigm: A 25 Year Persepective*, pages 375–398. Springer, Berlin, 1999.

[NCdW97] S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence.* Springer, Berlin, 1997.

[Plo78] G. Plotkin. $\mathbb{T}^\omega$ as a universal domain. *Journal of Computer and System Sciences*, 17:209–236, 1978.

[Prz88] T.C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, Los Altos, CA, 1988.

[RZ01] W.C. Rounds and G.-Q. Zhang. Clausal logic and logic programming in algebraic domains. *Information and Computation*, 171(2):156–182, 2001.

[Sub99] V.S. Subrahmanian. Nonmonotonic logic programming. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):143–152, January/February 1999.