Annals of Mathematics and Artificial Intelligence (2024) 92:505–533
https://doi.org/10.1007/s10472-024-09925-x

Sequential composition of propositional logic programs

Christian Antić[1]

Accepted: 9 January 2024 / Published online: 15 February 2024

Abstract
This paper introduces and studies the sequential composition and decomposition of propositional logic programs. We show that acyclic programs can be decomposed into single-rule programs and provide a general decomposition result for arbitrary programs. We show that the immediate consequence operator of a program can be represented via composition which allows us to compute its least model without any explicit reference to operators. This bridges the conceptual gap between the syntax and semantics of a propositional logic program in a mathematically satisfactory way.

Keywords Modular logic programming · Algebra of logic programs

Mathematics Subject Classification (2010) 68T27 · 03B70

1 Introduction

Rule-based reasoning is an essential part of human intelligence prominently formalized in artificial intelligence research via logic programs (cf. [8, 9, 37, 44, 46, 56]). Logic programming is a well-established subfield of theoretical computer science and AI with applications to such diverse fields as expert systems, database theory, diagnosis, planning, learning, natural language processing, and many others (cf. [11, 24, 40, 53]).

The propositional Horn fragment studied in this paper is particularly relevant in database theory via the query language datalog (see e.g. [21]) and in answer set programming see e.g. [43], a prominent and successful dialect of logic programming incorporating negation as failure [23] and many other constructs such as aggregates (cf. [33, 52]), external sources [32], and description logic atoms [31].

Describing complex programs as the composition of simpler ones is a common strategy in computer programming and logic programming is no exception as is witnessed by the large amount of research on **modular logic programming** in the 1980s and 1990s (e.g. [13, 15–19, 27, 34, 36, 47, 51]), and more recently on modular answer set programming (e.g. [25, 49, 50]).

✉ Christian Antić
christian.antic@icloud.com

[1] Vienna University of Technology, Vienna, Austria

The **purpose of this paper** is to add to the logic programmer's repertoire another modularity operation in the form of the sequential composition of propositional logic programs, which is naturally induced by their rule-like structure. The close relationship to the modularity operations introduced in [51] and [19] is discussed in detail in § 7.

The **motivation** for introducing a novel operation is because we feel the need for a *syntactic* composition operation complementing the semantic ones from the literature[1] — in § 7 we show that the semantic composition operators can be recovered from the syntactic one (but not vice versa which is evident from their definition). Thus, in a sense, syntactic composition is more fundamental than its semantic counterpart.

The **main results** of this paper can be summarized as follows:

(1) We introduce the sequential composition of propositional logic programs and show that the space of all propositional logic programs over some fixed alphabet is closed under composition which means that composition is a total function defined for all pairs of programs. Composition is in general non-associative (Example 10). The composition of propositional Krom programs consisting only of rules with at most one body atom is associative and gives rise to the algebraic structure of a monoid, and the restricted class of proper propositional Krom programs consisting only of rules with exactly one body atom yields an idempotent semiring (Theorem 12).
(2) In § 5, we study *decompositions* of programs. The main results here are that acyclic programs (cf. [10]) can be decomposed into a product of single-rule programs (Theorem 31), followed by some general results on decompositions of programs in § 5.4.
(3) On the semantic side, the van Emden-Kowalski immediate consequence operator T_P is at the core of logic programming and we show that it can be represented via composition (Theorem 35), which allows us to compute its least model semantics without any explicit reference to operators (Theorem 40). This bridges the conceptual gap between the syntax and semantics of a propositional logic program in a mathematically satisfactory way.

In a broader sense, this paper is a further step towards an algebra of logic programs and in the future we plan to adapt and generalize the methods of this paper to wider classes of programs, most importantly to first-, and higher-order logic programs [8, 22, 44, 48], and non-monotonic logic programs under the stable model [35] or answer set semantics and extensions thereof (cf. [11, 14, 30, 43] — see [6]).

2 Preliminaries

In this section, we first recall the algebraic structures occurring in the rest of the paper, and then we recall the syntax and semantics of propositional logic programs.

2.1 Algebraic structures

We recall some basic algebraic notions and notations by mainly following the lines of [38].

Given two sets A and B, we write $A \subseteq_k B$ in case A is a subset of B with k elements, for some non-negative integer k.We denote the ***identity function*** on a set A by Id_A. A ***permutation*** of a set A is any mapping $A \to A$ which is one-to-one and onto. We denote the

[1] This is partly motivated by recent work on algebraic logic program synthesis via analogical proportions in [4].

composition of two functions $f : A \to A$ and $g : A \to A$ by $f \circ g$ with the usual definition $(f \circ g)(x) = f(g(x))$.

We call a binary relation $\leq$ ***reflexive*** if $x \leq x$, ***anti-symmetric*** if $x \leq y$ and $y \leq x$ implies $x = y$, and ***transitive*** if $x \leq y$ and $y \leq z$ implies $x \leq z$, for all x, y, z. A ***partially ordered set*** (or ***poset***) is a set L together with a reflexive, transitive, and anti-symmetric binary relation $\leq$ on L. A ***prefixed point*** of an operator f on a poset L is any element $x \in L$ such that $f(x) \leq x$; moreover, we call any $x \in L$ a ***fixed point*** of f if $f(x) = x$.

A ***magma*** is a set M together with a binary operation $\cdot$ on M. We call $(M, \cdot, 1)$ a ***unital magma*** if it contains a unit element 1 such that $1x = x1 = x$ holds for all $x \in M$. A ***semigroup*** is a magma $(S, \cdot)$ in which $\cdot$ is associative. A ***monoid*** is a semigroup containing a unit element 1 such that $1x = x1 = x$ holds for all x. A ***group*** is a monoid which contains an inverse x^{-1} for every x such that $xx^{-1} = x^{-1}x = 1$. A ***left*** (resp., ***right***) ***zero*** is an element 0 such that $0x = 0$ (resp., $x0 = 0$) holds for all $x \in S$. An ***ordered semigroup*** is a semigroup S together with a partial order $\leq$ that is compatible with the semigroup operation, meaning that $x \leq y$ implies $zx \leq zy$ and $xz \leq yz$ for all $x, y, z \in S$. An ***ordered monoid*** is defined in the obvious way. A non-empty subset I of S is called a ***left*** (resp., ***right***) ***ideal*** if $SI \subseteq I$ (resp., $IS \subseteq I$), and a (***two-sided***) ***ideal*** if it is both a left and right ideal. An element $x \in S$ is ***idempotent*** if $x \cdot x = x$.

A ***seminearring*** is a set S together with two binary operations $+$ and $\cdot$ on S, and a constant $0 \in S$, such that $(S, +, 0)$ is a monoid and $(S, \cdot)$ is a semigroup satisfying the following laws:

(1) $(x + y) \cdot z = x \cdot z + y \cdot z$ for all $x, y, z \in S$ (right-distributivity); and
(2) $0 \cdot x = 0$ for all $x \in S$ (left zero).

We say that S is ***idempotent*** if $x + x = x$ holds for all $x \in S$. Typical examples of seminearrings are given by the set of all mappings on a monoid together with composition of mappings, point-wise addition of mappings, and the zero function.

A ***semiring*** is a seminearring $(S, +, \cdot, 0)$ such that $+$ is commutative and additionally to the laws of a seminearring the following laws are satisfied:

(1) $x \cdot (y + z) = x \cdot y + x \cdot z$ for all $x, y, z \in S$ (left-distributivity); and
(2) $x \cdot 0 = 0$ for all $x \in S$ (right zero).

For example, $(\mathbb{N}, +, \cdot, 0)$ and $(2^A, \cup, \cap, \emptyset)$ are semirings.

2.2 Propositional logic programs

We recall the syntax and semantics of propositional logic programs.

2.2.1 Syntax

In the rest of the paper, A denotes a finite alphabet of propositional atoms.

A (***propositional Horn logic***) ***program*** over A is a finite set of ***rules*** of the form

$$a_0 \leftarrow a_1, \ldots, a_k, \quad k \geq 0, \tag{1}$$

where $a_0, \ldots, a_k \in A$ are propositional atoms. It will be convenient to define, for a rule r of the form (1),

$$h(r) := \{a_0\} \quad \text{and} \quad b(r) := \{a_1, \ldots, a_k\}$$

extended to programs by

$$h(P) := \bigcup_{r \in P} h(r) \quad \text{and} \quad b(P) := \bigcup_{r \in P} b(r).$$

In this case, the ***size*** of r is k and denoted by $sz(r)$.

A ***fact*** is a rule with empty body and a ***proper rule*** is a rule which is not a fact. We denote the facts and proper rules in P by $f(P)$ and $p(P)$, respectively.

We call a rule r of the form (1) ***Krom***[2] if it has at most one body atom, and we call r ***binary*** if it contains at most two body atoms. A ***tautology*** is any Krom rule of the form $a \leftarrow a$, for $a \in A$. We call a program ***Krom*** if it contains only Krom rules, and we call it ***binary*** if it consists only of binary rules.

A program is called a ***single-rule program*** if it contains exactly one non-tautological rule.

We call a program ***minimalist*** if it contains at most one rule for each rule head.

Given two programs P and R, we say that P ***depends on*** R if the intersection of $b(P)$ and $h(R)$ is not empty.

We call P ***acyclic*** if there is a mapping $\ell : A \to \{0, 1, 2, \ldots\}$ such that for each rule $r \in P$, we have $\ell(h(r)) > \ell(b(r))$, and in this case we call ℓ a ***level mapping*** for P. Of course, every level mapping ℓ induces an ordering on rules via $r \leq_\ell s$ if $\ell(h(r)) \leq \ell(h(s))$. We can transform this ordering into a total ordering by arbitrarily choosing a particular linear ordering within each level, that is, if $\ell(a) = \ell(b)$ then we can choose between $a <_\ell b$ or $b <_\ell a$.

Example 1 The program

$$P := \left\{ \begin{array}{l} a \\ b \leftarrow a \\ c \leftarrow a, b \end{array} \right\}$$

is acyclic with level mapping $\ell(a) = 0$, $\ell(b) = 1$, $\ell(c) = 2$. Adding the rule $a \leftarrow c$ to P yields a non-acyclic program since $\ell(a) \not> \ell(c)$.

Define the ***dual*** of P by

$$P^d := f(P) \cup \{b \leftarrow h(r) \mid r \in p(P), b \in b(r)\}.$$

Roughly, we obtain the dual of a program by reversing all the arrows of its proper rules.

2.2.2 Semantics

An ***interpretation*** is any set of atoms from A. We define the ***entailment relation***, for every interpretation I, inductively as follows: (i) for an atom a, $I \models a$ if $a \in I$; (ii) for a set of atoms B, $I \models B$ if $B \subseteq I$; (iii) for a rule r of the form (1), $I \models r$ if $I \models b(r)$ implies $I \models h(r)$; and, finally, (iv) for a propositional logic program P, $I \models P$ if $I \models r$ holds for each rule $r \in P$. In case $I \models P$, we call I a ***model*** of P. The set of all models of P has a least element with respect to set inclusion called the ***least model*** of P and denoted by $LM(P)$. We say that P and R are ***equivalent*** if $LM(P) = LM(R)$.

Define the ***van Emden-Kowalski operator*** of P, for every interpretation I, by

$$T_P(I) := \{h(r) \mid r \in P : I \models b(r)\}.$$

[2] Krom rules were first introduced and studied in [41] and are sometimes called "binary" in the literature; we prefer "Krom" here since we reserve "binary" for programs with two body atoms.

The van Emden-Kowalski operator is at the core of logic programming since we have the following well-known operational characterization of models [57]:

Proposition 2 *An interpretation I is a model of P iff I is a prefixed point of T_P.*

We call an interpretation I a ***supported model*** of P if I is a fixed point of T_P. We say that P and R are ***subsumption equivalent*** [45] if $T_P = T_R$, denoted by $P \equiv_{ss} R$.

The ***least fixed point*** computation of T_P is defined by

$$\begin{aligned} T_P^0 &= \emptyset, \\ T_P^{n+1} &= T_P(T_P^n), \\ T_P^\infty &= \bigcup_{n \geq 0} T_P^n. \end{aligned}$$

The following constructive characterization of least models is due to [57]:

Proposition 3 *The least model of a propositional logic program coincides with the least fixed point of its associated van Emden-Kowalski operator, that is, for any program P we have*

$$LM(P) = T_P^\infty. \tag{2}$$

Example 4 Reconsider the program P of Example 1. Intuitively, we expect the least model of P to include all the atoms a, b, and c. The following least fixed point computation shows that this is indeed the case:

$$\begin{aligned} &T_P(\emptyset) = \{a\} \\ &T_P(\{a\}) = \{a, b\} \\ &T_P(\{a, b\}) = \{a, b, c\} \\ &T_P(\{a, b, c\}) = \{a, b, c\}. \end{aligned}$$

3 Sequential composition

We are interested in the algebraic structure of the space of all propositional logic programs. For this we define the sequential composition operation on programs and show in Theorem 9 that the so-obtained space is closed under such compositions. This will allow us later to decompose programs into simpler ones (§ 5), to represent the immediate consequence operator on syntactic level (§ 6.1), and to define an algebraic semantics of programs without any explicit reference to operators (§ 6.2).

Notation 5 *In the rest of the paper, P and R denote propositional logic programs over some joint alphabet A.*

We are ready to introduce the main notion of the paper.

Definition 6 We define the (***sequential***) ***composition*** of P and R by

$$P \circ R = \left\{ h(r) \leftarrow b(S) \;\middle|\; r \in P, S \subseteq_{sz(r)} R, h(S) = b(r) \right\}.$$

We will write PR in case the composition operation is understood.

Roughly, we obtain the composition of P and R by resolving all body atoms in P with 'matching' rule heads of R. The following example shows why we use $\subseteq_{sz(r)}$ instead of $\subseteq$ in the above definition.

Example 7 Let r and R be given by

$$r := a \leftarrow b \quad \text{and} \quad R := \left\{ \begin{array}{l} b \leftarrow c \\ b \leftarrow d \end{array} \right\}.$$

Since r consists of a single body atom, we have $sz(r) = 1$. This means that $\{r\} \circ R$ consists of all rules $a \leftarrow b(S)$ so that $S \subseteq_1 R$ (i.e., S consists of a single rule) and $h(S) = \{b\}$. This yields

$$\{r\} \circ R = \left\{ \begin{array}{l} a \leftarrow c \\ a \leftarrow d \end{array} \right\}.$$

This coincides with what we intuitively expect from the composition of r and R. On the other hand, if we define composition using $\subseteq$ instead of $\subseteq_{sz(r)}$, then the above requirement changes to $S \subseteq R$ such that $h(S) = \{b\}$, which means that we can now put $S = R$ adding the undesired rule $a \leftarrow b(S) = a \leftarrow c, d$ to $\{r\} \circ R$.

Notice that we can reformulate sequential composition as

$$P \circ R = \bigcup_{r \in P} (\{r\} \circ R), \tag{3}$$

which directly implies right-distributivity of composition, that is,

$$(P \cup Q) \circ R = (P \circ R) \cup (Q \circ R) \quad \text{holds for all propositional logic programs } P, Q, R. \tag{4}$$

Example 8 The following counter-example shows that left-distributivity fails in general:

$$\{a \leftarrow b, c\} \circ (\{b\} \cup \{c\}) = \{a\} \quad \text{whereas} \quad (\{a \leftarrow b, c\} \circ \{b\}) \cup (\{a \leftarrow b, c\} \circ \{c\}) = \emptyset.$$

We can write P as the union of its facts and proper rules, that is,

$$P = f(P) \cup p(P). \tag{5}$$

Hence, we can rewrite the composition of P and R as

$$PR = (f(P) \cup p(P))R \overset{(4)}{=} f(P)R \cup p(P)R = f(P) \cup p(P)R, \tag{6}$$

which shows that the facts in P are preserved by composition, that is, we have

$$f(P) \subseteq f(PR). \tag{7}$$

The facts of the composition of to two programs is given by

$$f(PR) \overset{(6)}{=} f(f(P) \cup p(P)R) = f(f(P)) \cup f(p(P)R) = f(P) \cup p(P)f(R).$$

We can compute the heads and bodies via

$$h(P) = PA \quad \text{and} \quad b(P) = p(P)^d A. \tag{8}$$

Moreover, we have

$$h(PR) \subseteq h(P) \quad \text{and} \quad b(PR) \subseteq b(R).$$

Define the ***unit program*** (over A) by the propositional Krom program

$$1_A := \{a \leftarrow a \mid a \in A\}.$$

In the sequel, we will often omit the reference to A.

We are now ready to state the main structural result of the paper:

Theorem 9 *The space of all propositional logic programs over some fixed alphabet forms a finite unital magma with respect to composition ordered by set inclusion with the neutral element given by the unit program. Moreover, the empty program is a left zero and composition distributes from the right over union, that is, for any programs P, Q, R we have*

$$P1 = 1P = P \tag{9}$$

$$\emptyset P = \emptyset \tag{10}$$

$$(P \cup R)Q = (PQ) \cup (RQ). \tag{11}$$

Proof The space of all propositional logic programs is obviously closed under composition, which shows that it forms a magma.

We proceed by proving that 1 is neutral with respect to composition. By definition of composition, we have

$$P1 = \{h(r) \leftarrow b(S) \mid r \in P, S \subseteq_{sz(r)} 1, h(S) = b(r)\}.$$

Now, by definition of 1 and $S \subseteq_{sz(r)} 1$, we have $h(S) = b(S)$ and therefore $b(S) = b(r)$. Hence,

$$P1 = P.$$

Similarly, we have

$$1P = \{h(r) \leftarrow b(S) \mid r \in 1, S \subseteq_1 P, h(S) = b(r)\}.$$

As S is a subset of P with a single element, S is a singleton $S = \{s\}$, for some rule s with $h(s) = b(r)$ and, since $b(r) = h(r)$ holds for every rule in 1, $h(s) = h(r)$. Hence,

$$\begin{aligned} 1P &= \{h(r) \leftarrow b(s) \mid r \in 1, s \in P, h(s) = b(r)\} \\ &= \{h(s) \leftarrow b(s) \mid s \in P\} \\ &= P. \end{aligned}$$

This shows that 1 is neutral with respect to composition. That composition is ordered by set inclusion is obvious. We now turn our attention to the operation of union. In (4) we argued for the right-distributivity of composition. That the empty set is a left zero is obvious. As union is idempotent, the set of all propositional logic programs forms the structure of a finite idempotent seminearring. □

The following example shows that, unfortunately, composition is *not* associative.[3]

Example 10 Consider the rule

$$r := a \leftarrow b, c$$

[3] In Theorem 11 we will see that composition is associative for minimalist programs and in Corollary 36 we will see that it is associative modulo subsumption equivalence.

and the programs

$$P := \left\{ \begin{array}{l} b \leftarrow b \\ c \leftarrow b, c \end{array} \right\} \quad \text{and} \quad R := \left\{ \begin{array}{l} b \leftarrow d \\ b \leftarrow e \\ c \leftarrow f \end{array} \right\}.$$

Let us compute $(\{r\}P)R$. We first compute $\{r\}P$ by noting that the rule r has two body atoms and has therefore size 2, which means that there is only a single choice of subprogram S of P with two rules, namely $S = P$; this yields

$$\{r\}P = \{h(r) \leftarrow b(P)\} = \{a \leftarrow b, c\} = \{r\}.$$

Next we compute $(\{r\}P)R = \{r\}R$ by noting that now there are two possible $S_1, S_2 \subseteq_2 R$ with $h(S_1) = h(S_2) = b(r)$, given by

$$S_1 = \left\{ \begin{array}{l} b \leftarrow d \\ c \leftarrow f \end{array} \right\} \quad \text{and} \quad S_2 = \left\{ \begin{array}{l} b \leftarrow e \\ c \leftarrow f \end{array} \right\}.$$

This yields

$$\{r\}R = \left\{ \begin{array}{l} h(r) \leftarrow b(S_1) \\ h(r) \leftarrow b(S_2) \end{array} \right\} = \left\{ \begin{array}{l} a \leftarrow d, f \\ a \leftarrow e, f \end{array} \right\}.$$

Let us now compute $\{r\}(PR)$. We first compute PR. By (3) we have

$$PR = \{b \leftarrow b\}R \cup \{c \leftarrow b, c\}R.$$

We easily obtain

$$\{b \leftarrow b\}R = \left\{ \begin{array}{l} b \leftarrow d \\ b \leftarrow e \end{array} \right\}.$$

Similar computations as above show

$$\{c \leftarrow b, c\}R = \left\{ \begin{array}{l} c \leftarrow d, f \\ c \leftarrow d, e \end{array} \right\}.$$

So in total we have

$$PR = \left\{ \begin{array}{l} b \leftarrow d \\ b \leftarrow e \\ c \leftarrow d, f \\ c \leftarrow d, e \end{array} \right\}.$$

To compute $\{r\}(PR)$, we therefore see that there are four two rule subprograms $S_1, S_2, S_4, S_4 \subseteq_2 PR$ with b or c in their heads given by

$$S_1 = \left\{ \begin{array}{l} b \leftarrow d \\ c \leftarrow d, f \end{array} \right\} \quad S_2 = \left\{ \begin{array}{l} b \leftarrow d \\ c \leftarrow d, e \end{array} \right\} \quad S_3 = \left\{ \begin{array}{l} b \leftarrow e \\ c \leftarrow d, f \end{array} \right\} \quad S_4 = \left\{ \begin{array}{l} b \leftarrow e \\ c \leftarrow d, f \end{array} \right\}.$$

Hence, we have

$$\{r\}PR = \left\{ \begin{array}{l} h(r) \leftarrow b(S_1) \\ h(r) \leftarrow b(S_2) \\ h(r) \leftarrow b(S_3) \\ h(r) \leftarrow b(S_4) \end{array} \right\} = \left\{ \begin{array}{l} a \leftarrow d, f \\ a \leftarrow d, e \\ a \leftarrow d, e, f \end{array} \right\}.$$

We have thus shown

$$\{r\}(PR) = \left\{ \begin{array}{l} a \leftarrow d, f \\ a \leftarrow e, f \\ a \leftarrow d, e, f \end{array} \right\} \neq \left\{ \begin{array}{l} a \leftarrow d, f \\ a \leftarrow e, f \end{array} \right\} = (\{r\}P)R.$$

4 Restricted classes of programs

Now that we have a basic understanding of the sequential composition of programs, we continue by studying composition in some restricted classes of programs like minimalist (§ 4.1) and Krom programs (§ 4.2) where we find that associativity holds. Having a basic understanding of those restricted classes of programs will be useful later when we deal with decompositions of programs where factors of programs often have one of those restricted forms.

4.1 Minimalist programs

Recall that we call a program ***minimalist*** if it contains at most one rule for each head atom. Theorem 11 shows that minimalist programs satisfy a form of associativity.

Given a minimalist program M, in the computation of $P \circ M$ we can omit the reference to r in $\subseteq_{sz(r)}$ in Definition 6, that is, we have

$$P \circ M = \{h(r) \leftarrow b(S) \mid r \in P, S \subseteq M, h(S) = b(r)\}.$$

In Example 10 we have seen that composition is not associative in general. The situation changes for minimalist programs.

Theorem 11 *For any programs P, M, N if M and N are minimalist, then*

$$(PM)N = P(MN). \tag{12}$$

Proof As a consequence of (3), a rule r is in $(PM)N$ iff

$$\{r\} = \left(\{a_0 \leftarrow a_1, \ldots, a_k\} \circ \left\{ \begin{array}{l} a_1 \leftarrow B_1 \\ \vdots \\ a_k \leftarrow B_k \end{array} \right\} \right) \circ \left\{ \begin{array}{l} b_1 \leftarrow C_1 \\ \vdots \\ b_m \leftarrow C_m \end{array} \right\}, \tag{13}$$

for some rules $a_0 \leftarrow a_1, \ldots, a_k \in P$, $k \geq 0$, $b_1 \leftarrow B_1, \ldots, b_k \leftarrow B_k \in M$, $b_1 \leftarrow C_1, \ldots, b_m \leftarrow C_m \in N$, and $B_1 \cup \ldots \cup B_k = \{b_1, \ldots, b_m\}$, $m \geq 0$. Since

$$\left\{ \begin{array}{l} a_1 \leftarrow B_1 \\ \vdots \\ a_k \leftarrow B_k \end{array} \right\} \quad \text{and} \quad \left\{ \begin{array}{l} b_1 \leftarrow C_1 \\ \vdots \\ b_m \leftarrow C_m \end{array} \right\}$$

are minimalist, we can rewrite (13) as

$$\{r\} = \{a_0 \leftarrow a_1, \ldots, a_k\} \circ \left(\left\{ \begin{array}{l} a_1 \leftarrow B_1 \\ \vdots \\ a_k \leftarrow B_k \end{array} \right\} \circ \left\{ \begin{array}{l} b_1 \leftarrow C_1 \\ \vdots \\ b_m \leftarrow C_m \end{array} \right\} \right),$$

which shows $r \in P(MN)$. The proof that every rule in $P(MN)$ is in $(PM)N$ is analogous. □

4.2 Krom programs

Recall that we call a program ***Krom*** if it contains only rules with at most one body atom. This includes interpretations, unit programs, and permutations. Krom programs often appear as factors in decompositions of programs and it is therefore beneficial to know their basic algebraic properties (Theorem 12). First notice that for any Krom program K and any program P, composition simplifies to

$$K \circ P = f(K) \cup \{a \leftarrow B \mid a \leftarrow b \in p(K) \text{ and } b \leftarrow B \in P\}.$$

We have the following structural result as a specialization of Theorem 9:

Theorem 12 *The space of all propositional Krom programs forms a monoid with respect to composition with the neutral element given by the unit program, and it forms a seminearring with respect to sequential composition and union. More generally, we have*

$$K(PR) = (KP)R, \tag{14}$$

for arbitrary programs P and R. Moreover, Krom programs distribute from the left, that is, for any programs P and R, we have

$$K(P \cup R) = KP \cup KR. \tag{15}$$

This implies that the space of proper propositional Krom programs forms a finite idempotent semiring.

Proof We first prove (14), which implies associativity for propositional Krom programs. A rule r is in $K(PR)$ iff either $r = a \in K$, in which case we have $r \in (KP)R$, or

$$\{r\} = \{a \leftarrow b\} \circ \left(\{b \leftarrow b_1, \ldots, b_k\} \circ \left\{ \begin{array}{l} b_1 \leftarrow B_1 \\ \vdots \\ b_k \leftarrow B_k \end{array} \right\} \right),$$

for some $a \leftarrow b \in K$, $b \leftarrow b_1, \ldots, b_k \in P$, and $b_1 \leftarrow B_1, \ldots, b_k \leftarrow B_k \in R$, in which case we have $r = a \leftarrow B_1 \cup \ldots \cup B_k$. A simple computation shows

$$\{r\} = (\{a \leftarrow b\} \circ \{b \leftarrow b_1, \ldots, b_k\}) \circ \left\{ \begin{array}{l} b_1 \leftarrow B_1 \\ \vdots \\ b_k \leftarrow B_k \end{array} \right\}.$$

The proof that every rule in $(KP)R$ is in $K(PR)$ is analogous.

We now want to prove (15). For any proper Krom rule $r = a \leftarrow b$, we have

$$\begin{aligned} \{a \leftarrow b\} \circ (P \cup R) &= \{a \leftarrow B \mid b \leftarrow B \in P \cup R, B \subseteq A\} \\ &= \{a \leftarrow B \mid b \leftarrow B \in P\} \cup \{a \leftarrow B \mid b \leftarrow B \in R\} \\ &= (\{a \leftarrow b\} \circ P) \cup (\{a \leftarrow b\} \circ R). \end{aligned}$$

Hence, we have

$$K(P \cup R) \overset{(6)}{=} f(K) \cup p(K)(P \cup R)$$

$$\begin{aligned}
&\overset{(3)}{=} f(K) \cup \bigcup_{r \in p(K)} \{r\}(P \cup R) \\
&= f(K) \cup \bigcup_{r \in p(K)} (\{r\}P \cup \{r\}R) \\
&= f(K) \cup \bigcup_{r \in p(K)} \{r\}P \cup \bigcup_{r \in p(K)} \{r\}R \\
&= f(K) \cup p(K)P \cup p(K)R \\
&= (f(K) \cup p(K)P) \cup (f(K) \cup p(K)R) \\
&= KP \cup KR
\end{aligned}$$

where the sixth identity follows from the idempotency of union. This shows that Krom programs distribute from the left which implies that the space of all proper[4] Krom programs forms a semiring (cf. Corollary 9).

4.2.1 Interpretations

Notice that, formally, interpretations are Krom programs containing only rules with empty bodies called facts, which gives interpretations a special compositional meaning.

Fact 13 *Every interpretation I is a left zero, that is, for any program P, we have*

$$IP = I. \tag{16}$$

Consequently, the space of interpretations forms a right ideal.[5]

Corollary 14 *A program P commutes with an interpretation I iff I is a supported model of P, that is,*

$$PI = IP \quad \Leftrightarrow \quad I \in Supp(P).$$

Proof A direct consequence of Fact 13 and the forthcoming Theorem 35 (which is not depending on this result). □

4.2.2 Permutations

With every permutation $\pi : A \to A$, we associate the propositional Krom program

$$\pi = \{\pi(a) \leftarrow a \mid a \in A\}.$$

We adopt here the standard cycle notation for permutations. For instance, we have

$$\pi_{(a\,b)} = \left\{ \begin{array}{l} a \leftarrow b \\ b \leftarrow a \end{array} \right\} \quad \text{and} \quad \pi_{(a\,b\,c)} = \left\{ \begin{array}{l} a \leftarrow b \\ b \leftarrow c \\ c \leftarrow a \end{array} \right\}.$$

The inverse π^{-1} of π translates into the language of programs as

$$\pi^{-1} = \pi^d.$$

[4] If K contains facts, then $K\emptyset = f(K) \neq \emptyset$ violates the last axiom of a semiring (cf. § 2.1).

[5] See Corollary 38.

Interestingly, we can rename the atoms occurring in a program via permutations and composition by

$$\pi \circ P \circ \pi^d = \{\pi(h(r)) \leftarrow \pi(b(r)) \mid r \in P\}.$$

We have the following structural result as a direct instance of the more general result for permutations:

Fact 15 *The space of all permutation programs forms a group.*

4.3 Reducts

Reducing a program to a restricted alphabet is a fundamental operation on programs which we will often use in the rest of the paper, especially in § 5. For example, in Corollary 18 we will see that the computation of composition can be simplified using reducts. This motivates the following definition.

Definition 16 We define the ***left*** and ***right reduct***[6] of a program P, with respect to some interpretation I, respectively by

$${}^I P := \{r \in P \mid I \models h(r)\} \quad \text{and} \quad P^I := \{r \in P \mid I \models b(r)\}.$$

Our first observation is that we can compute the facts of P via the right reduct with respect to the empty set, that is, we have

$$f(P) = P^\emptyset = P\emptyset. \tag{17}$$

On the contrary, computing the left reduct with respect to the empty set yields

$${}^\emptyset P = \emptyset. \tag{18}$$

Moreover, for any interpretations I and J, we have

$${}^J I = I \cap J \quad \text{and} \quad I^J = I. \tag{19}$$

Notice that we obtain the reduction of P to the atoms in I, denoted $P|_I$, by

$$P|_I = {}^I(P^I) = ({}^I P)^I. \tag{20}$$

As the order of computing left and right reducts is irrelevant, in the sequel we will omit the parentheses in (20). Of course, we have

$${}^A P = P^A = {}^A P^A = P.$$

Moreover, we have

$$1^I = {}^I 1 = {}^I 1^I = 1_I = 1|_I \tag{21}$$

$$1^I \circ 1^J = 1^{I \cap J} = 1^J \circ 1^I = 1^I \cap 1^J \tag{22}$$

$$1^I \cup 1^J = 1^{I \cup J}. \tag{23}$$

We now want to relate reducts to composition and union.

[6] Our notion of right reduct coincides with the FLP reduct [33] well-known in answer set programming.

Proposition 17 *For any program P and interpretation I, we have*

$$ {}^{I}P = 1^{I} \circ P \quad \textit{and} \quad P^{I} = P \circ 1^{I}. \tag{24} $$

Moreover, we have

$$ P|_{I} = 1^{I} \circ P \circ 1^{I}. \tag{25} $$

Consequently, for any program R, we have

$$ {}^{I}(P \cup R) = {}^{I}P \cup {}^{I}R \quad \textit{and} \quad {}^{I}(PR) = {}^{I}PR \tag{26} $$

$$ (P \cup R)^{I} = P^{I} \cup R^{I} \quad \textit{and} \quad (PR)^{I} = PR^{I}. \tag{27} $$

Proof We compute

$$ \begin{aligned} 1^{I} \circ P &= \{h(r) \leftarrow b(s) \mid r \in 1^{I}, s \in P, h(s) = b(r)\} \\ &= \{h(r) \leftarrow b(s) \mid r \in 1, s \in P, h(s) = b(r), h(r) = b(r) \in I\} \\ &= \{h(s) \leftarrow b(s) \mid s \in P, h(s) \in I\} \\ &= \{s \in P \mid I \models h(s)\} \\ &= {}^{I}P. \end{aligned} $$

Similarly, we compute

$$ P \circ 1^{I} = \{h(r) \leftarrow b(S) \mid r \in P, S \subseteq_{sz(r)} 1^{I}, h(S) = b(r)\} \tag{28} $$

$$ = \left\{ h(r) \leftarrow b(S) \;\middle|\; \begin{array}{l} r \in P, S \subseteq_{sz(r)} 1, \\ h(S) = b(r), \\ h(S) = b(S) \subseteq I \end{array} \right\}. \tag{29} $$

By definition of 1, we have $b(S) = h(S)$ and therefore $b(S) = b(r)$ and $b(r) \subseteq I$. Hence, (28) is equivalent to

$$ \{h(r) \leftarrow b(r) \mid r \in P : I \models b(r)\} = P^{I}. $$

Finally, we have

$$ {}^{I}(P \cup R) \overset{(24)}{=} 1^{I}(P \cup R) \overset{(15)}{=} 1^{I}P \cup 1^{I}R \overset{(24)}{=} {}^{I}P \cup {}^{I}R $$

and

$$ {}^{I}(PR) \overset{(24)}{=} (1^{I})(PR) \overset{(14)}{=} ((1^{I})P)R(24) = {}^{I}PR $$

with the remaining identities in (27) holding by analogous computations. □

Consequently, we can simplify the computation of composition as follows.

Corollary 18 *For any programs P and R, we have*

$$ P \circ R = P^{h(R)} \circ {}^{b(P)}R. \tag{30} $$

4.4 Proper programs

By equation (17), we can extract the facts of a program P by computing the right reduct with respect to the empty set, $P^{\emptyset}$, and by composing P with the empty set, $P\emptyset$. Unfortunately, there is no analogous characterization of the proper rules in terms of composition. We have therefore introduced a unary operator p for that purpose and we now want to state some basic properties of that operator as it is used to characterize idempotent programs (Proposition 20). The proper rules operator satisfies the following identities, for any propositional logic programs P and R, and interpretation I:

$$p \circ p = p \tag{31}$$
$$p \circ f = f \circ p = \emptyset \tag{32}$$
$$p(1) = 1 \tag{33}$$
$$p(I) = \emptyset \tag{34}$$
$$p(P)\emptyset \overset{(17)}{=} f(p(P)) \overset{(32)}{=} \emptyset \tag{35}$$
$$p(P \cup R) = p(P) \cup p(R). \tag{36}$$

Of course, we have $p(P) = P$ iff P contains no facts, that is, iff $P^{\emptyset} = \emptyset$. The last identity says that the proper rules operator is compatible with union; however, the following counter-example shows that it is *not* compatible with sequential composition:

$$p\left(\{a \leftarrow b, c\} \circ \left\{ \begin{matrix} b \leftarrow b \\ c \end{matrix} \right\}\right) = \{a \leftarrow b\}$$

whereas

$$p(\{a \leftarrow b, c\}) \circ p\left(\left\{ \begin{matrix} b \leftarrow b \\ c \end{matrix} \right\}\right) = \emptyset.$$

This shows that the proper rule operator fails to be a homomorphism with respect to composition.

Proposition 19 *The space of all proper propositional logic programs forms a submagma of the space of all propositional logic programs with the zero given by the empty set.*

Proof The space of proper programs is closed under composition. It remains to show that the empty set is a zero, but this follows from the fact that it is a left zero by (16) and from the observation that for any proper program P, we have $P\emptyset \overset{(17)}{=} f(P) = \emptyset$. □

4.5 Idempotent programs

Recall that P is called ***idempotent*** if $P \circ P = P$. Idempotent programs have the nice property that their least models are trivial to compute as they coincide with the facts of the program. Characterizing idempotent programs on an algebraic level is therefore interesting, which is done in the next result in terms of their facts and proper rules.

Proposition 20 *A program P is idempotent iff*

$$p(P)f(P) \subseteq f(P) \quad \textit{and} \quad p(P^2) = p(p(P)P). \tag{37}$$

Proof The program P is idempotent iff we have (i) $f(P^2) = f(P)$ and (ii) $p(P^2) = p(P)$. For the first condition in (37), we compute

$$\begin{aligned} f(P^2) &\overset{(17)}{=} P^2\emptyset \\ &= P(P\emptyset) \\ &= P \circ f(P) \\ &\overset{(5)}{=} (f(P) \cup p(P))f(P) \\ &\overset{(4)}{=} f(P)f(P) \cup p(P)f(P) \\ &\overset{(16)}{=} f(P) \cup p(P)f(P). \end{aligned}$$

This yields

$$f(P^2) = f(P) \quad \Leftrightarrow \quad p(P)f(P) \subseteq f(P).$$

For the second condition in (37), we compute

$$\begin{aligned} p(P^2) &= p((f(P) \cup p(P))P) \\ &\overset{(4)}{=} p(f(P)P \cup p(P)P) \\ &\overset{(16)}{=} p(f(P) \cup p(P)P) \\ &\overset{(36}{=} p(f(P)) \cup p(p(P)P) \\ &\overset{(32)}{=} p(p(P)P). \end{aligned}$$

□

Fact 21 *Every interpretation is idempotent.*

Proof A direct consequence of (16).

5 Decomposition

It is often useful to know how a program is assembled from its simpler parts. In this section, we therefore study sequential decompositions of acyclic (§ 5.3) and arbitrary programs (§ 5.4). Before we do that we first look in § 5.1 at some ways how to algebraically transform programs via composition.

5.1 Adding and removing body atoms

Notice that we can manipulate rule bodies via composition on the right. For example, we have

$$\{a \leftarrow b, c\} \circ \left\{ \begin{array}{l} b \leftarrow b \\ c \end{array} \right\} = \{a \leftarrow b\}.$$

The general construction here is that we add a tautological rule $b \leftarrow b$ for every body atom b of P which we want to preserve, and we add a fact c in case we want to remove c from the rule bodies in P. We therefore define, for an interpretation I, the minimalist program

$$I^{\ominus} = 1^{A-I} \cup I.$$

Notice that $.^{\ominus}$ is computed with respect to some fixed alphabet A. For instance, we have

$$A^{\ominus} = A \quad \text{and} \quad \emptyset^{\ominus} = 1.$$

The first equation yields another explanation for (8), that is, we can compute the heads in P by removing all body atoms of P via

$$h(P) = PA^{\ominus} = PA.$$

Interestingly enough, we have

$$I^{\ominus}I = (1^{A-I} \cup I)I \overset{(4)}{=} 1^{A-I}I \cup I^2 \overset{(16),(19),(24)}{=} ((A - I) \cap I) \cup I = I$$

and

$$I^{\ominus}P = {}^{A-I}P \cup I.$$

Moreover, in the example above, we have

$$\{c\}^{\ominus} = \left\{ \begin{array}{l} a \leftarrow a \\ b \leftarrow b \\ c \end{array} \right\} \quad \text{and} \quad \{a \leftarrow b, c\} \circ \{c\}^{\ominus} = \{a \leftarrow b\}$$

as desired. Notice also that the facts of a program are, of course, not affected by composition on the right, that is, we cannot expect to remove facts via composition on the right (cf. (7)).

We have the following general result:

Proposition 22 *For any P and interpretation I, we have*

$$PI^{\ominus} = \{h(r) \leftarrow (b(r) - I) \mid r \in P\}.$$

In analogy to the above construction, we can add body atoms via composition on the right. For example, we have

$$\{a \leftarrow b\} \circ \{b \leftarrow b, c\} = \{a \leftarrow b, c\}.$$

Here, the general construction is as follows. For an interpretation I, define the minimalist program

$$I^{\oplus} = \{a \leftarrow (\{a\} \cup I) \mid a \in A\}.$$

For instance, we have

$$A^{\oplus} = \{a \leftarrow A \mid a \in A\} \quad \text{and} \quad \emptyset^{\oplus} = 1.$$

Interestingly enough, we have

$$I^{\oplus}I^{\ominus} = I^{\ominus} \quad \text{and} \quad I^{\oplus}I = I.$$

Moreover, in the example above, we have

$$\{c\}^{\oplus} = \left\{ \begin{array}{l} a \leftarrow a, c \\ b \leftarrow b, c \\ c \leftarrow c \end{array} \right\} \quad \text{and} \quad \{a \leftarrow b\} \circ \{c\}^{\oplus} = \{a \leftarrow b, c\}$$

as desired. As composition on the right does not affect the facts of a program, we cannot expect to append body atoms to facts via composition on the right. However, we can add arbitrary atoms to *all* rule bodies simultaneously and in analogy to Proposition 22, we have the following general result:

Proposition 23 *For any program P and interpretation I, we have*

$$PI^{\oplus} = f(P) \cup \{h(r) \leftarrow (b(r) \cup I) \mid r \in p(P)\}.$$

We now want to illustrate the interplay between the above concepts with an example.

Example 24 Consider the propositional logic programs

$$P = \left\{ \begin{array}{l} c \\ a \leftarrow b, c \\ b \leftarrow a, c \end{array} \right\} \quad \text{and} \quad \pi_{(a\,b)} = \left\{ \begin{array}{l} a \leftarrow b \\ b \leftarrow a \end{array} \right\}.$$

Roughly, we obtain P from $\pi_{(a\,b)}$ by adding the fact c to $\pi_{(a\,b)}$ and to each body rule in $\pi_{(a\,b)}$. Conversely, we obtain $\pi_{(a\,b)}$ from P by removing the fact c from P and by removing the body atom c from each rule in P. This can be formalized as[7]

$$P = \{c\}^* \pi_{(a\,b)} \{c\}^{\oplus} \quad \text{and} \quad \pi_{(a\,b)} = 1^{\{a,b\}} P \{c\}^{\ominus}.$$

Remark 25 It is important to emphasize that we cannot add *arbitrary* rules to a program via composition. For example, suppose we want to add the rule $r = a \leftarrow b$ to the empty program $\emptyset$. By (16) we have $\emptyset P = \emptyset$, for all P, which means that the only reasonable thing we can do is to compute $P\emptyset$—but $P\emptyset \overset{(17)}{=} f(P) \neq \{r\}$ shows that we cannot add r to the empty program via composition on the left. The intuitive reason is that in order to construct the rule r via composition, we need to be able to add body atoms to facts, which is easily seen to be impossible.

5.2 Closures

The following construction will often be useful when studying decompositions:

Definition 26 Define the ***closure*** of P with respect to some alphabet A by

$$c_A(P) := 1_A \cup P. \tag{38}$$

Roughly, we obtain the closure of a program by adding all possible tautological rules of the form $a \leftarrow a$, for $a \in A$. Of course, the closure operator preserves semantic equivalence, that is,

$$P \equiv c_A(P), \quad \text{for all alphabets } A.$$

Lemma 27 *For any programs P and R, if P does not depend on R, we have*

$$P(Q \cup R) = PQ. \tag{39}$$

Proof We compute

$$P(Q \cup R) \overset{(30)}{=} P^{h(Q \cup R)} \left({}^{b(P)}(Q \cup R) \right) \overset{(26)}{=} P^{h(Q \cup R)} \left({}^{b(P)}Q \cup {}^{b(P)}R \right). \tag{40}$$

Since P does not depend on R, we have $b(P) \cap h(R) = \emptyset$ and, hence,

$${}^{b(P)}R = \{r \in R \mid b(P) \models h(r)\} = \{r \in R \mid h(r) \in b(P)\} = \emptyset.$$

[7] Here, we have $\{c\}^* = 1 \cup \{c\}$ and $\{c\}^* \pi_{(a\,b)} = \pi_{(a\,b)} \cup \{c\}$ by the forthcoming equation (57).

Moreover, we have

$$P^{h(Q\cup R)} = P^{h(Q)\cup h(R)} = \{r \in P \mid b(r) \subseteq h(Q) \cup h(R)\} = \{r \in P \mid b(r) \subseteq h(Q)\} = P^{h(Q)}.$$

Hence, (40) is equivalent to

$$\left(P^{h(Q)}\right)\left({}^{b(P)}Q\right) \overset{(30)}{=} PQ.$$

□

Lemma 28 *For any programs P and R, in case P does not depend on R, we have*

$$P \cup R = c_{h(R)}(P)c_{b(P)}(R) \quad \text{and} \quad c_A(P \cup R) = c_A(P)c_A(R). \tag{41}$$

Moreover, for any alphabets A and B, we have

$$c_A(c_B(P)) = c_{A\cup B}(P).$$

Proof We compute

$$\begin{aligned}
c_{h(R)}(P)c_{b(P)}(R) &= (1_{h(R)} \cup P)(1_{b(P)} \cup R)\\
&\overset{(4)}{=} 1_{h(R)}(1_{b(P)} \cup R) \cup P(1_{b(P)} \cup R)\\
&\overset{(15)}{=} 1_{h(R)}1_{b(P)} \cup 1_{h(R)}R \cup P(1_{b(P)} \cup R)\\
&\overset{(22),(24)}{=} 1_{h(R)\cap b(P)} \cup {}^{h(R)}R \cup P(1_{b(P)} \cup R)\\
&= R \cup P(1_{b(P)} \cup R)\\
&\overset{(39)}{=} R \cup P1_{b(P)}\\
&\overset{(24)}{=} R \cup P^{b(P)}\\
&= R \cup P
\end{aligned}$$

where the fourth equality follows from $1_{h(R)\cap b(P)} = \emptyset$ and ${}^{h(R)}R = R$, the fifth equality holds since P does not depend on R (Lemma 27), and the last equality follows from $P^{b(P)} = P$.

For the second equality, we compute

$$c_A(P)c_A(R) = (1 \cup P)(1 \cup R) \overset{(4)}{=} 1 \cup R \cup P(1 \cup R) \overset{(39)}{=} 1 \cup R \cup P = c_A(P \cup R)$$

where the third identity follows from the fact that P does not depend on R (Lemma 27).

Lastly, we compute

$$c_A(c_B(P)) = 1_A \cup 1_B \cup B \overset{(23)}{=} 1_{A\cup B} \cup P = c_{A\cup B}(P).$$

5.3 Acyclic programs

Recall that a program is called ***acyclic*** if it has a level mapping so that the head of each rule is strictly greater than each of its body atom with respect to that mapping. Acyclic programs appear frequently in practice [10], which is the motivation for studying their decompositions in this section. The main result of this section shows that acyclic programs can be decomposed into single-rule programs (Theorem 31).

Example 29 We define a family of acyclic programs over A, which we call ***elevator programs***, as follows: given a sequence $(a_1, \ldots, a_n) \in A^n$, $1 \leq n \leq |A|$, of distinct atoms, let $E_{(a_1,\ldots,a_n)}$ be the acyclic program

$$E_{(a_1,\ldots,a_n)} = \{a_1\} \cup \{a_i \leftarrow a_{i-1} \mid 2 \leq i \leq n\}.$$

So, for instance, $E_{(a,b,c)}$ is the program

$$E_{(a,b,c)} = \left\{ \begin{array}{l} a \\ b \leftarrow a \\ c \leftarrow b \end{array} \right\}.$$

The mapping ℓ given by $\ell(a) = 1$, $\ell(b) = 2$, and $\ell(c) = 3$ is a level mapping for $E_{(a,b,c)}$ and we can decompose $E_{(a,b,c)}$ into a product of single-rule programs by

$$\begin{aligned} E_{(a,b,c)} &= (1^{\{b,c\}} \cup \{a\})(1^{\{c\}} \cup \{b \leftarrow a\})(1^{\{a\}} \cup \{c \leftarrow b\}) \\ &= c_{\{b,c\}}(\{a\})c_{\{c\}}(\{b \leftarrow a\})c_{\{c\}}(\{c \leftarrow b\}). \end{aligned}$$

We now want to generalize the reasoning pattern of Example 29 to arbitrary acyclic programs. For this, we will need the following lemma.

Lemma 30 *For any programs P and R and alphabet B, if P and 1_B do not depend on R, we have*

$$c_B(c_{h(R)}(P)c_{b(P)}(R)) = c_{B\cup h(R)}(P)c_{B\cup b(P)}(R). \tag{42}$$

Proof Since P does not depend on R, as a consequence of Lemma 28 we have

$$c_{h(R)}(P)c_{b(P)} = P \cup R.$$

Hence,

$$\begin{aligned} &c_B(c_{h(R)}(P)c_{b(P)}(R)) \\ &\quad = c_B(P \cup R) \\ &\quad = 1_B \cup P \cup R \\ &\quad \overset{(41)}{=} c_{h(R)}(1_B \cup P)c_{b(1_B\cup P)}(R) \qquad (h(R) \cap B = \emptyset) \\ &\quad = c_{B\cup h(R)}(P)c_{B\cup b(P)}(R). \end{aligned}$$

□

We further will need the following auxiliary construction. Define, for any linearly ordered rules $r_1 < \ldots < r_n$, $n \geq 2$,

$$bh_i(\{r_1 < \ldots < r_n\}) := b(\{r_1, \ldots, r_{i-1}\}) \cup h(\{r_{i+1}, \ldots, r_n\}).$$

We are now ready to prove the following decomposition result for acyclic programs.

Theorem 31 *We can sequentially decompose any acyclic program $P = \{r_1 <_\ell \ldots <_\ell r_n\}$, $n \geq 2$, linearly ordered by a level mapping ℓ, into single-rule programs as*

$$P = \prod_{i=1}^{n} c_{bh_i(P)}(\{r_i\}).$$

This decomposition is unique up to reordering of rules within a single level.[8]

[8] See the construction of the total ordering $<_\ell$ in § 2.2.1.

Proof The proof is by induction on the number n of rules in P. For the induction hypothesis $n = 2$ and $P = \{r_1 <_\ell r_2\}$, we proceed as follows. First, we have

$$c_{bh_1(P)}(\{r_1\})c_{bh_2(P)}(\{r_2\}) = (1_{h(\{r_2\})} \cup \{r_1\})(1_{b(\{r_1\})} \cup \{r_2\}) \tag{43}$$
$$= 1_{h(\{r_2\})}(1_{b(\{r_1\})} \cup \{r_2\}) \cup \{r_1\}(1_{b(\{r_1\})} \cup \{r_2\}). \tag{44}$$

Since Krom programs distribute from the left (Theorem 12), we can simplify (43), by applying (23) and (24), into

$$1_{h(\{r_2\})\cap b(\{r_1\})} \cup {}^{h(\{r_2\})}\{r_2\} \cup \{r_1\}^{b(\{r_1\})} = 1_{h(\{r_2\})\cap b(\{r_1\})} \cup \{r_1, r_2\}. \tag{45}$$

Now, since r_1 does not depend on r_2, that is, $h(r_2) \cap b(r_1) = \emptyset$, the first term in (45) equals $1_\emptyset = \emptyset$ which implies that (45) is equivalent to P as desired.

For the induction step $P = \{r_1 <_\ell \ldots <_\ell r_{n+1}\}$, we proceed as follows. First, by definition of bh_i, we have

$$\prod_{i=1}^{n+1}(c_{bh_i(P)}(\{r_i\})) = \prod_{i=1}^{n+1}(1_{bh_i(P)} \cup \{r_i\}) \tag{46}$$
$$= \left[\prod_{i=1}^{n}(1_{bh_i(\{r_1,\ldots,r_n\})} \cup 1_{h(r_{n+1})} \cup \{r_i\})\right](1_{b(\{r_1,\ldots,r_n\})} \cup \{r_{n+1}\}). \tag{47}$$

Second, by idempotency of union we can extract the term

$$1_{h(r_{n+1})} = 1_{h(r_{n+1})} \ldots 1_{h(r_{n+1})} \quad (n \text{ times})$$

occurring in (47) thus obtaining

$$\left[1_{h(r_{n+1})} \cup \prod_{i=1}^{n}(1_{bh_i(\{r_1,\ldots,r_n\})} \cup 1_{h(r_{n+1})} \cup \{r_i\})\right](1_{b(\{r_1,\ldots,r_n\})} \cup \{r_{n+1}\}). \tag{48}$$

Now, since r_i and $1_{bh_i(\{r_1,\ldots,r_n\})}$ do not depend on r_{n+1}, we can simplify (48) further to

$$\left[1_{h(r_{n+1})} \cup \prod_{i=1}^{n}(1_{bh_i(\{r_1,\ldots,r_n\})} \cup \{r_i\})\right](1_{b(\{r_1,\ldots,r_n\})} \cup \{r_{n+1}\}). \tag{49}$$

By applying the induction hypothesis to (49), we obtain

$$(1_{h(r_{n+1})} \cup \{r_1, \ldots, r_n\})(1_{b(\{r_1,\ldots,r_n\})} \cup \{r_{n+1}\})$$

which, by right-distributivity of composition, is equal to

$$1_{h(r_{n+1})}(1_{b(\{r_1,\ldots,r_n\})} \cup \{r_{n+1}\}) \cup \{r_1, \ldots, r_n\}(1_{b(\{r_1,\ldots,r_n\})} \cup \{r_{n+1}\}). \tag{50}$$

Now again since r_i does not depend on r_{n+1}, for all $1 \leq i \leq n$, as a consequence of (15), (24), (23), and (39), the term in (50) equals

$$1_{h(r_{n+1})\cap b(\{r_1,\ldots,r_n\})} \cup {}^{h(r_{n+1})}\{r_{n+1}\} \cup \{r_1, \ldots, r_n\}^{b(\{r_1,\ldots,r_n\})}. \tag{51}$$

Finally, since $1_{h(r_{n+1})\cap b(\{r_1,\ldots,r_n\})} = 1_\emptyset = \emptyset$, (51) equals $\{r_1, \ldots, r_{n+1}\}$, which proves our theorem. □

5.4 General decompositions

In the last subsection, we studied the decomposition of acyclic programs and showed that they are assembled of single-rule programs. In this subsection, we study decompositions of arbitrary programs (Theorem 32) which turns out to be much more difficult given that we now have to deal with circularities among the rules (Problem 34).

We first wish to generalize Lemma 28 to arbitrary propositional logic programs. For this, we define, for every interpretation I and disjoint copy $I' = \{a' \mid a \in I\}$ of I, the minimalist program (recall that A is the underlying propositional alphabet)

$$\begin{aligned}[I \leftarrow I'] &:= c_{A-I}(\{a \leftarrow a' \mid a \in I\}) \\ &= \{a \leftarrow a' \mid a \in I\} \cup \{a \leftarrow a \mid a \in A - I\}.\end{aligned}$$

We have

$$[A \leftarrow A'] = \{a \leftarrow a' \mid a \in A\}$$

and therefore

$$P[A \leftarrow A'] = \{h(r) \leftarrow b(r)' \mid r \in P\},$$

where $b(r)' = \{b' \mid b \in b(r)\}$, and

$$[A' \leftarrow A]P = \{h(r)' \leftarrow b(r) \mid r \in P\}.$$

Moreover, we have

$$[A \leftarrow A'][A' \leftarrow A] = 1_A. \tag{52}$$

We are now ready to prove the main result of this subsection which shows that we can represent the union of programs via composition and closures:

Theorem 32 *For any programs P and R, we have*

$$P \cup R = c_{h(R)}(P[A \leftarrow A'])c_{b(P[A \leftarrow A'])}(R)c_A([A' \leftarrow A]).$$

Proof Since $P[A \leftarrow A']$ does not depend on R, we have, as a consequence of Lemma 28,

$$P[A \leftarrow A'] \cup R = c_{h(R)}(P[A \leftarrow A'])c_{b(P[A \leftarrow A'])}(R).$$

Finally, we have

$$\begin{aligned}(P[A \leftarrow A'] \cup R)c_A([A' \leftarrow A]) &\overset{(4)}{=} P[A \leftarrow A']c_A([A' \leftarrow A]) \cup R\, c_A([A' \leftarrow A]) \\ &= P[A \leftarrow A'](1_A \cup [A' \leftarrow A]) \cup R(1_A \cup [A' \leftarrow A]) \\ &= P[A \leftarrow A'][A' \leftarrow A] \cup R \\ &\overset{(52)}{=} P \cup R\end{aligned}$$

where the third identity follows from the fact that $P[A \leftarrow A']$ does not depend on 1_A and R does not depend on $[A' \leftarrow A]$ (apply Lemma 27). □

Example 33 Let $A = \{a, b, c\}$ and consider the program $P = R \cup \pi_{(b\,c)}$ where

$$R = \left\{ \begin{array}{l} a \leftarrow b, c \\ a \leftarrow a, b \\ b \leftarrow a \end{array} \right\} \quad \text{and} \quad \pi_{(b\,c)} = \left\{ \begin{array}{l} b \leftarrow c \\ c \leftarrow b \end{array} \right\}.$$

We wish to decompose P into a product of R and $\pi_{(b\,c)}$ according to Theorem 32. For this, we first have to replace the body of R with a distinct copy of its atoms, that is, we compute

$$R[A \leftarrow A'] = \left\{ \begin{array}{l} a \leftarrow b', c' \\ a \leftarrow a', b' \\ b \leftarrow a' \end{array} \right\}.$$

Notice that $R[A \leftarrow A']$ no longer depends on $\pi_{(b\,c)}$! Next, we compute the composition of

$$c_{h(\pi_{(b\,c)})}(R[A \leftarrow A']) = \left\{ \begin{array}{l} a \leftarrow b', c' \\ a \leftarrow a', b' \\ b \leftarrow a' \\ b \leftarrow b \\ c \leftarrow c \end{array} \right\}$$

and

$$c_{b(R[A \leftarrow A'])}(\pi_{(b\,c)}) = \left\{ \begin{array}{l} b \leftarrow c \\ c \leftarrow b \\ a' \leftarrow a' \\ b' \leftarrow b' \\ c' \leftarrow c' \end{array} \right\}$$

by

$$R' = c_{h(\pi_{(b\,c)})}(R[A \leftarrow A'])c_{b(R[A \leftarrow A'])}(\pi_{(b\,c)}) = \left\{ \begin{array}{l} a \leftarrow b', c' \\ a \leftarrow a', b' \\ b \leftarrow a' \\ b \leftarrow c \\ c \leftarrow b \end{array} \right\}.$$

Finally, we need to replace the atoms from A' in the body of R' by atoms from A, that is, we compute

$$R'c_A([A' \leftarrow A]) = \left\{ \begin{array}{l} a \leftarrow b', c' \\ a \leftarrow a', b' \\ b \leftarrow a' \\ b \leftarrow c \\ c \leftarrow b \end{array} \right\} \circ \left\{ \begin{array}{l} a \leftarrow a \\ b \leftarrow b \\ c \leftarrow c \\ a' \leftarrow a \\ b' \leftarrow b \\ c' \leftarrow c \end{array} \right\} = P$$

as expected.

Problem 34 Unfortunately, at the moment we have no such nice decomposition result for arbitrary programs as for acyclic programs (Theorem 31), not relying on auxiliary atoms in A', which remains as future work (cf. § 8).

6 Algebraic semantics

Recall that the fixed point semantics of a propositional logic program is defined in terms of least fixed point computations of its associated van Emden-Kowalski immediate consequence operator (Proposition 3). That is, the semantics is defined *operationally* outside the syntactic space of the programs. In this section, we reformulate the fixed point semantics of programs

in terms of sequential composition by first showing that we can express the van Emden-Kowalski operators via composition (Theorem 35), and then showing how programs (not operators) can be iterated bottom-up to obtain their least models (Theorem 40). This bridges the conceptual gap between the syntax and semantics of a program.

6.1 The van Emden-Kowalski operator

The next results show that we can simulate the van Emden-Kowalski operators on a *syntactic* level without any explicit reference to operators.

Theorem 35 *For any program P and interpretation I, we have*

$$T_P(I) = PI. \tag{53}$$

Moreover, we have for any program R,

$$T_P \cup T_R = T_{P \cup R} \quad \text{and} \quad T_P \circ T_R = T_{PR} \quad \text{and} \quad T_\emptyset = \emptyset \quad \text{and} \quad T_1 = Id_{2^A}. \tag{54}$$

Proof All of the equations follow immediately from the definition of composition. To illustrate some of the already derived results, we compute

$$T_P(I) = h(P^I) \overset{(8)}{=} P^I A \overset{(24)}{=} (P(1^I))A = P((1^I)A) \overset{(24)}{=} P({}^I A) \overset{(19)}{=} P(I \cap A) = PI$$

where the last equality follows from I being a subset of A. □

Corollary 36 *Sequential composition is associative modulo subsumption equivalence, that is,*

$$P(QR) \equiv_{ss} (PQ)R \quad \text{for any programs } P, Q, R.$$

This implies

$$(PR)I = P(RI), \quad \text{for any programs } P, R \text{ and interpretation } I. \tag{55}$$

Proof Associativity modulo subsumption equivalence is an immediate consequence of Theorem 35 which shows

$$T_{P(QR)} = T_{(PQ)R} \quad \Rightarrow \quad P(QR) \equiv_{ss} (PQ)R.$$

The identity is shown via (54) by

$$(PR)I = T_{PR}(I) = T_P(T_R(I)) = P(RI).$$

□

As a direct consequence of Proposition 2 and Theorem 35, we have the following algebraic characterization of (supported) models:

Corollary 37 *An interpretation I is a model of P iff $PI \subseteq I$, and I is a supported model of P iff $PI = I$.*

Corollary 38 *The space of all interpretations forms an ideal.*

Proof By Fact 13, we know that the space of interpretations forms a right ideal and Theorem 35 implies that it is a left ideal—hence, it forms an ideal. □

6.2 Least models

We interpret propositional logic programs according to their least model semantics and since least models can be constructively computed by bottom-up iterations of the associated van Emden-Kowalski operators (Proposition 3), and since these operators can be represented in terms of composition (Theorem 35), we can finally reformulate the fixed point semantics of programs algebraically in terms of composition (Theorem 40).

We make the convention that for any $n \geq 3$,

$$P^n = (((PP)P)\dots P)P \quad (n \text{ times}).$$

Definition 39 Define the unary ***Kleene star*** and ***plus operations*** by

$$P^* = \bigcup_{n \geq 0} P^n \quad \text{and} \quad P^+ = P^* P. \tag{56}$$

Moreover, define the ***omega operation*** by

$$P^\omega = P^+ \emptyset \overset{(17)}{=} f(P^+).$$

Notice that the unions in (56) are finite since P is finite. For instance, for any interpretation I, we have as a consequence of Fact 21,

$$I^* = 1 \cup I \quad \text{and} \quad I^+ = I \quad \text{and} \quad I^\omega = I. \tag{57}$$

Interestingly enough, we can add the atoms in I to P via

$$P \cup I \overset{(16)}{=} P \cup IP \overset{(4)}{=} (1 \cup I)P \overset{(57)}{=} I^* P. \tag{58}$$

Hence, as a consequence of (5) and (58), we can decompose P as

$$P = f(P)^* p(P),$$

which, roughly, says that we can sequentially separate the facts from the proper rules in P.

We are now ready to characterize the least model of a program via composition as follows.

Theorem 40 *For any program P, we have*

$$LM(P) = P^\omega.$$

Proof A direct consequence of Proposition 3 and Theorem 35. □

Corollary 41 *Two programs P and R are equivalent iff $P^\omega = R^\omega$.*

To summarize, we have thus shown that we can use the syntactic sequential composition operation to capture the least model or fixed point semantics of a program algebraically *within* the space of programs giving rise to the algebraic characterization of the semantics of a program in Theorem 40 and the semantic equivalence of programs in Corollary 41.

7 Related work

O'Keefe [51] is probably the first to study the composition of logic programs. He first defines the denotation of P by[9] $[\![P]\!] = (T_P \cup Id)^*$ and then defines the semantic composition of P and R so that $[\![P \circ R]\!] = [\![P]\!] \circ [\![R]\!]$ (cf. [19, p. 450]). That is, he defines the composition of two programs on a *semantic* level in terms of the compositions of their immediate consequence operators which is straightforward given that we already know how to compose functions; in contrast, in our approach of this paper we define the sequential composition of two programs explicitly on a *syntactic* level which allows us to distinguish algebraically between subsumption equivalent programs (at the cost of losing associativity; see Example 10 and Corollary 36).

It is important to emphasize that since we can represent the immediate consequence operator in terms of sequential composition (Theorem 35), O'Keefe's definition can be resembled in terms of sequential composition as well via[10]

$$\begin{aligned}
[\![P \circ R]\!](I) &= ([\![P]\!] \circ [\![R]\!])(I) \\
&= ((T_P \cup Id)^* \circ (T_R \cup Id)^*)(I) \\
&= (T_P \cup Id)^*((T_R \cup Id)^*(I)) \\
&\overset{(53)}{=} (P \cup 1)^* \circ ((P \cup 1)^* \circ I) \\
&\overset{(38)}{=} c(P)^* \circ (c(R)^* \circ I) \quad \text{for all interpretations } I.
\end{aligned}$$

Bugliesi et al. [19] study *modular logic programming* by defining algebraic operations on programs similar to [51]. More formally, O'Keefe's composition is captured in [19] framework as $[\![P \circ R]\!] = [\![(P \cup R^\omega)^\omega]\!]$, where $[\![P^\omega]\!] = T_P^\infty$ in our notation (cf. [19, p. 450]). This shows that we can represent [19] algebraic operations in terms of sequential composition as well. The work of [15] is similar in spirit to the aforementioned works. Formally, [15] first define the admissible Herbrand model of P denoted (again) by $[\![P]\!]$ and then define a *semantic* composition operation $\bullet$ which allows one to compute the semantics of the union of two programs in the sense that $[\![P \cup R]\!] = [\![P]\!] \bullet [\![R]\!]$ (cf. [15, Proposition 3.4]).

Dong and Ginsburg [27, 28] study the composition and decomposition of datalog program *mappings*—where datalog programs are function-free logic programs used mainly in database theory (cf. [20, 21])—for query optimization in the context of databases. Plambeck [54, 55] employs tools from semigroup theory for query optimization. His work is similar in spirit to the work of [39] who map Horn rules to an relational algebra operator, where the set of all such operators forms a semiring, and answer queries via solving linear equations. The authors then study decompositions of relational algebra operators for optimization purposes. To the best of our knowledge, these are the only works utilizing semigroups and semirings in logic programming.[11]

Dix et al. [26] consider the following *principle of partial evaluation* as a semantic preserving transformation of a given program P. Let $r = a_0 \leftarrow a_1, \ldots, a_k \in P$ and choose some

[9] For an operator T, we define T^* in the same way as P^* in § 6.2 as $T^* = Id \cup T \cup T^2 \cup \ldots$; notice that [51] uses the notation T^ω instead of T^* which we refuse to use here to remain consistent with the use of the symbols in § 6.2.

[10] Notice that the identity function Id corresponds to the unit program 1 since $Id(I) = 1 \circ I$ holds for all interpretations I.

[11] There are semiring-based works on *weighted* logic programs, where semirings are used to add probabilities and other quantitative information to rules and programs e.g. [12].

fixed body atom a among $a_1, \ldots, a_k$; without loss of generality, we can assume $a = a_1$. If all the rules in P with a_1 in the head are given by

$$a_1 \leftarrow B_1 \quad \ldots \quad a_1 \leftarrow B_n, \tag{59}$$

then we can replace r by the n rules

$$a_0 \leftarrow B_1 \cup \{a_2, \ldots, a_k\} \quad \ldots \quad a_0 \leftarrow B_n \cup \{a_2, \ldots, a_k\}.$$

There is some similarity between this transformation and the computation of $P^2 = P \circ P$. Formally, if we add to P the tautological rules

$$a_2 \leftarrow a_2 \quad \ldots \quad a_k \leftarrow a_k,$$

then P^2 contains all the rules in (59); however, P^2 may contain many more rules with a_1 in its head depending on the rules in P with $a_2, \ldots, a_k$ as head atoms. So the main difference between the above principle and the sequential composition operation of this paper is that the former operates on *single* programs and on *single* body atoms whereas the latter is defined as a binary operation between *two* programs and operates on *all* body atoms at once.

Antić [4] uses the sequential composition and decomposition of (first-order) logic programs for logic program synthesis via analogical proportions [2], and [5] uses it for defining a qualitative notion of syntactic program similarity which allows one to answer queries across different domains. Antić [7] provides algebraic characterizations of least model and uniform equivalence of propositional Krom logic programs in terms of sequential composition. Antić [3] introduces the index and period of a logic program in terms of the sequential composition as an algebraic measure of its cyclicality. Finally, the cascade product of answer set programs has been defined in [1].

From a purely mathematical point of view, sequential composition of propositional logic programs as introduced in this paper is very similar to composition in multicategories (cf. [42, Section 2.1]).

8 Future work

In the future, we plan to extend the constructions and results of this paper to wider classes of logical formalisms, most importantly to first-order and higher-order logic programs [8, 44], disjunctive programs [29], and non-monotonic logic programs under the stable model or answer set semantics [35] see [11, 14, 30, 43]. The first task is non-trivial as function symbols require the use of most general unifiers in the definition of composition and give rise to infinite algebras, whereas the non-monotonic case is more difficult to handle algebraically due to negation as failure [23] occurring in rule bodies (and heads). In [6], the author studies the non-monotonic case in detail by lifting the concepts and results of this paper from the Horn to the general case containing negation as failure. Some of the results of this paper are subsumed by more general results in [6]. However, lifting the results on decompositions of programs of § 5 to programs with negation turns out to be highly non-trivial and therefore remains an open problem. Even more problematic, disjunctive rules yield non-deterministic behavior which is more difficult to handle algebraically. Nonetheless, we expect interesting results in all of the aforementioned cases to follow.

Another major line of research is to study sequential *decompositions* of programs as was initiated in Problem 34. Specifically, we wish to compute decompositions of arbitrary propositional logic programs (and extensions thereof) into simpler programs in the vein of

Theorem 31, where we expect permutation programs (§ 4.2.2) to play a fundamental role in such decompositions. For this, it will be necessary to resolve the issue of defining the notion of a "prime" or indecomposable program. From a practical point of view, a mathematically satisfactory theory of program decompositions is relevant to modular logic programming and optimization of reasoning.

9 Conclusion

This paper studied the (sequential) composition of propositional logic programs. We showed in our main structural result Theorem (refthm:Horn) that the space of all such programs forms a finite unital magma with respect to composition ordered by set inclusion, which distributes from the right over union. Moreover, we showed that the restricted class of propositional Krom programs is distributive and therefore its proper instance forms an idempotent semiring (Theorem 12). From a logical point of view, we obtained algebraic tools for reasoning about propositional logic programs. Algebraically, we obtained a correspondence between propositional logic programs and finite unital magmas and other algebraic structures, which hopefully enables us in the future to transfer concepts from the algebraic literature to the logical setting. In a broader sense, this paper is a first step towards an algebra of logic programs and we expect interesting concepts and results to follow.

Acknowledgements We would like to thank the reviewers for their thoughtful and constructive comments, and for their helpful suggestions to improve the presentation of the article.

Funding Open access funding provided by TU Wien (TUW).

Data Availability The manuscript has no data associated.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Antić, C.: On cascade products of answer set programs. Theory Pract. Log. Program. **14**(4–5), 711–723 (2014)
2. Antić, C.: Analogical proportions. Ann. Math. Artif. Intell. **90**(6), 595–644 (2022). https://doi.org/10.1007/s10472-022-09798-y
3. Antić, C.: The index and period of a logic program (2023a). https://hal.science/hal-04219244
4. Antić, C.: Logic program proportions. Annals of Mathematics and Artificial Intelligence (2023b). https://doi.org/10.1007/s10472-023-09904-8
5. Antić, C.: On syntactically similar logic programs and sequential decompositions (2023c). arXiv:2109.05300pdf

6. Antić, C.: Sequential composition of answer set programs (2023d). arXiv:2104.12156pdf
7. Antić, C.: Sequential composition of propositional Krom logic programs (2023e). https://hal.science/hal-04023753
8. Apt, K.R.: Logic programming. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 493–574. Elsevier, Amsterdam (1990)
9. Apt, K.R.: From Logic Programming to Prolog. C.A.R. Hoare Series. Prentice Hall, Prentice Hall, Englewood Cliffs, NJ (1997)
10. Apt, K.R., Bezem, M.: Acyclic programs. N. Gener. Comput. **9**(3–4), 335–363 (1991)
11. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
12. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint logic programming–syntax and semantics. ACM Trans. Program. Lang. Syst. **23**(1), 1–29 (2001)
13. Bossi, A., Bugliesi, M., Gabbrielli, M., Levi, G., Meo, M.: Differential logic programs: Programming methodologies and semantics. Sci. Comput. Program. **27**, 217–262 (1996)
14. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
15. Brogi, A., Lamma, E., Mello, P.: Compositional model-theoretic semantics for logic programs. N. Gener. Comput. **11**, 1–21 (1992)
16. Brogi, A., Mancarella, P., Pedreschi, D., Turini, F.: Meta for modularising logic programming. In: META 1992, pp. 105–119 (1992b)
17. Brogi, A., Mancarella, P., Pedreschi, D., Turini, F.: Modular logic programming. ACM Trans. Program. Lang. Syst. **16**(4), 1361–1398 (1999)
18. Brogi, A., Turini, F.: Fully abstract compositional semantics for an algebra of logic programs. Theor. Comput. Sci. **149**(2), 201–229 (1995)
19. Bugliesi, M., Lamma, E., Mello, P.: Modularity in logic programming. J. Logic Program. **19–20**(1), 443–502 (1994)
20. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Trans. Knowl. Data Eng. **1**(1), 146–166 (1989)
21. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Surveys in Computer Science. Springer-Verlag, Berlin/Heidelberg (1990)
22. Chen, W., Kifer, M., Warren, D.S.: HiLog: A foundation for higher-order logic programming. J. Logic Program. **15**(3), 187–230 (1993)
23. Clark, K.L.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press, New York (1978)
24. Coelho, H. Cotta, J.C.: Prolog by Example. How to Learn, Teach, and Use It. Springer-Verlag, Berlin/Heidelberg (1988)
25. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T. Modular.: nonmonotonic logic programming revisited. In: ICLP, LNCS 5649, pp. 145–159. Springer-Verlag, Berlin/Heidelberg (2009)
26. Dix, J., Osorio, M., Zepeda, C.: A general theory of confluent rewriting systems for logic programming and its applications. Ann. Pure Appl. Logic **108**(1–3), 153–188 (2001)
27. Dong, G., Ginsburg, S.: On the decomposition of datalog program mappings. Theor. Comput. Sci. **76**(1), 143–177 (1990)
28. Dong, G., Ginsburg, S.: On decompositions of chain datalog programs into $\mathcal{P}$ (left-)linear 1-rule components. J. Logic Program. **23**(3), 203–236 (1995)
29. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive datalog. ACM Trans. Database Syst. **22**(3), 364–418 (1997)
30. Eiter, T., Ianni, G., Krennwallner, T.: Answer set programming: a primer. In: Reasoning Web. Semantic Technologies for Information Systems, vol. 5689 of Lecture Notes in Computer Science, pp. 40–110. Springer, Heidelberg (2009)
31. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the Semantic Web. Artif. Intell. **172**(12–13), 1495–1539 (2008)
32. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In: Kaelbling, L.P., Saffiotti, A. (eds.) IJCAI 2005, pp. 90–96 (2005)
33. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: semantics and complexity. In: Alferes, J., Leite, J. (eds.) JELIA 2004, LNCS 3229, pp. 200–212. Springer, Berlin (2004)
34. Gaifman, H. Shapiro, E.: Fully abstract institute of mathematics fully abstract compositional semantics for logic programs. In: Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages, pp. 134–142. ACM Press (1989)
35. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. N. Gener. Comput. **9**(3–4), 365–385 (1991)

36. Hill, P., Lloyd, J.W.: The Gödel Programming Language. The MIT Press (1994)
37. Hodges, W.: Logical features of Horn clauses. In: Gabbay, D.M., Hogger, C.J., Robinson, J.A.(eds.) Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 1, pp. 449–503. Clarendon Press, Oxford/New York (1994)
38. Howie, J.M.: Fundamentals of Semigroup Theory. London Mathematical Society Monographs New Series. Oxford University Press, Oxford (2003)
39. Ioannidis, Y.E., Wong, E.: Towards an algebraic theory of recursion. J. ACM **38**(2), 329–381 (1991)
40. Kowalski, R.: Logic for Problem Solving. North-Holland, New York (1979)
41. Krom, M.R.: The decision problem for a class of first-order formulas in which all disjunctions are binary. Math. Logic Q **13**(1–2), 15–20 (1967)
42. Leinster, T.: Higher Operads, Higher Categories, vol. 298 of London Mathematical Society Lecture Note Series. Cambridge University Press, Cambridge (2004)
43. Lifschitz, V.: Answer Set Programming. Springer Nature Switzerland AG, Cham, Switzerland (2019)
44. Lloyd, J.W.: Foundations of Logic Programming (2nd edn.). Springer-Verlag, Berlin, Heidelberg (1987)
45. Maher, M.J.: Equivalences of logic programs. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, chap. 16, pp. 627–658. Morgan Kaufmann Publishers (1988)
46. Makowsky, J.A.: Why Horn formulas matter in computer science: Initial structures and generic examples. J. Comput. Syst. Sci. **34**(2–3), 266–292 (1987)
47. Mancarella, P., Pedreschi, D.: An algebra of logic programs. In: Kowalski, R., Bowen, K.A. (eds.) Proceedings of the 5th International Conference on Logic Programming, pp. 1006–1023. The MIT Press, Cambridge MA (1988)
48. Miller, D., Nadathur, G.: Programming with Higher-Order Logic. Cambridge University Press (2012)
49. Oikarinen, E.: Modular Answer Set Programming. Ph.D. thesis, Helsinki University of Technology, Helsinki, Finland (2006)
50. Oikarinen, E., Janhunen, T.: Modular equivalence for normal logic programs. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) Proc. 17th European Conference on Artificial Intelligence, pp. 412–416. IOS Press, Amsterdam, Netherlands (2006)
51. O'Keefe, R.A.: Towards an algebra for constructing logic programs. In: SLP 1985, pp. 152–160 (1985)
52. Pelov, N.: Semantics of Logic Programs with Aggregates. Ph.D. thesis, Katholieke Universiteit Leuven, Leuven (2004)
53. Pereira, F.C.N., Shieber, S.M.: Prolog and Natural-Language Analysis. Microtome Publishing, Brookline, Massachusetts (2002)
54. Plambeck, T.: Semigroup techniques in recursive query optimization. In: PODS 1990, pp. 145–153 (1990a)
55. Plambeck, T.E.: Semigroups and Transitive Closure in Deductive Databases. Ph.D. thesis, Stanford University, Stanford (1990b)
56. Sterling, L., Shapiro, E.: The Art of Prolog: Advanced Programming Techniques (2nd edn.) The MIT Press, Cambridge MA (1994)
57. van Emden, M.H., Kowalski, R.: The semantics of predicate logic as a programming language. J. ACM **23**(4), 733–742 (1976)