Generalized Horn Clause Programs *

Saumya K. Debray
Department of Computer Science
The University of Arizona
Tucson, AZ 85721

Raghu Ramakrishnan
Computer Sciences Department
University of Wisconsin
Madison, WI 53706

November 23, 1994

Abstract

This paper considers, in a general setting, an axiomatic basis for Horn clause logic programming. It characterizes a variety of "Horn-clause-like" computations, arising in contexts such as deductive databases, various abstract interpretations, and extensions to logic programming involving E-unification, quantitative deduction, and inheritance, in terms of two simple operators, and discusses algebraic properties these operators must satisfy. It develops fixpoint and model-theoretic semantics in this generalized setting, and shows that the fixpoint semantics is well-defined and coincides with the model-theoretic semantics. This leads to a generalized notion of a Horn clause logic program that captures a variety of fixpoint computations proposed in different guises, and allows concise expression in the logic programming idiom of several programs that involve aggregate operations.

*S. Debray was supported in part by NSF grant CCR-9123520. R. Ramakrishnan's work was supported in part by NSF grant IRI-9011563, a Presidential Young Investigator award, and a David and Lucile Packard Foundation Fellowship in Science and Engineering.

1 Introduction

A logic program consists of a set of clauses of the form 'H :- $B_1, \ldots, B_n$.' Logically, such a clause is read as an implication "B_1 *and* $\ldots$ *and* B_n *implies* H." This reading gives rise to two distinct notions of the semantics of a Horn program: the *model-theoretic* semantics, which relates to the set of facts that follow logically from the program; and the *fixpoint* semantics, which relates to the set of facts that can be computed using the rules defined by the program. In the case of "ordinary" Horn programs (essentially an idealized version of Prolog), the model-theoretic semantics of a program is given by its least Herbrand model, which is the intersection of all its Herbrand models (i.e., models where function symbols are uninterpreted). Conventionally, the fixpoint semantics of a Horn program P is obtained by repeatedly "applying" the clauses comprising P to a set of atoms that is initially empty, and collecting together the results, until nothing new can be generated: this process can be formalized in terms of the least fixpoint of an operator T_P defined in terms of P. A fundamental property of Horn programs is that every program P has a (unique) least Herbrand model that coincides with the least fixpoint of the associated operator T_P.

The fact that the model-theoretic and fixpoint semantics of a Horn program coincide is very pleasant mathematically, and establishes the fundamental connection between the operational (fixpoint) and intended (model-theoretic) semantics for a large class of systems. Researchers who have investigated variations of and extensions to Horn clause programming have generally striven to prove similar results for the various different cases considered (for examples of such variations and extensions, see Section 6). Unfortunately, the development of the various model-theoretic and fixpoint semantics in such cases has typically proceeded from scratch. This is unsatisfactory for two reasons: first, much of the machinery involved in these proofs is reinvented, with minor modifications, for each such variant; and second, by approaching these results piecemeal on a per-application basis, we miss the fundamental essence of "Horn-like" computations that causes these results to hold. The aim of this paper is to address this situation by considering, in a fairly general setting, an axiomatic basis for Horn clause logic programming. The basic idea behind our approach is very simple: we define a generic notion of Horn computations in a simple algebraic setting, and present a small set of axioms that seek to capture the essence of bottom-up fixpoint evaluation for a variety of "Horn-like" computations. These axioms serve to specify a wide class of algebraic structures over which generalized Horn programs can be defined in a manner that conforms with our intuitions of Horn computations, and guarantees the existence of intuitively reasonable least fixpoints and least models that coincide. In the process, they show how a variety of fixpoint computations, proposed in different guises, can be expressed concisely in the logic programming idiom. In Section 6, we demonstrate the generality of these results by considering several proposed "Horn-like" frameworks. Finally, we discuss connections to OLDT resolution and memoizing top-down computations in Section 8.

2 Preliminaries

We consider the language of Horn clauses [21]; however, because we want to reason about fixpoint computations over both standard and non-standard domains, we refer to *values* rather than terms: values are simply syntactic entities manipulated during a computation, and can be atoms, terms, feature structures, graph structures, abstract domain elements, etc. The set of all values is denoted by D. Further, it is technically convenient to treat a predicate of arity k as a unary predicate whose argument is a k-tuple of terms.

The fixpoint evaluation of a program P on an input set of values R is the set of values that can be obtained by repeatedly applying the clauses of P to inferred values, with the initial set of inferred values being R, until no new tuples can be inferred.[1] There are two reasons we consider arbitrary input sets instead of evaluating P on $\emptyset$: first, in deductive database applications the "base relations" are usually considered to be part of the input rather than of the program itself; second, we may want to fix the meanings of some predicates, such as *plus* and *minus* that specify arithmetic operations.

We assume that we have a binary operator $\otimes : \mathsf{D} \times \mathsf{D} \longrightarrow \mathsf{D}$. The notion of a value being an instance of another is defined as follows.

Definition 2.1 Given $t_1, t_2 \in \mathsf{D}$, t_1 is an *instance* of t_2 if and only if there is some $t_3 \in \mathsf{D}$ such that $t_1 = t_2 \otimes t_3$. We denote this as $t_1 \preceq t_2$. ∎

Since bottom-up evaluation requires the manipulation of sets of values, we "lift" the operator $\otimes$ to operate on sets, to obtain an operator $\widehat{\otimes} : \mathcal{P}(\mathsf{D}) \times \mathcal{P}(\mathsf{D}) \longrightarrow \mathcal{P}(\mathsf{D})$:

Definition 2.2 For any S_1, S_2 in $\mathcal{P}(\mathsf{D})$, $S_1 \mathbin{\widehat{\otimes}} S_2 = \{s_1 \otimes s_2 \mid s_1 \in S_1, s_2 \in S_2\}$. ∎

Sets of values may reasonably be assumed to convey "information"—indeed, the T_P operator of van Emden and Kowalski [43], like other immediate consequence operators that have been proposed for various extensions to Horn logic programming, essentially specifies what information can be inferred using the rules of a given program given some information that is known to be true. In general, it may happen that two different sets of values S_1 and S_2 contain the same amount of information: this can be expressed by specifying, for each set of values S, a "canonical representative" $S^\sharp$ that contains the same amount of information as S: the idea is that all sets that contain the same amount of information have the same canonical representative. To this end, we assume that we are given a *normalization operator* $^\sharp : \mathcal{P}(\mathsf{D}) \longrightarrow \mathcal{P}(\mathsf{D})$ that yields, for any set of values S, the canonical representative of those sets of values whose information content is the same as that of S. The notion of a set containing "the same amount of information" as another can be captured via a relation $\approx$:

Definition 2.3 Given $S_1, S_2 \in \mathcal{P}(\mathsf{D})$, $S_1 \approx S_2$ if and only if $S_1^\sharp = S_2^\sharp$. ∎

Recall that the fixpoint evaluation of a Horn program can be seen in terms of repeatedly applying the rules of the program and "collecting together" the sets of tuples obtained from each rule. The

[1] Of course, we have no way of knowing at this time whether such a set always exists, or, if it exists, is unique, so perhaps it is premature to speak of "the set" of values so obtained. At this point, the appeal is primarily to the reader's intuitions: later in the paper we will consider conditions that guarantee that there is a unique "smallest" set that can be obtained by repeated application of rules in this way.

notion of "collecting together" a set of sets of tuples can now be defined in terms of a *merge* operator $\widehat{\oplus}$:

Definition 2.4 Given any set of sets of values $\mathbf{S} \subseteq \mathcal{P}(\mathsf{D})$, their merge is defined as $\widehat{\oplus}\ \mathbf{S} = (\cup \mathbf{S})^\sharp$. ∎

Finally, given a set S of (values that are) tuples, the selection operator $\downarrow_1$ returns the set consisting of the first element of each (tuple) element in S.

A Summary of the Development so far : *We have assumed that we are given two operators $\otimes$ and $^\sharp$. These, together with the operations of set union and selection ($\downarrow_1$), yield an algebra with operations $\cup, \downarrow_1, {}^\sharp$, and $\otimes$. Using these, we have defined the derived operators $\widehat{\otimes}$ and $\widehat{\oplus}$, and the relation $\approx$, which are notationally more convenient in the development that follows. The reader may wish to keep in mind, however, that everything in the rest of the paper can easily be restated in terms of only $\cup, \downarrow_1, {}^\sharp$, and $\otimes$, essentially by a process of macro expansion.*

3 Axioms for Horn Programs

Our first axiom states that normalization preserves information content (and thus that $^\sharp$ is idempotent):

Axiom 1 *For any $S \in \mathcal{P}(\mathsf{D})$, $S^\sharp \approx S$.*

It is straightforward, from the definition of $\approx$, that it is an equivalence relation.

NOTATION : *In the remainder of the paper, the "domain of computation" we will typically be concerned with will be $\mathcal{P}(\mathsf{D})/\approx$. For brevity of notation, this set will be denoted by $\mathcal{D}$.*

The next axiom states, intuitively, that the operations of interest in our algebra, namely, union, $\widehat{\otimes}$, and the selection of the first element of a tuple, respect the notion of information content:

Axiom 2 *$\approx$ is a congruence relation with respect to $\cup$, $\widehat{\otimes}$, and $\downarrow_1$, i.e.:*

1. *Let $\{S_i \mid i \geq 0\}$ and $\{S'_i \mid i \geq 0\}$ be subsets of $\mathcal{D}$, then if $S_i \approx S'_i$ for each $i \geq 0$, then $(\cup_{i\geq 0} S_i) \approx (\cup_{i \geq 0} S'_i)$.*
2. *For any $\{S_1, S'_1, S_2, S'_2\} \subseteq \mathcal{D}$, if $S_1 \approx S'_1$ and $S_2 \approx S'_2$ then $S_1 \ \widehat{\otimes}\ S_2 \approx S'_1 \ \widehat{\otimes}\ S'_2$.*
3. *If $S \approx S'$ for $S, S' \in \mathcal{D}$, then $S \downarrow_1 \approx\ S' \downarrow_1$.*

Our third axiom states that $\otimes$ behaves sensibly when applied to tuples (so that, for example, $\otimes$ applied to two 5-tuples does not yield a 17-tuple):

Axiom 3 *Let $r_1, \ldots, r_n$ and $s_1, \ldots, s_n$ be elements of D, then $\langle r_1, \ldots, r_n \rangle \otimes \langle s_1, \ldots, s_n \rangle = \langle t_1, \ldots, t_n \rangle$, where $t_i \preceq r_i$ and $t_i \preceq s_i, 1 \leq i \leq n$.*

Note that this axiom states only that $\otimes$ cannot behave in completely unconstrained ways when applied to tuples. It may not say everything we might want to say about the behavior of $\otimes$ on tuples in the context of a particular application (e.g., that variables shared across different elements of the tuples are handled consistently): such additional constraints have to be taken into account explicitly when defining $\otimes$ for a particular application. These axioms are sufficient to show our first main result, which states that every program has a least fixpoint. Our second result states that every program has a least model. The notion of a model rests upon a reinterpretation of the notion of entailment based on the notion of "information content" discussed above. To establish this result we need an additional axiom that expresses the requisite properties for $\otimes$. This axiom consists of two parts: intuitively, the first part states that for any values t and t', the value $t \otimes t'$ conveys no more information than does t, while the second part elaborates on the special case where $t' = t$, stating the value $t \otimes t$ conveys no less information than does t.

Axiom 4 *For any $t, t' \in \mathsf{D}$, (1) $\{t\} \mathbin{\widehat{\oplus}} \{t \otimes t'\} \approx \{t\}$; and (2) $\{t\} \mathbin{\widehat{\oplus}} \{t \otimes t\} \approx \{t \otimes t\}$.*

Our first use of Axiom 4 is in Section 5; preceding results require only Axioms 1–3.

4 A Generic Fixpoint Semantics

A generic fixpoint semantics for Horn programs, in terms of the operators $\otimes$ and $\widehat{\oplus}$, can be defined as follows:

1. *Rule Application* :
   To evaluate a clause C given a set $\mathbf{R}$ of values for its body literals, it is necessary to generate instances C' of C such that each body literal of C' is an instance of an appropriate value in $\mathbf{R}$. If $C \equiv$ 'H :– $B_1, \ldots, B_n$', then any such instance is given by $\langle H, B_1, \ldots, B_n \rangle \otimes \langle H, B'_1, \ldots, B'_n \rangle$, where $B'_i \in \mathbf{R}, 1 \leq i \leq n$. The set of all such instances, therefore, is simply
   $$\{\langle H, B_1, \ldots, B_n \rangle\} \mathbin{\widehat{\otimes}} \{\langle H, B'_1, \ldots, B'_n \rangle \mid B'_i \in \mathbf{R}, 1 \leq i \leq n\}.$$
   The appropriate instances of the head of the clause C can then be obtained simply by selecting the first element of each tuple so computed:
   $$\begin{aligned} \mathit{apply\_rule}(C, \mathbf{R}) = (\{\langle H, B_1, \ldots, B_n \rangle\} \mathbin{\widehat{\otimes}} \{\langle H, B'_1, \ldots, B'_n \rangle \mid B'_i \in \mathbf{R}, 1 \leq i \leq n\}) \downarrow_1 \\ \textbf{where } C \equiv \text{'}H \text{ :– } B_1, \ldots, B_n\text{'}. \end{aligned}$$

2. *Evaluating a Program* :
   Evaluating a program on input $\mathbf{R}$ now involves computing the least fixpoint of an operator $\mathcal{T}_P$ that yields the values that can be inferred by using $\mathbf{R}$ in the body literals of rules of P.
   $$\mathcal{T}_P(\mathbf{R}) = \widehat{\oplus} \{\mathit{apply\_rule}(C, \mathbf{R}) \mid C \in P\}.$$

3. *Fixpoint Semantics* :
   The fixpoint semantics of a program P is given by the least fixpoint $lfp(\mathcal{T}_P)$. Corollary 4.9 below shows that this exists and is well-defined for any program P.

The following example illustrates our approach, and also shows that it can capture some important cases of Horn computations.

Example 4.1 Consider the case where, given a Horn program P, the set of values D is the set of terms of the language of P augmented with a distinguished element $\bot$, denoting failure of unification, and the instance operator $\otimes$ and the normalization operator $^\sharp$ are defined as follows:

1. $\otimes$ is the "usual" first order notion of "most general instance" [31, 33]: Given two (tuples of) terms t_1 and t_2, $t_1 \otimes t_2$ is their most general instance, if one exists, and $\bot$ otherwise.
2. $^\sharp$ is the identity function, modulo ignoring $\bot$: for any $S \in \mathcal{D}$, $S^\sharp = S \setminus \{\bot\}$. Thus, $\widehat{\oplus}$ is set union, modulo ignoring $\bot$.

This case corresponds to the "usual" case of Horn program evaluation, and yields the S-model semantics of Falaschi et al. [13].

Alternatively, we can consider the following variation on this example: the operator $\otimes$ is defined as before, but $^\sharp$ is modified so that it yields only *irredundant* sets of elements, where an element of a set S is redundant if and only if it is subsumed by some other element of S [22]. In other words, $S^\sharp$ is the set of maximal elements of S, where the partial order is the usual "more general than" ordering on terms (modulo variable renaming). In this case, the computation involves subsumption checking each time the rules are applied. ∎

The following properties of $\widehat{\oplus}$ and $\widehat{\otimes}$ that are useful in developing our results.

Lemma 4.1 $\widehat{\oplus}$ *is associative, commutative, and idempotent.* ∎

Proposition 4.2 $\widehat{\otimes}$ *distributes over* $\widehat{\oplus}$ *.*

Proof: By definition, $S \widehat{\otimes} S' = \{s \otimes s' \mid s \in S \wedge s' \in S'\}$ for any $S, S' \in \mathcal{P}(\mathsf{D})$. Thus, we have

$$\begin{aligned}
&S \widehat{\otimes} (S_1 \cup S_2) \\
&= \{s \otimes s' \mid s \in S \wedge s' \in S_1 \cup S_2\} \\
&= \{s \otimes s' \mid (s \in S \wedge s' \in S_1) \vee (s \in S \wedge s' \in S_2)\} \\
&= \{s \otimes s' \mid s \in S \wedge s' \in S_1\} \cup \{s \otimes s' \mid s \in S \wedge s' \in S_2\} \\
&= (S \widehat{\otimes} S_1) \cup (S \widehat{\otimes} S_2).
\end{aligned}$$

Now from Axiom 1, $S \approx S^\sharp$, and since, from Axiom 2, $\approx$ is a congruence with respect to $\cup$, we have

$$S \mathbin{\widehat{\otimes}} (S_1 \cup S_2) \approx S \mathbin{\widehat{\otimes}} (S_1 \cup S_2)^\sharp = S \mathbin{\widehat{\otimes}} (S_1 \mathbin{\widehat{\oplus}} S_2).$$

Similarly, $(S \mathbin{\widehat{\otimes}} S_1) \cup (S \mathbin{\widehat{\otimes}} S_2) \approx ((S \mathbin{\widehat{\otimes}} S_1) \cup (S \mathbin{\widehat{\otimes}} S_2))^\sharp = ((S \mathbin{\widehat{\otimes}} S_1) \mathbin{\widehat{\oplus}} (S \mathbin{\widehat{\otimes}} S_2))$. Thus, we have

$$S \mathbin{\widehat{\otimes}} (S_1 \mathbin{\widehat{\oplus}} S_2) \approx (S \mathbin{\widehat{\otimes}} S_1) \mathbin{\widehat{\oplus}} (S \mathbin{\widehat{\otimes}} S_2),$$

which shows that $\widehat{\otimes}$ is left-distributive over $\widehat{\oplus}$. A symmetric argument can be used to show that $\widehat{\otimes}$ is also right-distributive over $\widehat{\oplus}$. Thus, $\widehat{\otimes}$ distributes over $\widehat{\oplus}$. ∎

Intuitively, given sets of values $S_1, S_2 \in \mathcal{D}$, $S_1 \mathbin{\widehat{\oplus}} S_2$ contains the information present in both S_1 and S_2 This implies that $S_1 \mathbin{\widehat{\oplus}} S_2$ can reasonably be expected to contain more information that either S_1 or S_2 by itself. The notion of "contains more information than" can be formalized by defining a binary relation $\sqsubseteq$ over $\mathcal{D}$ as follows:

Definition 4.1 For any $S_1, S_2 \in \mathcal{D}$, $S_1 \sqsubseteq S_2$ if and only if $S_1 \mathbin{\widehat{\oplus}} S_2 \approx S_2$. ∎

Proposition 4.3 *$\mathcal{D}$ is partially ordered by $\sqsubseteq$, and forms a complete lattice with least element $\emptyset$ and greatest element D, and join operation $\widehat{\oplus}$.*

Proof: That $\langle \mathcal{D}, \sqsubseteq \rangle$ is a poset is a straightforward consequence of Lemma 4.1.

It is straightforward to show, from the definition of $\sqsubseteq$, that for any S_1 and S_2 in $\langle \mathcal{D}, \sqsubseteq \rangle$, $S_1 \mathbin{\widehat{\oplus}} S_2$ is the least upper bound of S_1 and S_2. First, from Lemma 4.1, $\widehat{\oplus}$ is associative and idempotent, whence it is easy to show that for any $S_1, S_2 \in \mathcal{D}$,

$$S_1 \mathbin{\widehat{\oplus}} (S_1 \mathbin{\widehat{\oplus}} S_2) \approx S_1 \mathbin{\widehat{\oplus}} S_2$$

whence $S_1 \sqsubseteq (S_1 \mathbin{\widehat{\oplus}} S_2)$. A similar argument establishes that $S_2 \sqsubseteq (S_1 \mathbin{\widehat{\oplus}} S_2)$. Thus, $S_1 \mathbin{\widehat{\oplus}} S_2$ is an upper bound for S_1 and S_2 with respect to $\sqsubseteq$. We now show that it is the least upper bound: consider any $S_3 \in \mathcal{D}$ that is an upper bound for S_1 and S_2, i.e. $S_1 \sqsubseteq S_3$ and $S_2 \sqsubseteq S_3$. From the definition of $\sqsubseteq$, it follows that $S_1 \mathbin{\widehat{\oplus}} S_3 \approx S_3$ and $S_2 \mathbin{\widehat{\oplus}} S_3 \approx S_3$. Then, we have

$$\begin{aligned}
(S_1 \mathbin{\widehat{\oplus}} S_2) \mathbin{\widehat{\oplus}} S_3 &\approx (S_1 \mathbin{\widehat{\oplus}} S_3) \mathbin{\widehat{\oplus}} (S_2 \mathbin{\widehat{\oplus}} S_3) && \text{from Lemma 4.1} \\
&\approx S_3 \mathbin{\widehat{\oplus}} S_3 && \text{since } (S_1 \mathbin{\widehat{\oplus}} S_3) \approx S_3 \approx (S_2 \mathbin{\widehat{\oplus}} S_3) \\
&\approx S_3 && \text{since, from Lemma 4.1, } \widehat{\oplus} \text{ is idempotent}
\end{aligned}$$

whence $(S_1 \mathbin{\widehat{\oplus}} S_2) \sqsubseteq S_3$, i.e. $S_1 \mathbin{\widehat{\oplus}} S_2$ is the least upper bound of S_1 and S_2 with respect to $\sqsubseteq$.

Now $\emptyset \subseteq S$ for any $S \in \mathcal{D}$, whence we have $\emptyset \mathbin{\widehat{\oplus}} S = (\emptyset \cup S)^\sharp = S^\sharp$ so that $\emptyset \sqsubseteq S$, i.e. $\emptyset$ is the least element of $\mathcal{D}$with respect to $\sqsubseteq$.

Since $\cup \mathbf{S} \in \mathcal{D}$ for any $\mathbf{S} \subseteq \mathcal{D}$, it follows, from the definition of $\widehat{\oplus}$, that $\widehat{\oplus}\, \mathbf{S} \in \mathcal{D}$ for any $\mathbf{S} \subseteq \mathcal{D}$. Thus, every subset of $\mathcal{D}$ has a least upper bound in $\mathcal{D}$. Since $\langle \mathcal{D}, \sqsubseteq \rangle$ also has a least element, it follows [6] that it is a complete lattice.

To see that D is the greatest element of the lattice $\langle \mathcal{D}, \sqsubseteq \rangle$, note that $S \subseteq \mathsf{D}$ for any $S \in \mathcal{D}$, so $(S \cup \mathsf{D}) = \mathsf{D}$. It follows that $(S \cup \mathsf{D})^\sharp = \mathsf{D}^\sharp$, which means that $(S \mathbin{\widehat{\oplus}} \mathsf{D}) \approx \mathsf{D}$, or equivalently, $S \sqsubseteq \mathsf{D}$. ■

Lemma 4.4 *Let C be any clause in a program. For any $S_1, S_2 \in \mathcal{D}$, if $S_1 \approx S_2$ then $apply\_rule(C, S_1) \approx apply\_rule(C, S_2)$.*

Proof: Let C be a clause '$p(\bar{t})$:− $q_1(\bar{t}_1), \ldots, q_n(\bar{t}_n)$', and let $S_1, S_2 \in \mathcal{D}$ such that $S_1 \approx S_2$. For simplicity of notation, let R_1 denote the set $\{\langle \bar{t}, \bar{t}'_1, \ldots, \bar{t}'_n \rangle \mid \bar{t}'_i \in (S_1)_{q_i}\}$, and R_2 the set $\{\langle \bar{t}, \bar{t}'_1, \ldots, \bar{t}'_n \rangle \mid \bar{t}'_i \in (S_2)_{q_i}\}$. Since $S_1 \approx S_2$, it follows, from the definition of $^\sharp$, that $R_1 \approx R_2$. From Axioms 2 and 3, we therefore have

$$(\{\langle \bar{t}, \bar{t}_1, \ldots, \bar{t}_n \rangle\} \mathbin{\widehat{\otimes}} R_1) \approx (\{\langle \bar{t}, \bar{t}_1, \ldots, \bar{t}_n \rangle\} \mathbin{\widehat{\otimes}} R_2).$$

It follows, from Axiom 2, that $(\{\langle \bar{t}, \bar{t}_1, \ldots, \bar{t}_n \rangle\} \mathbin{\widehat{\otimes}} R_1) \downarrow_1 \approx (\{\langle \bar{t}, \bar{t}_1, \ldots, \bar{t}_n \rangle\} \mathbin{\widehat{\otimes}} R_2) \downarrow_1$. From the definition of *apply_rule*, we then have $apply\_rule(C, S_1) \approx apply\_rule(C, S_2)$. ■

Lemma 4.5 *Let C be any clause in a program, and let $\mathbf{S} = \{S_i \mid i \geq 0\}$ be a chain in $\langle \mathcal{D}, \sqsubseteq \rangle$, i.e. $S_0 \sqsubseteq S_1 \sqsubseteq S_2 \sqsubseteq \ldots$. Then,*

$$\bigcup_{S \in \mathbf{S}} apply\_rule(C, S) \approx apply\_rule(C, \cup \mathbf{S}).$$

Proof: Let C be the clause 'H :− $B_1, \ldots, B_n$'. First, consider the set $\mathbf{R} = \{R_i \mid i \geq 0\}$, defined as follows:

$$R_0 = S_0;$$
$$R_i = R_{i-1} \cup S_i, \quad i > 0.$$

It is obvious that $\langle \mathbf{R}, \subseteq \rangle$ is a chain, i.e., $R_0 \subseteq R_1 \subseteq R_2 \subseteq \ldots$. We show, by induction on i, that $R_i \approx S_i$. The base case, for $i = 0$, is trivial. In the inductive case, assume that $R_i \approx S_i$ for $0 \leq i \leq k$, and consider $R_{k+1} = R_k \cup S_{k+1}$. Since $\mathbf{S}$ is a chain, $S_k \sqsubseteq S_{k+1}$, so $S_k \sqcup S_{k+1} = S_{k+1}$. Since $S_1 \sqcup S_2$ is nothing but $(S_1 \cup S_2)^\sharp$ for any S_1 and S_2, this implies that $(S_k \cup S_{k+1})^\sharp = S_{k+1}$, and therefore that $(S_k \cup S_{k+1})^{\sharp\sharp} = S_{k+1}^\sharp$. Since $^\sharp$ is idempotent (Axiom 1), this implies that $S_k \cup S_{k+1} \approx S_{k+1}$. Since $R_k \approx S_k$ from the inductive hypothesis, Axiom 2 implies that $R_k \cup S_{k+1} \approx S_{k+1}$, i.e., that $R_{k+1} \approx S_{k+1}$.

The proof proceeds by first considering the application of *apply_rule* to the chain $\langle \mathbf{R}, \subseteq \rangle$, then applying the results to the chain $\langle \mathbf{S}, \sqsubseteq \rangle$. First, consider an atom $H' \in \cup_i apply\_rule(C, R_i)$. It must be the case that $H' \in apply\_rule(C, R_i)$ for some $R_i \in \mathbf{R}$, which means that there must be atoms $B'_j \in R_i, 1 \leq j \leq n$, such that

$$H' = (\langle H, B_1, \ldots, B_n\rangle \otimes \langle H, B'_1, \ldots, B'_n\rangle) \downarrow_1.$$

But if this is true, then since $B'_j \in R_i$, it must also be the case that $B'_j \in \cup_i R_i, 1 \leq j \leq n$. This means that H' must also be in $apply\_rule(C, \cup_i R_i)$. Thus, $\cup_i apply\_rule(C, R_i) \subseteq apply\_rule(C, \cup_i R_i)$.

We next want to show that $apply\_rule(C, \cup_i R_i) \subseteq \cup_i apply\_rule(C, R_i)$. Consider an atom $H' \in apply\_rule(C, \cup_i R_i)$: suppose H' is in $apply\_rule(C, \cup_i R_i)$ because there is some set of atoms $B'_j \in \cup_i R_i, 1 \leq j \leq n$, such that

$$H' = (\langle H, B_1, \ldots, B_n\rangle \otimes \langle H, B'_1, \ldots, B'_n\rangle) \downarrow_1.$$

Clearly, for each $B'_j, 1 \leq j \leq n$, there must be some element of $\mathbf{R}$—call it $R^{(j)}$—such that $B'_j \in R^{(j)}$: for otherwise, B'_j cannot be in $\cup\mathbf{R}$. Thus, if we union together all such $R^{(j)}$, then all of the B'_j will appear in the result: in other words, let $R^{(\star)} = \cup_{j=1}^{n} R^{(j)}$, then $B'_j \in R^{(\star)}, 1 \leq j \leq n$. This implies that $H' \in apply\_rule(C, \cup_{j=1}^{n} R^{(j)})$. Now each of the sets $R^{(j)}$ is an element of the chain $\mathbf{R} = \{R_0, R_1, R_2, \ldots\}$: let

$$m = \max\{i \mid \exists j : R_i = R^{(j)}\}$$

then it is not difficult to see, from the definition of the set $\mathbf{R}$, that $R_m = \cup_{j=1}^{n} R^{(j)}$. Thus, we have

$$H' \in apply\_rule(C, R_m).$$

It follows from this that $H' \in \cup_i apply\_rule(C, R_i)$, i.e., that $apply\_rule(C, \cup_i R_i) \subseteq \cup_i apply\_rule(C, R_i)$. Thus, we have $apply\_rule(C, \cup_i R_i) = \cup_i apply\_rule(C, R_i)$.

Now $R_i \approx S_i$ for $i \geq 0$, and from Axiom 2 $\approx$ is a congruence with respect to $\cup$, whence it is easy to see that $\cup_i R_i \approx \cup_i S_i$. Then, from Lemma 4.4, we have $apply\_rule(C, \cup_i S_i) \approx \cup_i apply\_rule(C, S_i)$. The Lemma follows. ∎

Lemma 4.6 *Let* $\mathbf{C}$ *be any set of rules in a program* P*, and* $\mathbf{S} \subseteq \mathcal{D}$ *any set of sets of atoms. Then,*

$$\bigcup_{S \in \mathbf{S}} \bigcup_{C \in \mathbf{C}} apply\_rule(C, S) = \bigcup_{C \in \mathbf{C}} \bigcup_{S \in \mathbf{S}} apply\_rule(C, S).$$

Proof: It is easy to see, from the definition of the function *apply_rule*, that

$\cup_{S \in \mathbf{S}} \cup_{C \in \mathbf{C}} apply\_rule(C, S)$
$= \{\bar{u} \mid (\exists S \in \mathbf{S})(\exists C \in \mathbf{C})[\bar{u} \in apply\_rule(C, S)]\}$
$= \{\bar{u} \mid (\exists C \in \mathbf{C})(\exists S \in \mathbf{S})[\bar{u} \in apply\_rule(C, S)]\}$ since the choices of S and C are independent
$= \cup_{C \in \mathbf{C}} \cup_{S \in \mathbf{S}} apply\_rule(C, S)$.

■

The following is our first main result.

Theorem 4.7 *For any program P, the operator $\mathcal{T}_P : \mathcal{D} \longrightarrow \mathcal{D}$ is continuous.*

Proof: We wish to show that for any P, $\mathcal{T}_P(\sqcup \mathbf{S}) \approx \sqcup\{\mathcal{T}_P(S) \mid S \in \mathbf{S}\}$ for any chain $\mathbf{S} = \{S_0, S_1, \ldots\}$ in $\langle \mathcal{D}, \sqsubseteq \rangle$. Let p be a predicate in the program P, where p is defined by the rules $C_1, \ldots, C_n$. Then, we have

$$\begin{aligned} \cup_{i=1}^{n} apply\_rule(C_i, \cup \mathbf{S}) &\approx \cup_{i=1}^{n} \cup_{S \in \mathbf{S}} \, apply\_rule(C_i, S) && \text{From Lemma 4.5} \\ &\approx \cup_{S \in \mathbf{S}} \cup_{i=1}^{n} \, apply\_rule(C_i, S) && \text{from Lemma 4.6} \\ &\approx \cup_{S \in \mathbf{S}} \, \widehat{\bigoplus}_{i=1}^{n} apply\_rule(C_i, S) && \text{from Axiom 1.} \end{aligned}$$

From the definition of *eval_pred*, this gives

$$eval\_pred(p, \cup \mathbf{S}) \approx \cup_{S \in \mathbf{S}} eval\_pred(p, S).$$

Since $\mathcal{T}_P(\mathbf{R})$ is nothing but the union of $eval\_pred(p, \mathbf{R})$ for each predicate p defined in P, this implies immediately that $\mathcal{T}_P(\cup \mathbf{S}) \approx \cup_{S \in \mathbf{S}} \mathcal{T}_P(S)$. Since $S_1 \sqcup S_2 = (S_1 \cup S_2)^\sharp$ for any S_1 and S_2, this can be restated as $\mathcal{T}_P(\sqcup \mathbf{S}) \approx \sqcup_{S \in \mathbf{S}} \mathcal{T}_P(S)$. The theorem follows. ■

Next, we combine this result with the Knaster-Tarski fixpoint theorem, which may be stated as follows for our purposes.

Theorem 4.8 *[42] If $\langle D, \sqsubseteq \rangle$ is a complete lattice with meet and join operations $\sqcap$ and $\sqcup$ respectively, and $f : D \longrightarrow D$ is continuous, then f has a unique least fixpoint $lfp(f) \in D$, given by $lfp(f) = \sqcup_{i \geq 0} f^i(\perp) = \sqcap\{x \mid f(x) \sqsubseteq x\}$, where $\perp$ is the least element of D.* ■

The following corollary guarantees the existence of a unique least fixpoint for our generic immediate consequence operator $\mathcal{T}_P$:

Corollary 4.9 *For any Horn program P, the operator $\mathcal{T}_P : \mathcal{D} \longrightarrow \mathcal{D}$ has a unique least fixpoint $lfp(\mathcal{T}_P)$, given by $lfp(\mathcal{T}_P) = \sqcup_{i \geq 0} \mathcal{T}_P^i(\emptyset) = \sqcap\{x \mid \mathcal{T}_P(x) \sqsubseteq x\}$, where $\sqcap$ is the meet operation of the lattice $\langle \mathcal{D}, \sqsubseteq \rangle$.* ■

5 Models Over a Domain Equipped with $\widehat{\otimes}$ and $\widehat{\oplus}$

We now develop the notions of interpretations and models, and show that every program has a "least" model that coincides with the least fixpoint.

As we saw in earlier sections, the $\mathcal{T}_P$ operator computes elements of $\mathcal{D}$. The elements of $\mathcal{D}$ are sets of elements of D, and we think of each such set as conveying some information. Intuitively, an interpretation should assign a set of elements of D to each predicate in the program, and a model is an interpretation in which for each rule "instance" (obtained using $\widehat{\otimes}$), the head should follow from the interpretation whenever the facts used to instantiate the body follow from the interpretation. It is important to note the use of the phrase "follow from" in the above: since an interpretation is seen as conveying information, a fact follows from an interpretation if adding this fact to the interpretation does not change the information content. We formalize these intuitions below.

Definition 5.1 An *interpretation* of a program is an element of $\mathcal{D}$. Let S be an element of $\mathcal{D}$, and let a be an element of D. We say that $S \models a$ iff $S \widehat{\oplus} \{a\} \approx S$. ∎

The notion of $\models$ corresponds to the notion of *truth*; a fact a is "true" in an interpretation S if $S \models a$.

Definition 5.2 An interpretation I is a *model* for a rule $H \ :- \ B_1, \ldots, B_n$ if, for every set of atoms $B_1', \ldots, B_n'$ such that $I \models B_1', \ldots, I \models B_n'$, it is also the case that $I \models H'$, where

$$H' = (\langle H, B_1, \ldots, B_n\rangle \otimes \langle H, B_1', \ldots, B_n'\rangle) \downarrow_1 .$$

An interpretation I is a model for a program if it is a model for every rule in the program.

A model M is said to be *minimal* if for every $a \in M$, $M \setminus \{a\} \not\approx M$. ∎

Intuitively, applying $\widehat{\oplus}$ to two elements of $\mathcal{D}$ gives us all the information in the two elements, and similarly, $\widehat{\otimes}$ gives us the information that is common to the two elements. We will use the $\widehat{\oplus}$ and $\widehat{\otimes}$ as the union and intersection operations on interpretations (and therefore models). The following result provides an intuitive and useful connection between our notion of "entailment" ($\models$) and the notion of "information content".

Proposition 5.1 *Let S_1 and S_2 be elements of $\mathcal{D}$, then $S_1 \sqsubseteq S_2$ if and only if, for any element a of D, it is the case that $S_1 \models a$ implies $S_2 \models a$.*

Proof: $[\Rightarrow]$: Suppose that $S_1 \sqsubseteq S_2$, i.e., $S_1 \widehat{\oplus} S_2 \approx S_2$. Then, for any $a \in \mathsf{D}$,

$$\begin{aligned} S_2 \widehat{\oplus} \{a\} &\approx (S_1 \widehat{\oplus} S_2) \widehat{\oplus} \{a\}, && \text{since } S_1 \sqsubseteq S_2 \\ &\approx (S_1 \widehat{\oplus} \{a\}) \widehat{\oplus} S_2, && \text{since, from Lemma 4.1, } \widehat{\oplus} \text{ is commutative and associative.} \end{aligned}$$

Then, if $S_1 \models a$ then $S_1 \widehat{\oplus} \{a\} \approx S_1$, whence $S_2 \widehat{\oplus} \{a\} \approx (S_1 \widehat{\oplus} \{a\}) \widehat{\oplus} S_2 \approx S_1 \widehat{\oplus} S_2 \approx S_2$, i.e., $S_2 \models a$. This shows that if $S_1 \sqsubseteq S_2$, then for any $a \in \mathsf{D}$, $S_1 \models a$ implies $S_2 \models a$.

$[\Leftarrow]$ From the definition of $\models$, it is easy to show that $S_1 \models a$ for every $a \in S_1$. Now suppose that for every $a \in \mathsf{D}$, it is the case that $S_1 \models a$ implies $S_2 \models a$. Then, since $S_1 \models a$ for every $a \in S_1$, it follows also that $S_2 \models a$ for every $a \in S_1$, or, equivalently, that $S_2 \mathbin{\widehat{\oplus}} \{a\} \approx S_2$ for every $a \in S_1$. Then, it is a straightforward consequence of the definitions of $\models$ and $\approx$, and Axiom 1, that for any $S, T \in \mathcal{D}$,

$$(S \mathbin{\widehat{\oplus}} T) \approx S \quad \Leftrightarrow \quad (S \cup T) \approx S.$$

From Axiom 2, $\approx$ is a congruence with respect to $\cup$. Therefore, we have

$$\begin{array}{ll} \cup_{a \in S_1}(S_2 \cup \{a\}) \approx \cup_{a \in S_1} S_2 & \text{since } S_2 \mathbin{\widehat{\oplus}} \{a\} \approx S_2 \text{ for every } a \in S_1 \\ \Leftrightarrow S_2 \cup (\cup_{a \in S_1}\{a\}) \approx S_2 & \\ \Leftrightarrow (S_2 \cup S_1) \approx S_2 & \text{from Axiom 2, since } \cup_{a \in S_1}\{a\} = S_1 \\ \Leftrightarrow (S_2 \mathbin{\widehat{\oplus}} S_1) \approx S_2 & \\ \Leftrightarrow S_1 \sqsubseteq S_2. & \end{array}$$

■

Corollary 5.2 *For any $S_1, S_2 \in \mathcal{D}$, $S_1 \approx S_2$ if and only if, for every $a \in \mathsf{D}$, $S_1 \models a \Leftrightarrow S_2 \models a$.* ■

Proposition 5.3 *Let S_1 and S_2 be elements of $\mathcal{D}$, then $S_1 \mathbin{\widehat{\otimes}} S_2 \sqsubseteq S_1$ and $S_1 \mathbin{\widehat{\otimes}} S_2 \sqsubseteq S_2$.*

Proof: From Axiom 4 and the definition of $\sqsubseteq$, we have $\{a \otimes b\} \sqsubseteq \{a\}$ for any $a, b \in \mathsf{D}$. Now consider any $S_1, S_2 \in \mathcal{D}$: since, from Proposition 4.3, $\widehat{\oplus}$ is the join of the lattice $\langle \mathcal{D}, \sqsubseteq \rangle$, it follows that

$$\begin{aligned} \widehat{\oplus}\ \{\{s_1 \otimes s_2\} \mid s_1 \in S_1 \text{ and } s_2 \in S_2\} &\sqsubseteq\ \widehat{\oplus}\ \{\{s_1\} \mid s_1 \in S_1\} \\ &\approx S_1. \end{aligned}$$

Thus, $\widehat{\oplus}\ \{\{s_1 \otimes s_2\} \mid s_1 \in S_1 \text{ and } s_2 \in S_2\} \sqsubseteq S_1$. Now from the definition of $\widehat{\otimes}$, we have

$$\begin{aligned} S_1 \mathbin{\widehat{\otimes}} S_2 &= \{s_1 \otimes s_2 \mid s_1 \in S_1 \text{ and } s_2 \in S_2\} \\ &= \cup\{\{s_1 \otimes s_2\} \mid s_1 \in S_1 \text{ and } s_2 \in S_2\} \\ &\approx\ \widehat{\oplus}\ \{\{s_1 \otimes s_2\} \mid s_1 \in S_1 \text{ and } s_2 \in S_2\}. \end{aligned}$$

It follows from the definition of $\sqsubseteq$ that $S_1 \mathbin{\widehat{\otimes}} S_2 \sqsubseteq S_1$. A symmetric argument establishes that $S_1 \mathbin{\widehat{\otimes}} S_2 \sqsubseteq S_2$. ■

As observed earlier, we wish to use the $\widehat{\otimes}$ operation for model intersection. The following result justifies this choice by showing that, as one would expect, a fact is true in the "intersection" of two interpretations if and only if it is true in each of them.

Proposition 5.4 *Let I_1 and I_2 be elements of $\mathcal{D}$, and let a be an element of D, then $I_1 \mathbin{\widehat{\otimes}} I_2 \models a$ if and only if $I_1 \models a$ and $I_2 \models a$.*

Proof: The *only if* direction of the proof follows immediately from Propositions 5.1 and 5.3.

To prove the *if* part, suppose that $I_1 \models a$ and $I_2 \models a$, and consider the set $I_1 \mathbin{\widehat{\otimes}} I_2$. From Axiom 2 and the definition of $\models$, we have $I_1 \mathbin{\widehat{\otimes}} I_2 \approx (I_1 \mathbin{\widehat{\oplus}} \{a\}) \mathbin{\widehat{\otimes}} (I_2 \mathbin{\widehat{\oplus}} \{a\})$. Now, from Proposition 4.2, $\widehat{\otimes}$ distributes over $\widehat{\oplus}$. It follows from this that

$$I_1 \mathbin{\widehat{\otimes}} I_2 \approx (I_1 \mathbin{\widehat{\otimes}} I_2) \mathbin{\widehat{\oplus}} (I_1 \mathbin{\widehat{\otimes}} \{a\}) \mathbin{\widehat{\oplus}} (\{a\} \mathbin{\widehat{\otimes}} I_2) \mathbin{\widehat{\oplus}} (\{a\} \mathbin{\widehat{\otimes}} \{a\})$$

From Axiom 4, $\otimes$ is idempotent, whence we have

$$I_1 \mathbin{\widehat{\otimes}} I_2 \approx (I_1 \mathbin{\widehat{\otimes}} I_2) \mathbin{\widehat{\oplus}} (I_1 \mathbin{\widehat{\otimes}} \{a\}) \mathbin{\widehat{\oplus}} (\{a\} \mathbin{\widehat{\otimes}} I_2) \mathbin{\widehat{\oplus}} \{a\}.$$

Then, if we add {a} to both sides, we get

$$\begin{aligned}(I_1 \mathbin{\widehat{\otimes}} I_2) \mathbin{\widehat{\oplus}} \{a\} &\approx \boldsymbol{(I_1 \mathbin{\widehat{\otimes}} I_2) \mathbin{\widehat{\oplus}} (I_1 \mathbin{\widehat{\otimes}} \{a\}) \mathbin{\widehat{\oplus}} (\{a\} \mathbin{\widehat{\otimes}} I_2) \mathbin{\widehat{\oplus}} \{a\}} \mathbin{\widehat{\oplus}} \{a\} \\ &\approx I_1 \mathbin{\widehat{\otimes}} I_2 \qquad \text{since, from Lemma 4.1, } \widehat{\oplus} \text{ is associative and idempotent.}\end{aligned}$$

It follows from this that $I_1 \mathbin{\widehat{\otimes}} I_2 \models a$. ∎

The following result is somewhat surprising, given our rather weak assumptions about the operator $\otimes$:

Proposition 5.5 $\widehat{\otimes}$ *is associative, commutative, and idempotent.*

Proof: To see that $\widehat{\otimes}$ is associative, note that for any $I_1, I_2, I_3 \in \mathcal{D}$, and for any $a \in \mathsf{D}$,

$$\begin{aligned}(I_1 \mathbin{\widehat{\otimes}} I_2) \mathbin{\widehat{\otimes}} I_3 \models a &\Leftrightarrow (I_1 \mathbin{\widehat{\otimes}} I_2) \models a \text{ and } I_3 \models a \qquad \text{from Proposition 5.4} \\ &\Leftrightarrow I_1 \models a \text{ and } I_2 \models a \text{ and } I_3 \models a \\ &\Leftrightarrow I_1 \models a \text{ and } (I_2 \mathbin{\widehat{\otimes}} I_3) \models a.\end{aligned}$$

It follows, from Corollary 5.2, that $(I_1 \mathbin{\widehat{\otimes}} I_2) \mathbin{\widehat{\otimes}} I_3 \approx I_1 \mathbin{\widehat{\otimes}} (I_2 \mathbin{\widehat{\otimes}} I_3)$.

To see that $\widehat{\otimes}$ is commutative, note that for any $I_1, I_2 \in \mathcal{D}$, and for any $a \in \mathsf{D}$,

$$\begin{aligned}(I_1 \mathbin{\widehat{\otimes}} I_2) \models a &\Leftrightarrow I_1 \models a \text{ and } I_2 \models a \qquad \text{from Proposition 5.4} \\ &\Leftrightarrow I_2 \models a \text{ and } I_1 \models a \\ &\Leftrightarrow (I_2 \mathbin{\widehat{\otimes}} I_1) \models a.\end{aligned}$$

It follows, from Corollary 5.2, that $I_1 \mathbin{\widehat{\otimes}} I_2 \approx I_2 \mathbin{\widehat{\otimes}} I_1$.

To see that $\widehat{\otimes}$ is idempotent, note that for any $I \in \mathcal{D}$, it follows from Proposition 5.4 that for any $a \in \mathsf{D}$, $(I \mathbin{\widehat{\otimes}} I) \models a$ if and only if $I \models a$ and $I \models a$, i.e., $(I \mathbin{\widehat{\otimes}} I) \models a$ if and only if $I \models a$. It follows, from Corollary 5.2, that $I \mathbin{\widehat{\otimes}} I \approx I$. ∎

Theorem 5.6 $\langle \mathcal{D}, \sqsubseteq \rangle$ *forms a complete distributive lattice with meet operation* $\widehat{\otimes}$ *, join operation* $\widehat{\oplus}$ *, least element* $\emptyset$ *and greatest element* D*.*

Proof: It has already been shown, from Proposition 4.3, that $\langle \mathcal{D}, \sqsubseteq \rangle$ forms a complete lattice with join operation $\widehat{\oplus}$, least element $\emptyset$, and greatest element D. From Proposition 4.2, $\widehat{\otimes}$ distributes over $\widehat{\oplus}$. It remains, therefore, to show only that $\widehat{\otimes}$ is the meet of this lattice. For this, it suffices to show that $\widehat{\otimes}$ is associative, commutative, idempotent, and satisfies the absorption laws

$$A \mathbin{\widehat{\otimes}} (A \mathbin{\widehat{\oplus}} B) \approx A \mathbin{\widehat{\oplus}} (A \mathbin{\widehat{\otimes}} B) \approx A \quad \text{for any } A, B \in \mathcal{D}.$$

The associativity, commutativity, and idempotence of $\widehat{\otimes}$ follows from Proposition 5.5. From Proposition 5.3, $A \mathbin{\widehat{\otimes}} B \sqsubseteq A$, whence we have $A \mathbin{\widehat{\oplus}} (A \mathbin{\widehat{\otimes}} B) \sqsubseteq A \mathbin{\widehat{\oplus}} A$, and from the idempotence of $\widehat{\oplus}$, it follows that

$$A \mathbin{\widehat{\oplus}} (A \mathbin{\widehat{\otimes}} B) \sqsubseteq A.$$

Since $\widehat{\oplus}$ is the join operation of the lattice $\langle \mathcal{D}, \sqsubseteq \rangle$, it follows that $A \sqsubseteq A \mathbin{\widehat{\oplus}} (A \mathbin{\widehat{\otimes}} B)$. This establishes that $A \mathbin{\widehat{\oplus}} (A \mathbin{\widehat{\otimes}} B) \approx A$. From Proposition 4.2, $\widehat{\otimes}$ distributes over $\widehat{\oplus}$, whence we have

$$\begin{aligned} A \mathbin{\widehat{\otimes}} (A \mathbin{\widehat{\oplus}} B) &\approx (A \mathbin{\widehat{\otimes}} A) \mathbin{\widehat{\oplus}} (A \mathbin{\widehat{\otimes}} B) \\ &\approx A \mathbin{\widehat{\oplus}} (A \mathbin{\widehat{\otimes}} B), \qquad \text{since from Proposition 5.5 } \widehat{\otimes} \text{ is idempotent} \\ &\approx A. \end{aligned}$$

Thus, the absorption laws are satisfied. It follows, therefore, that $\widehat{\otimes}$ is the meet operation of the lattice $\langle \mathcal{D}, \sqsubseteq \rangle$. ∎

The algebraic structure of $\mathcal{D}$ given by this theorem can be elaborated further—the following result, which is an easy consequence of Theorem 5.6 and the definitions of $\widehat{\otimes}$ and $\widehat{\oplus}$, is interesting because of the connections that it establishes with transitive closure computations (see also [18]).

Corollary 5.7 $\langle \mathcal{D}, \widehat{\otimes}, \widehat{\oplus}, \emptyset, \mathsf{D} \rangle$ *is a closed semiring.*

The following proposition states that the meet of two models is also a model; an important consequence is that every program has a unique *least* model.

Proposition 5.8 *Let* M_1 *and* M_2 *be models of a program* P*. Then,* $M_1 \mathbin{\widehat{\otimes}} M_2$ *is also a model.*

Proof: Consider a rule instance $\langle t :- t_1, \ldots, t_n \rangle$ such that $M_1 \mathbin{\widehat{\otimes}} M_2 \models t_i, 1 \leq i \leq n$. From Proposition 5.4, it follows that $M_1 \models t_i$ and $M_2 \models t_i$, for $i = 1 \ldots n$. Since M_1 and M_2 are models, $M_1 \models t$ and $M_2 \models t$. From Proposition 5.4, it follows that $M_1 \mathbin{\widehat{\otimes}} M_2 \models t$. ∎

The next result establishes an important link between the model-theoretic and fixpoint semantics.

Theorem 5.9 *An interpretation I is a model for a program P if and only if $\mathcal{T}_P(I) \sqsubseteq I$.*

Proof: *If*: Suppose that $\mathcal{T}_P(I) \sqsubseteq I$ but I is not a model of P. Then, there must be some rule $C \equiv H \ :-\ B_1, \ldots, B_n, n \geq 0$, in P, such that for some set of atoms $\{B'_1, \ldots, B'_n\} \subseteq I$, for which we have

$$\langle H'', B''_1, \ldots, B''_n \rangle = \langle H, B_1, \ldots, B_n \rangle \mathbin{\widehat{\otimes}} \langle H, B'_1, \ldots, B'_n \rangle$$

for some $B''_1, \ldots, B''_n$ in D, such that $I \models B''_i, 1 \leq i \leq n$, but $I \not\models H''$. However, from the definition of the function *apply_rule*, we have $H'' \in \mathit{apply\_rule}(C, I)$, and therefore that $\{H''\} \sqsubseteq \mathit{apply\_rule}(C, I)$. It follows, from the definition of $\mathcal{T}_P$, that $\{H''\} \sqsubseteq \mathcal{T}_P(I)$, i.e., that $I \mathbin{\widehat{\oplus}} \{H''\} \approx I$, which implies that $I \models H''$. This is a contradiction. We conclude, therefore, that I is a model of P.

Only if: Suppose that I is a model of P, i.e., for every rule $H \ :-\ B_1, \ldots, B_n$ in P, and for every set of atoms $B'_1, \ldots, B'_n$ such that $I \models B'_i$, where B'_i is an instance of B_i, $1 \leq i \leq n$, if

$$H' = (\langle H, B_1, \ldots, B_n \rangle \otimes \langle H, B'_1, \ldots, B'_n \rangle) \downarrow_1$$

then $I \models H'$, i.e., that $I \mathbin{\widehat{\oplus}} \{H'\} \approx I$. Since this is true for every such instance H' for each rule C in P, it follows from the definition of the function *apply_rule* that

$$I \mathbin{\widehat{\oplus}} \mathit{apply\_rule}(C, I) \approx I$$

or, in other words, that $\mathit{apply\_rule}(C, I) \sqsubseteq I$. It follows immediately that $\mathcal{T}_P(I) \sqsubseteq I$. ■

The main result of this section can now be proved along essentially the same lines as the corresponding result in [21]:

Corollary 5.10 *For any Horn program P, let M_P denote the least model of P (with respect to the ordering $\sqsubseteq$), then $\mathit{lfp}(\mathcal{T}_P) \approx M_P$.*

Proof: From Theorem 5.6, $\widehat{\otimes}$ is the meet of the lattice $\langle \mathcal{D}, \sqsubseteq \rangle$. Therefore, we have

$$\begin{aligned} M_P &\approx \widehat{\otimes}\ \{I \mid\ I \text{ is a model for } P\} && \text{from Proposition 5.8} \\ &\approx \widehat{\otimes}\ \{I \mid \mathcal{T}_P(I) \sqsubseteq I\} && \text{from Theorem 5.9} \\ &\approx \mathit{lfp}(\mathcal{T}_P) && \text{from Corollary 4.9.} \end{aligned}$$

■

6 Applications

This section considers the application of the framework developed in the preceding sections to a variety of fixpoint computations based on Horn logic programs. We briefly describe several proposed fixpoint computations and define the operators $\otimes$ and $^\sharp$ in each case. It is easy to show that our axioms are satisfied, although we do not do so in this abstract for lack of space. The existence of a least fixpoint and a least model and their equality then follows as a consequence of the theorems proved in earlier sections. It is usually considerably simpler to prove the existence of a least fixpoint by reasoning only about "local" properties of simple operators to show that Axioms 1-3 are satisfied, and appealing to Corollary 4.9, than by carrying out an explicit proof of continuity that involves "global" reasoning about the behavior of relatively more complex operators on chains and at limit points. A similar remark holds with regard to results about least models.

6.1 Quantitative Deduction

A quantitative deduction system is described by van Emden for probabilistic inference in logic programs [44]. In such a system, each inferred atom $p(\bar{t})$ is associated with a real number (its *weight*) $a \in (0,1]$ that gives our "confidence" in the truth of $p(\bar{t})$: an atom A with weight w is written $A : w$. Thus, D is the set of weighted ground atoms of the language under consideration, augmented with a distinguished element $\perp$ denoting failure of unification. Additionally, each clause has associated with it a real number $f \in (0,1]$, called a *factor*. A clause with head H, body $B_1, \ldots, B_n$ and factor f is written '$H \stackrel{f}{:-} B_1, \ldots, B_n$'. The idea is that given a set of weighted ground atoms $\mathbf{R}$, such a rule can be evaluated by taking a ground instance of the rule '$H' \stackrel{f}{:-} B'_1, \ldots, B'_n$', where $\{B'_1 : w_1, \ldots, B'_n : w_n\} \subseteq \mathbf{R}$. Then, the weighted atom inferred is $H' : w$, where $w = f \times \min\{w_1, \ldots, w_n\}$. When evaluating a set of rules against a set of weighted ground atoms $\mathbf{R}$, a given atom may be inferred from many different rules, in a number of different ways, and with different weights: the weight associated with this inferred atom is taken to be the largest of the different weights associated with it. In other words, inference involves minimization of weights within a rule, and maximization across rules.

To formulate this system as an instance of our framework, the instance operator $\otimes$ is defined as follows: $(t_1 : a_1) \otimes (t_2 : a_2) = t : a$, where t is the most general instance of t_1 and t_2 ($\perp$ if they have no common instance), and $a = \min(a_1, a_2)$. Given a set S of (tuples of) terms with weights, let $maxwt(t, S)$ denote the largest value in the set $\{a \mid t : a \in S\}$, if this set is nonempty, and 0 otherwise. The normalization operator $^\sharp$ is defined as:

$$S^\sharp = \{t : a \mid a = maxwt(t, S)\} \setminus \{t : a \mid t = \perp \vee a = 0\}.$$

6.2 Logic Programming with Inheritance

Aït-Kaci and Nasr describe a system that extends conventional logic programming languages by incorporating subtyping and inheritance directly into the unification algorithm [1, 2]. In this case, the notions of instance and merge are defined with respect to a type semilattice that is user-definable. Objects in D are partially ordered type structures called ψ-terms. These concepts are defined more formally in [1]: for our purposes, it suffices to note that if the set of type constructors $\Sigma_\perp$, partially ordered via a subtype relation $\trianglelefteq$, forms a join-semilattice (with $\perp \trianglelefteq a$ for all $a \in \Sigma$), then the set of ψ-terms D inherits this semilattice structure. Here, $\perp$ denotes the empty set; this implies that any term that contains an occurrence of $\perp$ also denotes the empty set, and may be identified with $\perp$. Then, it can be shown that for any pair of ψ-terms t_1 and t_2, their greatest lower bound $t_1 \bigtriangledown t_2$ (with respect to the type hierarchy) exists, is also a ψ-term, and $t_1 \bigtriangledown t_2$ can be constructed effectively [1].

In this case, the instance operator $\otimes$ is nothing but the join operator on ψ-terms, $\bigtriangledown$. The normalization operator $^\sharp$ is the identity function with the difference that it discards all terms that have $\perp$ as a subterm. Thus, $S^\sharp = S \setminus \{t \mid \perp \text{ occurs in } t\}$ for all $S \in \mathcal{D}$.

6.3 Logic Programming with Equality

A number of authors have considered extending logic programming languages by generalizing unification to "*E-unification*", i.e. unification with respect to an equational theory E (see, for example, [3, 17, 20, 39, 40]—a survey is given in [5]). Theoretical aspects of extended unification in the context of logic programming have been studied by Jaffar et al. [19] and Gallier and Raatz [14]. Here we consider how logic programming languages extended to deal with certain kinds of equality theories can be viewed as instances of our framework.

Let D denote the set of terms of the language under consideration, augmented with a distinguished element $\perp$. An equational theory E is said to be *unitary* if any two terms t_1 and t_2 that are E-unifiable have a unique (modulo equivalence under E) most general unifier. Apart from the "usual" notion of unification of first order terms, equational theories that admit unique most general unifiers include that of Boolean rings [28], and theories that are either left-distributive or right-distributive (but not both) [35]. We assume that the equational theory E under consideration is unitary. Given two E-unifiable terms t_1 and t_2, let $mgu_E(t_1, t_2)$ denote the (E-unique) most general unifier of t_1 and t_2. The instance and normalization operators are defined as follows:

$$t_1 \otimes t_2 = \begin{cases} t & \text{if } t_1 \text{ and } t_2 \text{ are } E\text{-unifiable, and } t = \theta(t_1), \text{ where } \theta = mgu_E(t_1, t_2) \\ \perp & \text{otherwise.} \end{cases}$$

The normalization operator $^\sharp$ is essentially the identity function: $S^\sharp = S \setminus \{\perp\}$.

6.4 Aggregate Computations

As an example of an aggregate computation, consider the program

```
path(A, B, N) :- edge(A, B, N).
path(A, B, N) :- path(A, C, N1), path(C, B, N2), plus(N1, N2, N).
```

Let the input consist of a relation `edge` such that $\langle a, b, n\rangle \in$ `edge` if and only if there is an edge from a node a to a node b with cost n; and a relation `plus`, such that $\langle n_1, n_2, n\rangle \in$ `plus` if and only if $n = n_1 + n_2$.

Suppose that we are interested not in all paths between any pair of nodes, but only in shortest paths. This semantics is difficult to specify in the usual Horn clause formalism. In this case, we may consider any two sets of atoms for `path` to be equivalent in terms of "information content" if they agree on the costs of the shortest paths between each pair of nodes. More formally, given a set S of tuples for `path`, let $mindist(a, b, S)$ denote the cost of the shortest path, according to S, between nodes a and b:

$$mindist(a, b, S) = \begin{cases} \min\{n \mid \langle a, b, n\rangle \in S\} & \text{if } \{n \mid \langle a, b, n\rangle \in S\} \neq \emptyset \\ \bot & \text{otherwise} \end{cases}$$

The normalization operator can now be defined as follows: For any set S of triples of the form $\langle a, b, n\rangle$, where a and b represent nodes in the graph and n is a natural number,

$$S^\sharp = \{\langle a, b, n\rangle \mid n = mindist(a, b, S) \wedge n \neq \bot\}.$$

The instance operator $\otimes$ is given by the usual first order notion of "most general instance". The fixpoint evaluation of this program with the operators $\sharp$ and $\otimes$ defined in this manner yields only the shortest paths between any pair of nodes. Notice the important operational differences between the evaluation in this case, and that in the case of evaluation using the "usual" operators: at any point, this definition retains only the shortest distances between nodes, making for more compact representations and more efficient computation; moreover, the evaluation of the program terminates in finitely many steps, as long as the graph does not contain any cycles of negative cost. It is not difficult to show that this definition of the operators satisfies our axioms.

6.5 Abstract Interpretation

An abstract interpretation may be thought of as an "execution" of the program over an abstract domain $\mathcal{D}_{abs}$, rather than the concrete domain of computation $\mathcal{D}_{conc}$. A concretization function $\mathsf{conc} : \mathcal{D}_{abs} \longrightarrow \mathcal{D}_{conc}$ maps each abstract domain element to the concrete domain element it describes. Abstract interpretation of Horn programs has been studied by various researchers (see, for example, [4, 9, 10, 25, 26, 27, 29]). Given the structural relationships between the abstract and concrete domains, the notion of $\otimes$ in $\mathcal{D}_{abs}$ can be derived without much trouble from the notion of *instance* in $\mathcal{D}_{conc}$. For example, one plausible definition is the following: given elements $\mathbf{s}$ and $\mathbf{t}$ in $\mathcal{D}_{abs}$, $\mathbf{s}$ is an instance of $\mathbf{t}$ if and only if every element in $\mathsf{conc}(\mathbf{s})$ is an instance of some element in $\mathsf{conc}(\mathbf{t})$. For a groundness analysis, for example, let $\mathbf{g}$, $\mathbf{nv}$ and $\mathbf{any}$ be abstract domain elements denoting, respectively, the set of ground terms, the set of non-variable terms, and the set of all terms. Then, $\mathbf{g}$ is an instance of $\mathbf{nv}$, which in turn is an instance of $\mathbf{any}$. The operator $\sharp$ can be defined in at least two ways: either as set union, or by means of a LUB operation in the abstract domain. The former is more precise but less efficient, the latter less precise but more efficient.

We give two examples of how abstract interpretations of logic programs may be formulated within our framework.

Example 6.1 (*Success Pattern Analysis* [24, 25, 34])
This analysis uses "depth-k abstractions" to obtain finite descriptions of infinite sets of terms. Let Ddenote the set of terms of the language under consideration, augmented with a distinguished

element $\perp$ denoting failure. A term t is said to be a *canonical depth-k abstraction* if the depth of t is at most k, and no variable occurs more than once in t or at a depth less than k [25]. Let the canonical depth-k abstraction of a term t, modulo variable renaming, be denoted by $\delta_k(t)$. The set D is the set of all canonical depth-k abstractions of the language under consideration. Then, the instance and normalization operators are defined as follows:

- The instance operator $\otimes$ is defined as follows:

  $$t_1 \otimes t_2 = \delta_k(mgi(t_1, t_2))$$

  where $mgi(t_1, t_2)$ is the most general instance of t_1 and t_2 (in the usual first order sense) if one exists, $\perp$ otherwise; and $\delta_k(\perp) = \perp$.

- The normalization operator $^\sharp$ is essentially the identity function: $S^\sharp = S \setminus \{\perp\}$.

■

Example 6.2 (*Groundness Analysis* [10, 11, 26, 27, 23])
First, consider a very simple groundness analysis that uses the special constant g to represent terms known to be definitely ground, and the constant any to represent the set of all terms of the language [11, 23]. For notational convenience, define the predicate *is_ground* as follows: Given a term t,

$$is\_ground(t) \text{ if and only if either } t = \mathsf{g}, \text{ or } t \text{ is a ground term and } t \neq \mathsf{any}.$$

The instance operator is defined as follows:

$$t_1 \otimes t_2 = \begin{cases} \perp & \text{if } t_1 = \perp \text{ or } t_2 = \perp \\ \perp & \text{if } t_1 \notin \{\mathsf{g}, \mathsf{any}\},\ t_2 \notin \{\mathsf{g}, \mathsf{any}\}, \text{ and } t_1, t_2 \text{ are not unifiable} \\ \mathsf{g} & \text{if } is\_ground(t_1) \text{ or } is\_ground(t_2) \\ \mathsf{any} & \text{otherwise.} \end{cases}$$

The normalization operator $^\sharp$ is essentially the identity function: $S^\sharp = S \setminus \{\perp\}$.

A more sophisticated groundness analysis may be obtained using propositional formulae to describe dependencies between variables. A class of formulae that has received considerable attention in this regard is the class $Pos_\perp$ of *positive propositional formulae*, which consists of the propositional constant *false*, together with formulae that can be constructed using variables and the connectives $\wedge$, $\vee$ and $\leftrightarrow$ [10, 26, 27]. To express this analysis in our framework, it suffices to have $\mathsf{D} = Pos_\perp$, $\otimes = \wedge$, and $\downarrow_1$ as $\exists$. The abstract domain $\mathcal{D}_{abs}$ is therefore $\mathcal{P}(Pos_\perp)$. Intuitively, two sets of propositional formulae convey the same amount of information if they are logically equivalent: one simple way to capture this is to identify the "information content" of a set of such formulae with the set of its logical consequences. Thus, for any $S \in \mathcal{D}_{abs}$, $S^\sharp = \{s \mid S \models s\}$. ■

7 Generalizing the *Instance* Operator $\otimes$

The discussion so far has assumed that the instance operator $\otimes$ is a function, $\otimes : \mathsf{D} \times \mathsf{D} \longrightarrow \mathsf{D}$. However, there is no *a priori* reason why this should be so. Given $t_1, t_2 \in \mathsf{D}$, it is not unreasonable to assume that $t_1 \otimes t_2$ need not be unique. This is the case, for example, for most nontrivial equational theories [35]. Intuitively, this would correspond to $\otimes$ being something like a relation rather than a function. Technically, it turns out to be more convenient to model such a generalized instance operator as a set-valued function:

$$\otimes : \mathsf{D} \times \mathsf{D} \longrightarrow \mathcal{P}(\mathsf{D}), \quad \text{i.e.,} \quad \otimes : \mathsf{D} \times \mathsf{D} \longrightarrow \mathcal{D}.$$

The "lifted" operator $\widehat{\otimes} : \mathcal{D} \times \mathcal{D} \longrightarrow \mathcal{D}$ is then defined as follows: for any $S_1, S_2 \in \mathcal{D}$,

$$S_1 \mathbin{\widehat{\otimes}} S_2 = \cup\{t_1 \otimes t_2 \mid t_1 \in S_1, t_2 \in S_2\}.$$

With this minor change, the treatment given earlier, including the results of Sections 4 and 5—in particular, Corollaries 4.9 and 5.10)—extends directly to a variety of applications. Below we give two examples.

7.1 Logic Programming in Equational Theories with Non-unique MGUs

Most nontrivial equational theories E do not admit E-unique most general unifiers [35]. For example, unification in equational theories that are commutative, or idempotent, or associative-commutative, or associative-commutative-idempotent, can have (finitely) many most general unifiers that are not E-equivalent to each other, while theories that are associative, or distributive, or associative-commutative-distributive, can have infinitely many most general unifiers.

Given an equational theory E, and two terms t_1 and t_2, let $mgu_E(t_1, t_2)$ now denote the set of most general E-unifiers of t_1 and t_2. The operators $\otimes$ and $\sharp$ can now be defined in the expected way: for any two terms t_1 and t_2,

$$t_1 \otimes t_2 = \{t \mid \exists\theta \in mgu_E(t_1, t_2) : t = \theta(t_1)\}$$

and for any set of terms S, $S^\sharp = S \setminus \{\bot\}$. The proof that these operators satisfy our axioms is straightforward.

Example 7.1 (*Natural Language Processing*)
The following describes a generator/parser for a simple fragment of English. Here, the infix function symbol $\circ$ is used to denote the concatenation of sequences of constants. Concatenation is associative, i.e., $\circ$ obeys the axiom

$$(x \circ y) \circ z = x \circ (y \circ z) \qquad \text{for every sequence of constants } x, y, z.$$

The following clauses describe the grammar under consideration:

```
sentence(N ∘ V, s(N1,V1)) :- noun_phrase(N, N1), verb_phrase(V, V1).

noun_phrase(N, np(N)) :- proper_noun(N).
noun_phrase(D ∘ N, np(D, N)) :- det(D), noun(N).
noun_phrase(D ∘ N ∘ P, np(D, N, P1)) :- det(D), noun(N), prep_phrase(P,
P1).

verb_phrase(V ∘ N, vp(V,N1)) :- verb(V), noun_phrase(N, N1).
verb_phrase(V ∘ P, vp(V1,P1)) :- verb_phrase(V, V1), prep_phrase(P, P1).

prep_phrase(P ∘ N, pp(P, N1)) :- preposition(P), noun_phrase(N, N1).
```

The vocabulary part of this grammar, specified via the relations `determiner`, `preposition`, `noun`, `proper_noun`, and `verb`, may be either be specified separately as inputs to the interpreter ("base relations", in the language of deductive databases), or as part of the program itself. For example, consider the vocabulary given by the following:

```
proper_noun(john).
noun(telescope).
noun(man).
verb(saw).
preposition(with).
determiner(a).
```

From the associativity of $\circ$, this is able to detect ambiguity in certain kinds of sentences: for example, there are distinct values of t such that the fixpoint evaluation of the above program contains

`sentence(john ∘ saw ∘ a ∘ man ∘ with ∘ a ∘ telescope,` t`)`,

for example:

$t =$ `s(np(john), vp(saw, np(a, man, pp(with, np(a, telescope)))))`

corresponding to the reading "John saw x, where $x =$ a man with a telescope"; and

$t =$ `s(np(john), vp(vp(saw, np(a, man)), pp(with, np(a, telescope))))`

corresponding to the reading "John saw x with a telescope, where $x =$ a man". ■

7.2 Logic Programming with Polymorphism and Subtyping

An earlier section discussed the application of our framework to logic programming with inheritance [1, 2]. There, it was assumed that the subtype relation over the set of type constructors is a (finite) semilattice. The theory of ψ-terms [1, 2] can be formulated in terms of order-sorted logic [36]. It turns out that order-sorted unification, i.e., unification when some types may be subtypes of other types, admits unique most general unifiers when the sort structure is a finite semilattice and there is no overloading of constructors [46]. In the presence of overloading or polymorphism, however, where there can be more than one type assignment per constructor (as is the case, for example, with the list constructor *cons*), the existence of unique most general unifiers can no longer be guaranteed unless a number of additional restrictions are imposed [37].

As the discussion earlier indicates, however, the operational aspects of this case can be handled in a straightforward way within our framework, even when the existence of unique most general unifiers cannot be guaranteed. The instance operator $\otimes$ is defined as described at the beginning of this section, and normalization is defined to be the identity function. The proof that this definition satisfies our axioms is straightforward.

8 Top-Down Computational Models for Logic Programming

A question that presents itself immediately when we consider the extended model of Horn clause computation presented in this paper is the following: Is there a similar generalization of the top-down model of computation based on SLD-resolution? Things are complicated by the fact that a bottom-up fixpoint computation can use $\sharp$ to prune the set of inferences during a computation, while a top-down strategy can use $\otimes$ to restrict the search space. Thus, the top-down and bottom-up approaches restrict the computation in distinct ways; respectively, goal-directed search and early use of $\sharp$.

One approach is to consider top-down computations that do memoing [47], i.e., that record all generated facts and goals are recorded (with the effect that $\widehat{\oplus}$ is treated as set union). Now, $\sharp$ can be applied as a final step to compute the answers. OLDT resolution [41] provides a formal basis for such computation methods, and methods such as QSQR [45] and Extension Tables [12, 48] represent specific algorithms that are based upon OLDT resolution. It is well-known (see e.g., [7]) that there is a close correspondence between bottom-up evaluation of programs rewritten using the Magic Templates transformation [32], and QSQR. Both methods generate and record the same sets of goals and facts. It is straightforward to reformulate QSQR in terms of our algebraic operators (and of course, we can simply substitute our generic fixpoint evaluation for the fixpoint evaluation phase of the Magic approach). Using the distributivity of $\widehat{\otimes}$ over $\widehat{\oplus}$ (see Theorem 5.6), it is not hard to show that these versions of QSQR and Magic are equivalent in that they compute the same answer sets for the query predicate.

9 Related Work and Future Directions

We have presented an extended model of Horn clause programs. To our knowledge, the algebraic formulation given is novel, although Carre has proposed a *path algebra* for a special class of programs based on transitive closure [8]. Giacobazzi *et al.* give an algebraic semantics for constraint logic programs that is similar in spirit to this work [16], but the details of their development are very different: for example, while we focus on a minimal set of axioms necessary to establish the necessary semantic equivalences, [16] takes a different approach by starting with closed semirings and extending this to a class of cylindric algebras. Also related is Parker's work on *partial order programing* [30], which shows how a variety of programming problems can be formulated in terms of minimizing the value of an expression, given constraints that specify a partial order over the domain of computation. Our goals are both more limited and more ambitious than Parker's: they are more limited because, unlike Parker, we restrict ourselves to the domain of "Horn-like" programs; and they are more ambitious because, within this domain, we strive to examine the structure of computations at a much finer level of granularity, and give axioms that characterize a wide variety of fixpoint computations that can be naturally expressed in the idiom of Horn programs.

Finally, an important problem is to optimize extended Horn programs. For example, the shortest path query may be specified as a logic program to compute all path-lengths between pairs of cities followed by a selection of the shortest path. How can we automatically derive the equivalent program that only retains the shortest path between a pair of cities at any point in the computation? While this is a difficult problem, requiring a set of program transformation rules over extended programs (with some well-chosen suite of choices for $\sharp$ and $\otimes$), some promising results are developed—although not in an algebraic setting—for the case of programs involving *min* and *max* aggregate operations in [15, 38].

10 Acknowledgements

The basic idea behind this paper had its roots in discussions with Suzanne Dietrich and David S. Warren. Hassan Aït-Kaci's patient explanations of the subtleties of order-sorted unification are gratefully acknowledged. Thanks are due to Michael Maher for many insightful comments on an earlier version of this paper.

References

[1] H. Aït-Kaci and R. Nasr, "Logic and Inheritance", *Proc. Thirteenth ACM Symposium on Principles of Programming Languages*, St. Petersburg Beach, FL, Jan. 1986, pp. 219-228.

[2] H. Aït-Kaci and R. Nasr, "LOGIN: A Logic Programming Language with Built-in Inheritance", *J. Logic Programming* vol 3 no. 3, Oct. 1986, pp. 185-215.

[3] R. Barbuti, M. Bellia, G. levi, and M. Martelli, "On the Integration of Logic Programming and Functional Programming", *Proc. 1984 International Symposium on Logic Programming*, Atlantic City, NJ, Feb. 1984, pp. 160-166.

[4] R. Barbuti, R. Giacobazzi, and G. Levi, "A Declarative Approach to Abstract Interpretation of Logic Programs", Technical Report TR-20/89, Dipartimento di Informatica, Università di Pisa, Pisa, Italy, 1989.

[5] M. Bellia and G. Levi, "The Relation between Logic and Functional Languages: A Survey", *J. Logic Programming* vol. 3 no. 3, Oct. 1986, pp. 217-236.

[6] G. Birkhoff, *Lattice Theory*, AMS Colloquium Publications vol. 25, 1940.

[7] F. Bry, "Query evaluation in recursive databases: Bottom-up and top-down reconciled", *IEEE Transactions on Knowledge and Data Engineering*, 5:289–312, 1990.

[8] B. Carre, *Graphs and Networks*, Clarendon Press, Oxford, England, 1979.

[9] M. Codish, D. Dams and E. Yardeni, "Bottom-up Abstract Interpretation of Logic Programs", Technical Report CS90-24, Dept. of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, Oct. 1990.

[10] A. Cortesi, G. Filé and W. Winsborough, "Optimal Groundness Analysis using Propositional Logic", *J. Logic Programming* (submitted for publication). (Preliminary version appeared in *Proc. Sixth IEEE Symposium on Logic in Computer Science*, 1991.)

[11] S. K. Debray and D. S. Warren, "Automatic Mode Inference for Logic Programs", *J. Logic Programming* vol. 5 no. 3 (Sept. 1988), pp. 207-229.

[12] S. W. Dietrich, "Extension tables: Memo relations in logic programming", *Proc. Symposium on Logic Programming*, pages 264–272, 1987.

[13] M. Falaschi, G. Levi, C. Palamidessi, and M. Martelli, "Declarative Modeling of the Operational Behavior of Logic Languages", *Theoretical Computer Science 69* (1989), pp. 289-318, North Holland.

[14] J. H. Gallier and S. Raatz, "SLD-Resolution Methods for Horn Clauses with Equality Based on E-Unification", in *Proc. IEEE Symposium on Logic Programming*, Salt Lake City, Utah, Sept. 1986, pp. 168-179.

[15] S. Ganguly, S. Greco, and C. Zaniolo, "Minimum and maximum predicates in logic programming", *Procedings of the ACM Symposium on Principles of Database Systems*, 1990.

[16] R. Giacobazzi, S. K. Debray, and G. Levi, "A Generalized Semantics for Constraint Logic Programs", in *Proc. International Conference on Fifth Generation Computer Systems*, Tokyo, 1992, pp. 581–591.

[17] J. A. Goguen and J. Meseguer, "Eqlog: Equality, Types, and Generic Modules for Logic Programming", in *Functional and Logic Programming*, eds. D. DeGroot and G. Lindstrom, Prentice Hall, 1985.

[18] Y.E. Ioannidis and E. Wong, "An Algebraic Approach to Recursive Inference", *Proc. First Int. Conf. Expert Database Systems*, Charleston, SC, 1987, pp. 295–309.

[19] J. Jaffar, J.-L. Lassez, and M. Maher, "A Theory of Complete Logic Programs with Equality", *J. Logic Programming* vol. 1 no. 3, Oct. 1984, pp. 211-224.

[20] W. A. Kornfeld, "Equality for Prolog", in *Proc. Eighth IJCAI*, Karlsruhe, W. Germany, 1983, pp. 514-519.

[21] J. W. Lloyd, *Foundations of Logic Programming*, Springer Verlag, 1984.

[22] M. Maher and R. Ramakrishnan, "Déjà Vu in Fixpoints of Logic Programs", Proc. NACLP-89, Cleveland, OH, Oct. 1989.

[23] H. Mannila and E. Ukkonen, "Flow Analysis of Prolog Programs", *Proc. Fourth IEEE Symposium on Logic Programming*, San Francisco, CA, Sept. 1987.

[24] K. Marriott and H. Søndergaard, "On Describing Success Patterns of Logic Programs", Technical Report 88/12, Dept. of Computer Science, University of Melbourne, Australia, May 1988.

[25] K. Marriott and H. Søndergaard, "Bottom-up Abstract Interpretation of Logic Programs", *Proc. Fifth International Conference on Logic Programming*, Seattle, 1988. MIT Press, pp. 733-748.

[26] K. Marriott and H. Søndergaard, "Precise and Efficient Groundness Analysis of Logic Programs", *ACM Letters on Programming Languages and Systems* vol. 2 nos. 1–4, March–Dec. 1993, pp. 181–196.

[27] K. Marriott, H. Søndergaard and N. D. Jones, "Denotational Abstract Interpretation of Logic Programs", *ACM Transactions on Programming Languages and Systems* vol. 16 no. 3, May 1994, pp. 607–648.

[28] U. Martin and T. Nipkow, "Unification in Boolean Rings", in *Proc. 8th International Conference on Automated Deduction*, Oxford, July 1986. Springer-Verlag LNCS vol. 230, pp. 506-513.

[29] K. Muthukumar and M. Hermenegildo, "Determination of Variable Dependence Information Through Abstract Interpretation", *J. Logic Programming* (special issue on Abstract Interpretation) vol. 13 nos. 2 & 3, July 1992, pp. 315–347.

[30] D. S. Parker, "Partial Order Programming", in *Proc. Sixteenth ACM Symposium on Principles of Programming Languages*, Austin, TX, Jan. 1989, pp. 260-266.

[31] G. D. Plotkin, "A Note on Inductive Generalization" in *Machine Intelligence 5*, B. Meltzer and D. Michie (eds.), Elsevier, New York, 1970, pp. 153-162.

[32] R. Ramakrishnan, "Magic Templates: A Spellbinding Approach to Logic Programs", *Proc. Fifth International Conference on Logic Programs*, Seattle, Aug. 1988, pp. 140-159. MIT Press.

[33] J. C. Reynolds, "Transformational Systems and the Algebraic Structure of Atomic Formulas", in *Machine Intelligence 5*, B. Meltzer and D. Michie (eds.), Elsevier, New York, 1970, pp. 135-151.

[34] T. Sato and H. Tamaki, "Enumeration of Success Patterns in Logic Programs", *Theoretical Computer Science* vol. 34, 1984, pp. 227-240.

[35] J. H. Siekmann, “Unification Theory”, in *Unification*, ed. C. Kirchner, Academic Press, 1990, pp. 1-68.

[36] G. Smolka and H. Aït-Kaci, “Inheritance Hierarchies: Semantics and Unification”, in *Unification*, ed. C. Kirchner, Academic Press, 1990, pp. 489-516.

[37] G. Smolka, W. Nutt, J. Goguen, and J. Meseguer, “Order-Sorted Equational Computation”, in *Resolution of Equations in Algebraic Structures, vol. 2: Rewriting Techniques*, eds. H. Aït-Kaci and M. Nivat, Academic Press, Cambridge, MA, 1989, pp. 297-367.

[38] S. Sudarshan and R. Ramakrishnan, “Aggregation and Relevance in Deductive Databases”, *Proc. International Conference on Very Large Databases*, Barcelona, Spain, September 1991.

[39] P. A. Subrahmanyam and J.-H. You, “Conceptual Basis and Evaluation Strategies for Integrating Functional and Logic Programming”, *Proc. 1984 International Symposium on Logic Programming*, Atlantic City, NJ, Feb. 1984, pp. 144-153.

[40] H. Tamaki, “Semantics of a Logic Programming Language with Equality”, *Proc. 1984 International Symposium on Logic Programming*, Atlantic City, NJ, Feb. 1984, pp. 259-264.

[41] H. Tamaki and T. Sato, “OLD Resolution with Tabulation”, Booktitle = iclp86, *Proc. International Conference on Logic Programming, Lecture Notes in Computer Science 225, Springer-Verlag*, 1986, pp. 84–98.

[42] A. Tarski, “A Lattice-Theoretic Fixpoint Theorem and its Applications”, *Pacific J. Math* 5 (1955), 285-309.

[43] M. H. van Emden and R. A. Kowalski, “The Semantics of Predicate Logic as a Programming Language”, *J. ACM 23*, 4 (Oct. 1976), pp. 733-742.

[44] M. H. van Emden, “Quantitative Deduction and its Fixpoint Theory”, *J. Logic Programming*, vol. 3 no. 1, April 1986, pp. 37-53.

[45] Laurent Vieille, “Database complete proof procedures based on SLD-resolution”, *Proc. Fourth International Conference on Logic Programming*, pages 74–103, 1987.

[46] Ch. Walther, “Unification in Many-sorted Theories”, in *Proc. 6th European Conference on Artificial Intelligence*, 1985, North-Holland, pp. 383-392.

[47] David S. Warren, “Memoing for logic programs”, *Communications of the ACM*, 35(3), March 1992.

[48] David S. Warren, “The XWAM: A Machine That Integrates Prolog and Deductive Database Query Evaluation”, *Technical Report 89/25, Department of Computer Science, SUNY at StonyBrook*, October 1989.