TPLP: Page 1–46. © The Author(s), 2026. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

doi:10.1017/S1471068426100696

Set-like Operations on Propositional Logic Programs[*]

CHRISTIAN ANTIĆ
Technische Universität Wien, Austria
(*e-mail:* christian.antic@icloud.com)

submitted 16 July 2025 UTC; revised 10 March 2026 UTC; accepted 31 May 2026 UTC

Abstract

Composition and decomposition of logic programs have been studied extensively in the context of modularity, and decomposing a program along the dependency structure of its atoms — by strata or strongly connected components in datalog, and by splitting sets in the non-monotonic setting of answer set programming — is standard practice. All of these approaches operate on the level of rule sets: programs are cut along dependencies between atoms, while the internal structure of the individual rules remains untouched. In this paper, we complement this picture by a finer-grained algebra. We introduce set-like operations on (propositional Horn) logic programs — body-union, body-intersection, body-complement, body-subtraction, body-symmetric-difference, and body-power-set — which manipulate rule bodies in analogy to the corresponding set operations, and we study their algebraic laws and their interaction with sequential composition and the least model semantics. Our main technical result is a decomposition theorem in this algebra: every minimalist program — containing at most one rule for each rule head — is the body-union of Krom programs consisting only of rules with at most one body atom, in such a way that its least model is the intersection of the least models of these components; for arbitrary programs we obtain corresponding approximations. Since Krom programs are algebraically better behaved than arbitrary programs — composition is associative and distributes from the left — this may support decomposition-based reasoning and provides a basis for compositional program construction.

KEYWORDS: algebraic logic programming, logic program decomposition, propositional Horn theories, program synthesis, program transformation

1 Introduction

Rule-based reasoning is an important component of artificial intelligence. The definite propositional logic programs considered in this paper can be viewed as Horn theories equipped with least-model semantics (Lloyd 1987; Apt 1990, 1997; Hodges 1994; Sterling and Shapiro 1994).

[*] I would like to thank the reviewers for their thoughtful and constructive comments which improved the quality of the article.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Composition and decomposition of logic programs are not new concerns. Standard approaches decompose programs along the dependency structure of their atoms, for example by stratification or strongly connected components in datalog (Ceri *et al.* 1989, 1990), and by splitting sets in the non-monotonic setting of answer set programming (Lifschitz and Turner 1994). These approaches operate primarily at the level of rules or program modules, while the internal structure of individual rule bodies remains unchanged. The present work complements this perspective by studying finer-grained operations that manipulate and decompose rule bodies. To our knowledge, existing algebraic approaches do not directly provide the body-level decompositions and corresponding semantic relationships developed here.

The propositional Horn setting studied in this paper is fundamental to logic programming. Grounding transforms first-order programs into propositional ones, possibly over an infinite alphabet, while function-free programs over finite domains lead to finite propositional programs of the kind considered here. Propositional rule representations obtained by grounding are central to answer set programming (Gelfond and Lifschitz 1991; Marek and Truszczyński 1999) (see also Baral (2003); Brewka *et al.* (2011); Lifschitz (2019)) and to datalog (Ceri *et al.* 1989, 1990). Closely related structures are studied as propositional Horn theories in knowledge representation and reasoning.

The idea of introducing algebraic structure into programming dates back at least to the work of John Backus (Backus 1978), who advocated algebraic methods for constructing and reasoning about programs. In logic programming, algebraic and compositional perspectives have been developed extensively in connection with modularity and program composition (O'Keefe 1985; Mancarella and Pedreschi 1988; Hirani and Subrahmanian 1990; Bossi *et al.* 1996; Brogi *et al.* 1990, 1992a, 1992b; Dong 1988; Gaifman and Shapiro 1989; Dong and Ginsburg 1990; Ioannidis and Wong 1991; Brogi *et al.* 1994; Bugliesi *et al.* 1994; Hill and Lloyd 1994; Brogi and Turini 1995). The present approach is complementary to this work in that it manipulates the bodies of individual rules directly. The set-like operations introduced below enrich the algebra based on sequential composition (Antić 2024d), which is recalled in Section 2.2.3. This paper also forms part of a broader line of work on algebraic methods for logic program synthesis and transformation (Antić 2014, 2024c, 2024d, 2025).

We introduce a family of set-like operations on propositional Horn logic programs and study their algebraic laws, their interaction with sequential composition, and their relationship to least-model semantics. A rule has the form $a \leftarrow B$, where a is an atom from a finite alphabet A and $B \subseteq A$ is its body. The body may be empty, in which case the rule is a fact. For rules with the same head, the operations introduced in this paper are modeled on the familiar set operations $\cup$, $\cap$, ^{c}, $-$, $\triangle$, and $\mathfrak{P}$:

$$\begin{aligned}
(a \leftarrow B) \sqcup (a \leftarrow C) &:= a \leftarrow (B \cup C),\\
(a \leftarrow B) \sqcap (a \leftarrow C) &:= a \leftarrow (B \cap C),\\
(a \leftarrow B)^{\copyright} &:= a \leftarrow (A - B),\\
(a \leftarrow B) \boxminus (a \leftarrow C) &:= a \leftarrow (B - C),\\
(a \leftarrow B) \boxtriangle (a \leftarrow C) &:= a \leftarrow (B \triangle C),\\
(a \leftarrow B)^{\mathfrak{P}} &:= \{a \leftarrow C \mid C \subseteq B\}.
\end{aligned}$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

These operations are extended to programs by combining rules with the same head. We refer to them as body-union, body-intersection, body-complement, body-subtraction, body-symmetric-difference, and body-power-set, respectively.

Although these operations are modeled on classical set operations, their behavior at the program level is more subtle. In particular, if a program contains several rules with the same head, body-union and body-intersection need not be idempotent, and familiar distributivity laws may hold only as inclusions, modulo subsumption equivalence, or under additional minimality assumptions. We determine a number of such laws and study how the operations interact with reducts, sequential composition, supported models, and least models. We also show that body-union induces a well-defined operation on subsumption-equivalence classes and yields, together with sequential composition, a seminearring structure on the resulting quotient.

The main technical contribution concerns decomposition into Krom programs, that is, programs whose rules contain at most one body atom. Let M be a minimalist program, meaning that it contains at most one rule for each rule head, and let $m = \max\{1, w(M)\}$, where $w(M)$ is the maximal number of body atoms occurring in a rule of M. The Minimalist Decomposition Theorem shows that there are Krom programs $K_1, \ldots, K_m \subseteq M^\vee$ such that

$$M = K_1 \sqcup \ldots \sqcup K_m$$

and

$$M^\omega = K_1^\omega \sqcup \ldots \sqcup K_m^\omega = K_1^\omega \cap \ldots \cap K_m^\omega,$$

where $^\omega$ denotes the least-model operator and $M^\vee$ is the Krom program obtained by replacing each proper rule by its one-atom body components. Thus, both the program and its least model admit compatible representations in terms of Krom components. This is a structural existence and representation result rather than, by itself, an independent algorithm for computing M^ω.

For an arbitrary program P, rules with the same head may interact under body-union and produce additional rules. Exact syntactic and semantic reconstruction can therefore no longer be guaranteed. Nevertheless, the Decomposition Theorem provides Krom programs $K_1, \ldots, K_m \subseteq P^\vee$, where $m = \max\{1, w(P)\}$, satisfying

$$P \subseteq K_1 \sqcup \ldots \sqcup K_m$$

and

$$P^\omega \subseteq K_1^\omega \sqcup \ldots \sqcup K_m^\omega = K_1^\omega \cap \ldots \cap K_m^\omega.$$

The distinction between exact decomposition for minimalist programs and approximation for arbitrary programs reflects the additional combinations created by rules sharing the same head.

Krom programs provide particularly simple components because sequential composition is associative when the left factor is Krom and distributes from the left over program union, whereas these properties fail for arbitrary programs. The decomposition results therefore express complex programs in terms of components with better

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

algebraic behavior and may support further work on compositional reasoning, program transformation, and program construction.

In the final section, we sketch possible applications of the proposed operations to automatic logic programming (Deville and Lau 1994), including inductive logic programming (ILP), learning from interpretation transitions, and analogical logic programming. Further work may investigate extensions to first-order and higher-order programs and to non-monotonic programs under stable-model and answer-set semantics.

2 Preliminaries

In this section, we recall some basic algebraic structures Section 2.1 occurring in the paper, and the syntax and semantics of (propositional Horn) logic programs Section 2.2.

2.1 Algebraic structures

A *permutation* of a set A is any mapping $A \to A$ which is one-to-one and onto.

We denote the *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 $f(x) = x$.

We now turn to algebras consisting of a set together with a binary operation (Howie 2003). 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 $1a = a1 = a$ holds for all $a \in M$. A *semigroup* is a magma $(S, \cdot)$ in which $\cdot$ is associative. A *monoid* is a semigroup containing a unit element.

A *left* (resp., *right*) *zero* is an element 0 such that $0a = 0$ (resp., $a0 = 0$) holds for all a.

An equivalence relation $\sim$ on S is called *left* (resp., *right*) *compatible* with the operation on S if $x \sim y$ implies $zx \sim zy$ (resp., $xz \sim yz$), for all $x, y, z \in S$. A left (resp., right) compatible equivalence relation is called a left (resp., right) congruence, and an equivalence relation which is both a left and a right congruence is called a congruence. Given an equivalence relation $\sim$, we denote the equivalence class of an element a with respect to $\sim$ by $[a]_\sim$.

A *bimonoid* (Droste *et al.* 2010) is a structure $(M, +, \cdot, 0, 1)$ such that $(M, +, 0)$ and $(M, \cdot, 1)$ are both monoids. A bimonoid is a *strong bimonoid* if the operation $+$ is commutative and 0 acts as multiplicative zero, that is, $a \cdot 0 = 0 \cdot a = 0$ for every $a \in M$.

A *seminearring* (van Hoorn and van Rootselaar 1967; Golan 1999) 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, for all $a, b, c \in S$:

$$(a + b) \cdot c = (a \cdot c) + (b \cdot c) \quad \text{(right-distributivity)} \tag{1}$$

$$0 \cdot a = 0 \quad \text{(left zero)}. \tag{2}$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

We say that S is *idempotent* if $a + a = a$ holds for all $a \in S$. 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, for all $a, b, c \in S$:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \quad \text{(left-distributivity)}$$
$$a \cdot 0 = 0 \quad \text{(right zero)}.$$

For example, $(\mathbb{N}, +, \cdot, 0)$ and $(2^A, \cup, \cap, \emptyset)$ are semirings. A characteristic feature of semirings (as opposed to rings) is that addition needs not be invertible.

2.2 Logic programs

We shall now recall the syntax Section 2.2.1 and least model semantics Section 2.2.2 of (propositional Horn) logic programs (Apt 1990) and their sequential composition Section 2.2.3.

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

$$a \leftarrow b_1, \ldots, b_k, \quad k \geq 0, \tag{3}$$

where $a, b_1, \ldots, b_k \in A$ are propositional atoms, and we denote the set of all programs over A by $\mathbb{P}_A$. With a slight abuse of notation, we will not distinguish between $a \leftarrow b_1, \ldots, b_k$ and $a \leftarrow \{b_1, \ldots, b_k\}$, which will allow us to write the rule as $a \leftarrow B$ with $B = \{b_1, \ldots, b_k\}$, a notation heavily used in the rest of the paper.

The *width* of a rule r is the cardinality of its body,

$$w(r) := |b(r)|.$$

The width of a program P is defined by

$$w(P) := \max\big(\{w(r) \mid r \in P\} \cup \{0\}\big).$$

It will be convenient to define, for a rule r of the form equation (3) the *head* and *body operators* h and b as

$$h(r) := \{a\} \quad \text{and} \quad b(r) := \{b_1, \ldots, b_k\}$$

extended to programs rule-wise.

For any programs P and R, we have

$$h(P \cup R) = h(P) \cup h(R), \tag{4}$$
$$b(P \cup R) = b(P) \cup b(R), \tag{5}$$

which means that the head and body operators are homomorphisms with respect to set union.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

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 have

$$f(P \cup R) = f(P) \cup f(R),$$
$$p(P \cup R) = p(P) \cup p(R),$$

which means that the facts and proper rules operators are homomorphisms with respect to set union as well.

We call a rule r of the form equation (3) *Krom*[1] if it has at most one body atom, and we call a program *Krom* if it contains only Krom rules. We denote the set of all Krom programs over A by $\mathbb{K}_A$.

A *minimalist* program contains at most one rule for each rule head and we denote the set of all minimalist programs over A by $\mathbb{M}_A$. For example, the program

$$a \leftarrow b$$
$$a \leftarrow c$$

implicitly contains a disjunction as it is logically equivalent to the single rule

$$a \leftarrow (b \vee c).$$

Hence, a program is minimalist iff it does not implicitly contain disjunctions of that form.

The *unit program* over A is the minimalist Krom program

$$1_A := \{a \leftarrow a \mid a \in A\}. \tag{6}$$

2.2.2 Semantics

An *interpretation* over A is any set of atoms from A and we denote the set of all such interpretations by $\mathbb{I}_A$ or simply by $\mathbb{I}$ in case A is understood.

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 equation (3), $I \models r$ 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 and we denote the set of all models of P by $Mod(P)$. 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 $LM(P)$.

Define the *van Emden-Kowalski operator* (van Emden and Kowalski 1976) $T_P : \mathbb{I}_A \to \mathbb{I}_A$ of P, for every interpretation $I \in \mathbb{I}_A$, by

$$T_P(I) := \{a \in A \mid a \leftarrow B \in P,\ B \subseteq I\}.$$

The van Emden-Kowalski operator is at the core of logic programming since the models of P coincide with the prefixed points of T_P with respect to the inclusion ordering thus providing an operational characterization of models:

[1] Krom rules were first introduced and studied in Krom (1967) and are sometimes called "binary" in the literature.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Theorem 1 van Emden and Kowalski (1976).
An interpretation I is a model of a program P iff $T_P(I) \subseteq I$, that is,

$$Mod(P) = \{I \in \mathbb{I}_A \mid T_P(I) \subseteq I\}.$$

We call an interpretation I a *supported model* of P if I is a fixed point of T_P and we denote the set of all supported models of P by $Supp(P)$.

We say that P and R are *subsumption equivalent* (Maher 1988) — denoted by $P \approx R$ — if $T_P = T_R$. Notice that subsumption equivalence implies that P and R have the same models.

The van Emden-Kowalski operator is monotone on the set of all interpretations ordered by set inclusion and its *least fixed point computation* is given 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 van Emden and Kowalski (1976):

Proposition 2.
The least model of a program coincides with the least fixed point of its associated van Emden-Kowalski operator, that is, for any program P we have

$$LM(P) = T_P^\infty. \tag{7}$$

2.2.3 Sequential composition

We define the *(sequential) composition* (Antić 2024d) of two programs P and R as

$$P \circ R := \{a \leftarrow (B_1 \cup \ldots \cup B_k) \mid a \leftarrow b_1, \ldots, b_k \in P,\ b_1 \leftarrow B_1, \ldots, b_k \leftarrow B_k \in R,\ k \geq 0\},$$

which we will usually denote simply by PR.

For example, we have

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

Composition satisfies the following identities (among others):

$$P1 = 1P = P \tag{8}$$

$$(P \cup R)Q = PQ \cup RQ \tag{9}$$

$$P\emptyset = f(P) \tag{10}$$

$$h(P) = PA. \tag{11}$$

Unfortunately, composition is *not* associative (Antić 2024d, Example 10) and does not distribute from the left over union since

$$\{a \leftarrow b, c\}(\{b\} \cup \{c\}) = \{a\} \quad \text{whereas} \quad \{a \leftarrow b, c\}\{b\} \cup \{a \leftarrow b, c\}\{c\} = \emptyset. \tag{12}$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

However, for minimalist programs M and N, composition *is* associative in the sense that

$$(PM)N = P(MN) \tag{13}$$

holds for every program P (Antić 2024d, Theorem 11). The same holds true for a Krom program K and all programs P and R (Antić 2024d, Theorem 12), that is,

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

Moreover, in contrast to the general case (see equation (12) above), Krom programs distribute from the left thus satisfying (Antić 2024d, Theorem 12)

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

Together with the corresponding closure properties, this yields the following structural result:

Proposition 3.
The structure $(\mathbb{K}_A, \cup, \circ, \emptyset, 1_A)$ forms a seminearring with unit 1_A and the structure $(\mathbb{M}_A, \circ, 1_A)$ forms a monoid.

Proof.
See (Antić 2024d, Theorems 11 and 12). □

The van Emden-Kowalski operator T_P can be represented via composition in the sense that

$$T_P(I) = PI. \tag{16}$$

This correspondence allows us to directly define concepts in terms of programs instead of the van Emden-Kowalski operator:

Fact 4.
An interpretation I is a model of a program P iff $PI \subseteq I$. Moreover, I is a supported model of P iff $PI = I$.

Fact 5.
Two programs P and R are subsumption equivalent iff $PI = RI$ holds for every interpretation I.

More specifically, we first define

$$P \lesssim R \quad :\Leftrightarrow \quad PI \subseteq RI, \quad \text{for every interpretation } I,$$

and

$$P \approx R \quad :\Leftrightarrow \quad P \lesssim R \quad \text{and} \quad R \lesssim P.$$

We clearly have

$$P \lesssim R \quad \Rightarrow \quad LM(P) \subseteq LM(R). \tag{17}$$

Subsumption equivalence is a congruence with respect to sequential composition and set union as a direct consequence of Antić (2024d, Corollary 36) which means that we

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

can form the factor algebra

$$[\mathfrak{P}_A] := ([\mathbb{P}_A], \cup, \circ, [\emptyset], [1_A]), \tag{18}$$

where

$$[\mathbb{P}_A] := \{[P] \mid P \in \mathbb{P}_A\},$$

and

$$[P] \cup [R] := [P \cup R] \quad \text{and} \quad [P] \circ [R] := [P \circ R],$$

isomorphic to

$$\mathfrak{T}_A := (\mathbb{T}_{\mathbb{P}_A}, \cup, \circ, T_\emptyset, T_{1_A}),$$

where

$$\mathbb{T}_{\mathbb{P}_A} := \{T_P \mid P \in \mathbb{P}_A\}$$

denotes the set of all van Emden–Kowalski operators over A.

We have the following structural result:

Proposition 6.
The isomorphic structures $\mathfrak{T}_A$ and $[\mathfrak{P}_A]$ are seminearrings.

We now turn to the least model semantics of programs. For this, define the *(Kleene) star* of a program P by

$$P^* := \bigcup_{n \geq 0} P^n,$$

where (recall that composition is not associative)

$$P^0 := 1, \quad P^1 := P, \quad P^{n+1} := P^n P.$$

Moreover, define the *omega* of P by

$$P^\omega := f(P^*) \overset{10}{=} P^*\emptyset. \tag{19}$$

For every interpretation I, we have

$$I^* = 1 \cup I \tag{20}$$

$$I^\omega = I. \tag{21}$$

We have the following algebraic characterization of the least model of a program as shown in Antić (2024d, Theorem 40): the omega of P coincides with the least model of P, that is,

$$P^\omega = LM(P). \tag{22}$$

For an interpretation I, define the minimalist program

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

The program

$$A^\oplus = \{a \leftarrow A \mid a \in A\}$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

will play an important role in this paper as it occurs as the neutral element with respect to body-intersection (see Fact 30).

2.2.4 Reducts

Definition 7.
Given an interpretation I, we define the *head-reduct* and *body-reduct* of a program P with respect to I respectively by[2]

$$^I P := \{a \leftarrow B \in P \mid a \in I\} \quad \text{and} \quad P^I := \{a \leftarrow B \in P \mid B \subseteq I\}.$$

We now recollect some basic properties of reducts (see (Antić 2024d, Section 4.3)). For two interpretations I and J, we have

$$I^J = I \tag{23}$$
$$^J I = I \cap J. \tag{24}$$

According to Antić (2024d, Proposition 17), the body-reduct operator is an endomorphism on $(\mathbb{P}_A, \cup, \cap, \emptyset, A)$, that is, for any programs P, R and interpretation I, we have

$$(P \cup R)^I = P^I \cup R^I \tag{25}$$
$$(P \cap R)^I = P^I \cap R^I \tag{26}$$
$$\emptyset^I = \emptyset \tag{27}$$
$$A^I = A, \tag{28}$$

and the head-reduct operator is an endomorphism on $(\mathbb{P}_A, \cup, \cap, \emptyset)$, that is,

$$^I(P \cup R) = {}^I P \cup {}^I R \tag{29}$$
$$^I(P \cap R) = {}^I P \cap {}^I R \tag{30}$$
$$^I\emptyset = \emptyset. \tag{31}$$

Moreover, the left and right reducts can be rewritten in terms of composition as

$$^I P = (1^I)P \tag{32}$$
$$P^I = P1^I \tag{33}$$

and

$$(PR)^I = PR^I \tag{34}$$
$$^I(PR) = {}^I PR. \tag{35}$$

3 Body-union

The operations introduced in this section and the following ones form a natural family of set-like transformations on rule bodies. While body-union plays a central role in the decomposition results of Section 10, the remaining operations are studied to obtain a

[2] The body-reduct coincides with the Faber-Leone-Pfeifer reduct (Faber *et al.* 2011) which is well-known in answer set programming.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

more complete picture of the algebraic structure of logic programs and to understand how these operations interact with program semantics.

This section introduces the first set-like operation on logic programs in analogy to the union of sets:

Definition 8.
We define the *body-union* of P and R by

$$P \sqcup R := \{a \leftarrow (B \cup C) \mid a \leftarrow B \in P,\ a \leftarrow C \in R\}.$$

For example,

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

We have the following interactions between body-union and some of our basic operations:

$$h(P \sqcup R) = h(P) \cap h(R) \tag{36}$$

$$f(P \sqcup R) = f(P) \cap f(R) \tag{37}$$

$$p(P \sqcup R) = (f(P) \sqcup p(R)) \cup (p(P) \sqcup f(R)) \cup (p(P) \sqcup p(R)). \tag{38}$$

Notice that the first two identities mean that h and f are both homomorphisms from $(\mathbb{P}, \sqcup)$ to $(\mathbb{P}, \cap)$.

Recall that the union of sets is idempotent and despite the similarity between body-union and union, we have the following observation (but see Corollary 22 where we will show that body-union is at least idempotent modulo subsumption equivalence):

Observation 9.
Body-union is not idempotent even for Krom programs, that is, there is a Krom program K such that

$$K \sqcup K \neq K.$$

For example, we have

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

However, notice that body-union is idempotent on interpretations as a consequence of the forthcoming equation (48).

Fact 10.
The structure $(\mathbb{P}_A, \sqcup, A)$ is a commutative monoid, that is,

$$(P \sqcup Q) \sqcup R = P \sqcup (Q \sqcup R)$$
$$P \sqcup A = A \sqcup P = P$$
$$P \sqcup R = R \sqcup P$$

holds for all programs $P, Q, R \in \mathbb{P}_A$.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

We now analyze the interaction between body-union and union. Our first observation is the following structural result:

Theorem 11.
The structure $(\mathbb{P}_A, \cup, \sqcup, \emptyset, A)$ is a commutative idempotent semiring, that is, in addition to $(\mathbb{P}_A, \cup, \emptyset)$ being an idempotent commutative monoid and $(\mathbb{P}_A, \sqcup, A)$ being a commutative monoid (see Fact 10), we have

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

$$Q \sqcup (P \cup R) = (Q \sqcup P) \cup (Q \sqcup R) \tag{40}$$

$$\emptyset \sqcup P = P \sqcup \emptyset = \emptyset. \tag{41}$$

Proof.
A direct consequence of the rule-wise definition of body-union. □

Remark 12.
In particular, $(\mathbb{P}_A, \cup, \sqcup, \emptyset, A)$ is a strong bimonoid in the sense of Droste *et al.* (2010). Notice that the structure with reversed roles $(\mathbb{P}_A, \sqcup, \cup, A, \emptyset)$ is a bimonoid but not a strong one, since $P \cup A \neq A$ in general.

The next result shows that $\cup$ does in general *not* distribute over $\sqcup$ (but see Theorem 14 where we show that it distributes modulo subsumption equivalence):

Observation 13.
There are an interpretation I and Krom programs K and L such that

$$(I \sqcup K) \cup L \neq (I \cup L) \sqcup (K \cup L).$$

For example, for

$$I := \{a\} \quad \text{and} \quad K := \{a \leftarrow b\} \quad \text{and} \quad L := \{a \leftarrow c\}$$

we obtain

$$(\{a\} \sqcup \{a \leftarrow b\}) \cup \{a \leftarrow c\} = \left\{ \begin{array}{l} a \leftarrow b \\ a \leftarrow c \end{array} \right\}$$

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

Theorem 14.
For any programs P, Q, R, we have

$$(P \sqcup R) \cup Q \approx (P \cup Q) \sqcup (R \cup Q).$$

Proof.
For every interpretation I, we have (the forward references are non-circular and thus harmless)

$$[(P \sqcup R) \cup Q]I \overset{9}{=} (P \sqcup R)I \cup QI$$

$$\overset{45}{=} (PI \sqcup RI) \cup QI$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

$$\begin{aligned}
&\overset{48}{=} (PI \cap RI) \cup QI \\
&= (PI \cup QI) \cap (RI \cup QI) \\
&\overset{9}{=} (P \cup Q)I \cap (R \cup Q)I \\
&\overset{48}{=} (P \cup Q)I \sqcup (R \cup Q)I \\
&\overset{45}{=} [(P \cup Q) \sqcup (R \cup Q)]I.
\end{aligned}$$

□

Proposition 15.
For any programs P, Q, R, we have

$$Q \sqcup (P \cap R) \subseteq (Q \sqcup P) \cap (Q \sqcup R). \tag{42}$$

Proof.

$$\begin{aligned}
Q \sqcup (P \cap R) &= \{a \leftarrow (B \cup C) \mid a \leftarrow B \in Q,\ a \leftarrow C \in P \cap R\} \\
&= \{a \leftarrow (B \cup C) \mid a \leftarrow B \in Q,\ a \leftarrow C \in P,\ a \leftarrow C \in R\} \\
&\subseteq \{a \leftarrow (B \cup C) \mid a \leftarrow B \in Q,\ a \leftarrow C \in P\} \\
&\cap \{a \leftarrow (B \cup C) \mid a \leftarrow B \in Q,\ a \leftarrow C \in R\} \\
&= (Q \sqcup P) \cap (Q \sqcup R),
\end{aligned}$$

where the inclusion cannot be turned into an identity since a rule occurring in both programs on the right may originate from a rule in P and a rule in R which are distinct and hence do not occur in $P \cap R$. □

Observation 16.
For $Q := \{a \leftarrow b\}$ and $P := \{a \leftarrow c\}$ and $R := \{a \leftarrow b, c\}$, we have $P \cap R = \emptyset$ and therefore

$$Q \sqcup (P \cap R) \overset{41}{=} \emptyset \subsetneq \{a \leftarrow b, c\} = \{a \leftarrow b, c\} \cap \{a \leftarrow b, c\} = (Q \sqcup P) \cap (Q \sqcup R).$$

We now study the interplay between body-union and the sequential composition of programs which is used to compute the semantic $^\omega$-operator. We first prove that composition distributes from the right over body-union (but not from the left; see Observation 21):

Theorem 17.
For any programs P, Q, R, we have

$$(P \sqcup R)Q \subseteq PQ \sqcup RQ, \tag{43}$$

$$(P \sqcup R)Q \approx PQ \sqcup RQ, \tag{44}$$

and equality

$$(P \sqcup R)Q = PQ \sqcup RQ \tag{45}$$

holds in each of the following two cases: (i) $B \cap C = \emptyset$ for all $a \leftarrow B \in P$ and $a \leftarrow C \in R$; (ii) $Q \sqcup Q = Q$.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Proof.
Given a set of atoms B, we call a family $(C_b)_{b\in B}$ of sets of atoms a *Q-selection for B* if $b \leftarrow C_b \in Q$ holds for every $b \in B$, so that the definition of composition reads as

$$PQ = \left\{ a \leftarrow \bigcup_{b\in B} C_b \;\middle|\; a \leftarrow B \in P,\ (C_b)_{b\in B} \text{ a } Q\text{-selection for } B \right\}.$$

We compute

$$\begin{aligned}
&(P \sqcup R)Q \\
&\quad = \{a \leftarrow (B \cup C) \mid a \leftarrow B \in P,\ a \leftarrow C \in R\}Q \\
&\quad = \left\{ a \leftarrow \bigcup_{b\in B\cup C} F_b \;\middle|\; a \leftarrow B \in P,\ a \leftarrow C \in R,\ (F_b)_{b\in B\cup C} \text{ a } Q\text{-selection for } B\cup C \right\} \\
&\quad \subseteq \left\{ a \leftarrow \left(\bigcup_{b\in B} D_b \cup \bigcup_{b\in C} E_b \right) \;\middle|\; a \leftarrow B \in P,\ a \leftarrow C \in R,\ (D_b)_{b\in B},\ (E_b)_{b\in C}\ Q\text{-selections} \right\} \\
&\quad = \left\{ a \leftarrow \bigcup_{b\in B} D_b \;\middle|\; a \leftarrow B \in P,\ (D_b)_{b\in B} \text{ a } Q\text{-selection for } B \right\} \\
&\qquad \sqcup \left\{ a \leftarrow \bigcup_{b\in C} E_b \;\middle|\; a \leftarrow C \in R,\ (E_b)_{b\in C} \text{ a } Q\text{-selection for } C \right\} \\
&\quad = PQ \sqcup RQ,
\end{aligned}$$

which proves equation (43). The inclusion in the third line holds since the restrictions $(F_b)_{b\in B}$ and $(F_b)_{b\in C}$ of a Q-selection for $B\cup C$ are Q-selections for B and C satisfying $\bigcup_{b\in B} F_b \cup \bigcup_{b\in C} F_b = \bigcup_{b\in B\cup C} F_b$. It is an identity — and hence proves equation (45) — in each of the cases (i) and (ii), since given Q-selections $(D_b)_{b\in B}$ and $(E_b)_{b\in C}$, the family

$$F_b := \begin{cases} D_b & b \in B - C, \\ E_b & b \in C - B, \\ D_b \cup E_b & b \in B \cap C, \end{cases}$$

is a Q-selection for $B\cup C$ satisfying $\bigcup_{b\in B\cup C} F_b = \bigcup_{b\in B} D_b \cup \bigcup_{b\in C} E_b$: in case (i) the third line is vacuous as $B\cap C = \emptyset$, and in case (ii) we have $b \leftarrow (D_b \cup E_b) \in Q \sqcup Q = Q$ for every $b \in B\cap C$.

Finally, we prove equation (44). Notice first that every interpretation J satisfies $J \sqcup J \overset{49}{=} J$, which means that equation (45) is applicable whenever the right factor is an interpretation. Hence, for every interpretation I we compute

$$\begin{aligned}
[(P \sqcup R)Q]I &\overset{Proposition\ 6}{\approx} (P \sqcup R)(QI) \overset{45}{=} P(QI) \sqcup R(QI) \overset{48}{=} P(QI) \cap R(QI) \\
&\overset{Proposition\ 6}{\approx} (PQ)I \cap (RQ)I \overset{48}{=} (PQ)I \sqcup (RQ)I \overset{45}{=} [PQ \sqcup RQ]I,
\end{aligned}$$

where we may replace $\approx$ by $=$ since all programs occurring in the computation are interpretations and subsumption equivalent interpretations coincide. □

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Remark 18.
Condition (ii) covers the two cases occurring throughout this paper: every interpretation I satisfies $I \sqcup I \overset{49}{=} I$, every minimalist program M satisfies $M \sqcup M = M$, and the program $A^* \overset{20}{=} 1_A \cup A$ used in Proposition 57 satisfies $A^* \sqcup A^* = A^*$. On the other hand, $Q := \{b \leftarrow d,\ b \leftarrow e\}$ satisfies $Q \sqcup Q = Q \cup \{b \leftarrow d, e\} \neq Q$, and indeed $P := \{a \leftarrow b\}$ and $R := \{a \leftarrow b, c\}$ together with $Q' := Q \cup \{c \leftarrow c\}$ give

$$(P \sqcup R)Q' = \left\{ \begin{matrix} a \leftarrow c, d \\ a \leftarrow c, e \end{matrix} \right\} \subsetneq \left\{ \begin{matrix} a \leftarrow c, d \\ a \leftarrow c, e \\ a \leftarrow c, d, e \end{matrix} \right\} = PQ' \sqcup RQ',$$

which shows that the inclusion in equation (43) can in general not be turned into an equality.

Lemma 19.
Subsumption equivalence is a congruence with respect to body-union, that is, $P \approx P'$ and $R \approx R'$ imply $P \sqcup R \approx P' \sqcup R'$, for all programs P, P', R, R'.

Proof.
Since every interpretation I satisfies $I \sqcup I \overset{49}{=} I$, we may apply equation (45) to compute

$$(P \sqcup R)I \overset{45}{=} PI \sqcup RI = P'I \sqcup R'I \overset{45}{=} (P' \sqcup R')I,$$

where the middle identity holds by assumption. □

Theorem 20.
The structure $([\mathbb{P}_A], \sqcup, \circ, [A], [1_A])$ forms a seminearring with unit $[1_A]$, where $[P] \sqcup [R] := [P \sqcup R]$.

Proof.
Body-union is well-defined on subsumption equivalence classes by Lemma 19, and $([\mathbb{P}_A], \sqcup, [A])$ is a monoid since $(\mathbb{P}_A, \sqcup, A)$ is one by Fact 10. Composition is associative modulo subsumption equivalence by Proposition 6, right-distributivity is equation (44), and $[A]$ is a left zero with respect to composition since

$$AP = A$$

holds for every program $P \in \mathbb{P}_A$ — indeed, A consists of facts only, which are left unchanged by composition. Finally, $[1_A]$ is a unit since $P1_A = 1_A P = P$. □

Observation 21.
Unfortunately, the distributivity law in equation (44) does *not* hold if composition is computed from the left as there are Krom programs K, L, N such that

$$K(L \sqcup N) \not\approx KL \sqcup KN,$$

which means that in general we have

$$Q(P \sqcup R) \not\approx QP \sqcup QR.$$

For example, we have

$$\left\{ \begin{matrix} a \leftarrow b \\ a \leftarrow c \end{matrix} \right\} (\{b \leftarrow b\} \sqcup \{c \leftarrow c\}) = \left\{ \begin{matrix} a \leftarrow b \\ a \leftarrow c \end{matrix} \right\} \emptyset = \emptyset$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

whereas

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

and the programs $\emptyset$ and $\{a \leftarrow b, c\}$ are, of course, not subsumption equivalent.

The next result is analogous to $P \sqcup P = P$:

Corollary 22.
Body-union is idempotent modulo subsumption equivalence, that is, for every program P,

$$P \sqcup P \approx P.$$

Proof.
For every interpretation I,

$$(P \sqcup P)I \overset{45}{=} PI \sqcup PI \overset{48}{=} PI. \qquad \square$$

Theorem 23.
Every model of P is a model of $P \sqcup R$, for any programs P and R. That is,

$$Mod(P) \subseteq Mod(P \sqcup R).$$

Proof.
By Theorem 1, we know that an interpretation I is a model of P iff $T_P(I) \subseteq I$, which by equation (16) is equivalent to $PI \subseteq I$. Now by equation (45), we have $(P \sqcup R)I = PI \sqcup RI \overset{48}{=} PI \cap RI \subseteq I$, which means that I is a model of $P \sqcup R$. $\square$

Proposition 24.
An interpretation I is a supported model of $P \sqcup R$ iff $PI \cap RI = I$, that is,

$$Supp(P \sqcup R) = \{I \in \mathbb{I} \mid PI \cap RI = I\}.$$

Proof.
By Fact 4, an interpretation I is a supported model of $P \sqcup R$ iff $(P \sqcup R)I \overset{45}{=} PI \sqcup RI \overset{48}{=} PI \cap RI = I$. $\square$

The next result is analogous to

$$P \sqcup R = P \quad \Leftrightarrow \quad R \subseteq P.$$

Proposition 25.
For any programs P and R,

$$P \sqcup R \approx P \quad \Leftrightarrow \quad P \lesssim R.$$

Proof.
($\Rightarrow$) For any interpretation I, we have

$$PI = (P \sqcup R)I = PI \sqcup RI = PI \cap RI \subseteq RI,$$

which implies $P \lesssim R$.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

($\Leftarrow$) From the assumption $PI \subseteq RI$, for every interpretation I, we deduce

$$(P \sqcup R)I = PI \sqcup RI = PI \cap RI = PI,$$

which implies $P \sqcup R \approx P$. □

Reducing a program to a restricted alphabet is a fundamental operation on programs which we will often use in the rest of the paper and in this section we analyze the interactions between body-union and reducts.

We can simplify the computation of the body-union using head-reducts as

$$P \sqcup R = {}^{h(R)}P \sqcup {}^{h(P)}R.$$

Notice that we have

$${}^{f(R)}P \cup {}^{f(P)}R \subseteq P \sqcup R.$$

Computing the body of $P \sqcup R$ is simple using head-reducts:

$$b(P \sqcup R) = b({}^{h(R)}P) \cup b({}^{h(P)}R). \tag{46}$$

Interestingly, the left reduct can be represented via body-union as (compare with equation (80))

$$I \sqcup P = P \sqcup I = {}^{I}P. \tag{47}$$

This implies that for any interpretations I and J,

$$I \sqcup J \overset{47}{=} {}^{J}I \overset{24}{=} I \cap J, \tag{48}$$

and therefore

$$I \sqcup I = I. \tag{49}$$

Moreover, it implies that computing the proper rules in $P \sqcup R$ can be written as

$$p(P \sqcup R) \overset{38,47}{=} (p(P) \sqcup p(R)) \cup {}^{f(R)}p(P) \cup {}^{f(P)}p(R). \tag{50}$$

Proposition 26.
The body-reduct operator is an endomorphism on the monoid $(\mathbb{P}_A, \sqcup, A)$, that is,

$$(P \sqcup R)^I = P^I \sqcup R^I$$

$$A^I \overset{23}{=} A.$$

Moreover, the head-reduct operator is an endomorphism on the semigroup $(\mathbb{P}_A, \sqcup)$, that is,

$${}^{I}(P \sqcup R) = {}^{I}P \sqcup {}^{I}R. \tag{51}$$

Proof.
The first two identities hold trivially. To prove the third, we compute

$${}^{I}(P \sqcup R) \overset{47}{=} I \sqcup P \sqcup R \overset{48}{=} I \sqcup P \sqcup I \sqcup R \overset{47}{=} {}^{I}P \sqcup {}^{I}R.$$ □

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Remark 27.
Notice that the head-reduct operator fails to be an endomorphism on the monoid $(\mathbb{P}_A, \sqcup, A)$ since, in case $I \neq A$, we have ${}^I A \overset{24}{=} I \cap A = I \neq A$.

Proposition 28.
For any programs P and R and interpretation I,

$$(P \cup I) \sqcup (R \cup I) = I \cup {}^I P \cup {}^I R \cup (P \sqcup R). \tag{52}$$

Proof.

$$\begin{aligned}(P \cup I) \sqcup (R \cup I) &= \{a \leftarrow B \cup C \mid a \leftarrow B \in P \cup I,\ a \leftarrow C \in R \cup I\}\\ &= \{a \mid a \in I\} \cup \{a \leftarrow B \mid a \leftarrow B \in P,\ a \in I\}\\ &\quad \cup \{a \leftarrow C \mid a \in I,\ a \leftarrow C \in R\} \cup \{a \leftarrow B \cup C \mid a \leftarrow B \in P,\ a \leftarrow C \in R\}\\ &= I \cup (I \sqcup P) \cup (I \sqcup R) \cup (P \sqcup R)\\ &\overset{47}{=} I \cup {}^I P \cup {}^I R \cup (P \sqcup R).\end{aligned}$$

□

4 Body-intersection

We now turn to our next set-like operation which is analogous to set intersection:

Definition 29.
Define the *body-intersection* of P and R by

$$P \sqcap R := \{a \leftarrow (B \cap C) \mid a \leftarrow B \in P,\ a \leftarrow C \in R\}.$$

Body-intersection is not idempotent even on Krom programs as is witnessed by the following counterexample:

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

However, body-intersection coincides with intersection on interpretations,

$$I \sqcap J = I \cap J, \tag{53}$$

which means that it is idempotent on interpretations,

$$I \sqcap I \overset{53}{=} I \cap I = I.$$

Fact 30.
The structure $(\mathbb{P}_A, \sqcap, A^{\oplus})$ is a commutative monoid, that is,

$$(P \sqcap Q) \sqcap R = P \sqcap (Q \sqcap R) \tag{54}$$

$$P \sqcap A^{\oplus} = A^{\oplus} \sqcap P = P \tag{55}$$

$$P \sqcap R = R \sqcap P. \tag{56}$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

We have the following interactions between body-intersection and and the head operator:

$$h(P \sqcap R) = h(P) \cap h(R).$$

This means that the head operator h is a homomorphism from $(\mathbb{P}, \sqcap)$ to $(\mathbb{P}, \cap)$.

Unfortunately, computing $f(P \sqcap R)$ is not as simple as computing $f(P \sqcup R)$ (see equation (37)) and before we give a formula in equation (57) below, we first introduce the following auxiliary construct:

Definition 31.
Define the *bowtie* of P and R by

$$P \bowtie R := (P \ltimes R) \cup (R \ltimes P).$$

where

$$P \ltimes R := \{a \leftarrow B \in P \mid \text{there exists } a \leftarrow C \in R \text{ with } B \cap C = \emptyset\}.$$

We can now compute the facts in $P \sqcap R$ as

$$f(P \sqcap R) = (f(P) \cap f(R)) \cup h(P \bowtie R) \overset{37}{=} f(P \sqcup R) \cup h(P \bowtie R). \tag{57}$$

Observation 32.
There are Krom programs K, L, M such that

$$M(K \sqcap L) \not\approx MK \sqcap ML,$$

which means that in general we have

$$Q(P \sqcap R) \neq QP \sqcap QR.$$

This is analogous to what we have encountered in Observation 21 and a similar counterexample works here as well: we have

$$\left\{ \begin{matrix} a \leftarrow b \\ a \leftarrow c \end{matrix} \right\} (\{b \leftarrow b\} \sqcap \{c \leftarrow c\}) = \emptyset$$

whereas

$$\left\{ \begin{matrix} a \leftarrow b \\ a \leftarrow c \end{matrix} \right\} \{b \leftarrow b\} \sqcap \left\{ \begin{matrix} a \leftarrow b \\ a \leftarrow c \end{matrix} \right\} \{c \leftarrow c\} = \{a \leftarrow b\} \sqcap \{a \leftarrow c\} = \{a\}.$$

Moreover, there are minimalist programs M, N and an interpretation I such that

$$(M \sqcap N)I \neq MI \sqcap NI,$$

which means that in general we have[3]

$$(P \sqcap R)Q \not\approx PQ \sqcap RQ.$$

Compare this with the positive result in Theorem 17! To see why, consider the counterexample

$$(\{a \leftarrow b, c\} \sqcap \{a \leftarrow c, d\})\{c\} = \{a\} \neq \{a \leftarrow b, c\}\{c\} \sqcap \{a \leftarrow c, d\}\{c\} = \emptyset.$$

[3] Compare to Theorem 17.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

We shall now analyze the interaction between body-intersection and reducts. Our first observation is as follows:

Observation 33.
In contrast to Proposition 26, the body-reduct operator is in general *not* an endomorphism on the monoid $(\mathbb{P}_A, \sqcap, A^{\oplus})$ since, in general, we have

$$(P \sqcap R)^I \not\subseteq P^I \sqcap R^I$$

as is witnessed by the following counterexample:

$$(\{a \leftarrow b\} \sqcap \{a \leftarrow c\})^{\{b\}} = \{a\}^{\{b\}} \overset{23}{=} \{a\}$$

whereas

$$\{a \leftarrow b\}^{\{b\}} \sqcap \{a \leftarrow c\}^{\{b\}} = \{a \leftarrow b\} \sqcap \emptyset = \emptyset.$$

However, we do have

$$P^I \sqcap R^I \subseteq (P \sqcap R)^I$$

since

$$P^I \sqcap R^I = \{a \leftarrow B \cap C \mid B, C \subseteq I\} \subseteq \{a \leftarrow B \cap C \mid B \cap C \subseteq I\} = (P \sqcap R)^I.$$

Proposition 34.
The head-reduct operator is an endomorphism on the semigroup $(\mathbb{P}_A, \sqcap)$, that is,

$$^I(P \sqcap R) = {}^I P \sqcap {}^I R.$$

Proof.
We have

$$^I(P \sqcap R) \overset{47}{=} I \sqcup (P \sqcap R) \overset{77}{=} (I \sqcup P) \sqcap (I \sqcup R) \overset{47}{=} {}^I P \sqcap {}^I R,$$

where the forward reference to equation (77) is non-circular and thus harmless. □

Proposition 35.
We have the following formula for the computation of the head-reduct:

$$^I P = {}^I(A^{\oplus}) \sqcap P. \tag{58}$$

Proof.

$$\begin{aligned} {}^I(A^{\oplus}) \sqcap P &= \{a \leftarrow A \mid a \in I\} \sqcap P = \{a \leftarrow (B \cap A) \mid a \leftarrow B \in P,\ a \in I\} \\ &= \{a \leftarrow B \in P \mid a \in I\} = {}^I P. \end{aligned}$$

□

5 Body-complement

In analogy to the complement of sets, we introduce the following unary operation on programs:

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Definition 36.
We define the *body-complement* of P by

$$P^{\copyright} := \{a \leftarrow (A - B) \mid a \leftarrow B \in P\}.$$

Example 37.
Let $A := \{a, b, c\}$. We then have, for instance,

$$\emptyset^{\copyright} = \emptyset, \qquad \{a\}^{\copyright} = \{a \leftarrow a, b, c\}, \qquad \left\{ \begin{array}{l} a \\ b \leftarrow a \\ c \leftarrow a, b \end{array} \right\}^{\copyright} = \left\{ \begin{array}{l} a \leftarrow a, b, c \\ b \leftarrow b, c \\ c \leftarrow c \end{array} \right\}.$$

Body-complement is involutive,

$$(P^{\copyright})^{\copyright} = P. \tag{59}$$

Moreover, it satisfies the following identities (among others):

$$h(P^{\copyright}) = h(P) \tag{60}$$

$$b(P^{\copyright}) = A - \bigcap_{a \leftarrow B \in P} B \tag{61}$$

$$f(P^{\copyright}) = h(P \cap A^{\oplus}) = \{a \in A \mid a \leftarrow A \in P\} \tag{62}$$

$$A^{\copyright} = A^{\oplus} \tag{63}$$

$$I^{\copyright} = {}^{I}(A^{\oplus}). \tag{64}$$

Fact 38.
Body-complement is an endomorphism on the boolean algebra $(\mathbb{P}_A, \cup, \cap, {}^c, \emptyset, F_A)$, that is,

$$(P \cup R)^{\copyright} = P^{\copyright} \cup R^{\copyright} \tag{65}$$

$$(P \cap R)^{\copyright} = P^{\copyright} \cap R^{\copyright} \tag{66}$$

$$(P^c)^{\copyright} = (P^{\copyright})^c \tag{67}$$

$$\emptyset^{\copyright} = \emptyset \tag{68}$$

$$F_A^{\copyright} = F_A, \tag{69}$$

where

$$F_A := \{a \leftarrow B \mid a \in A,\ B \subseteq A\} \tag{70}$$

is the *full program* over A consisting of all possible rules and facts that can be formed from A.

This gives us the following formula for the computation of body-complement:

$$P^{\copyright} = (f(P) \cup p(P))^{\copyright} \overset{65}{=} f(P)^{\copyright} \cup p(P)^{\copyright} \overset{64}{=} {}^{f(P)}(A^{\oplus}) \cup p(P)^{\copyright}.$$

The next result shows that we can smoothly adapt De Morgan's laws to body-union, body-intersection, and body-complement as desired:

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Proposition 39.
For any programs P and R,

$$(P \sqcup R)^{\copyright} = P^{\copyright} \sqcap R^{\copyright} \tag{71}$$
$$(P \sqcap R)^{\copyright} = P^{\copyright} \sqcup R^{\copyright}. \tag{72}$$

Proof.
Using De Morgan's laws, we compute

$$\begin{aligned}(P \sqcup R)^{\copyright} &= \{a \leftarrow (B \cup C) \mid a \leftarrow B \in P,\ a \leftarrow C \in R\}^{\copyright} \\ &= \{a \leftarrow (A - (B \cup C)) \mid a \leftarrow B \in P,\ a \leftarrow C \in R\} \\ &= \{a \leftarrow ((A - B) \cap (A - C)) \mid a \leftarrow B \in P,\ a \leftarrow C \in R\} \\ &= \{a \leftarrow (A - B) \mid a \leftarrow B \in P\} \sqcap \{a \leftarrow (A - C) \mid a \leftarrow C \in R\} \\ &= P^{\copyright} \sqcap R^{\copyright}.\end{aligned}$$

and a similar computation shows the second identity. □

The body-complement of a Krom program K is

$$K^{\copyright} = \{a \leftarrow A \mid a \in f(K)\} \cup \{a \leftarrow (A - \{b\}) \mid a \leftarrow b \in p(K)\}$$

which can be written more compactly as

$$K^{\copyright} = {}^{f(K)}(A^{\oplus}) \cup p(K)^{\copyright}.$$

The identities

$$(P^{\copyright})^I = \{a \leftarrow (A - B) \mid a \leftarrow B \in P,\ A - B \subseteq I\}$$
$$(P^I)^{\copyright} = \{a \leftarrow (A - B) \mid a \leftarrow B \in P,\ B \subseteq I\}$$

show that in general we have

$$(P^{\copyright})^I \neq (P^I)^{\copyright},$$

which means that the body-reduct does not commute with body-complement, but see Proposition 40 below where we show that the head-reduct does commute with body-complement. At least we (trivially) have

$$(P^{\copyright})^A = (P^A)^{\copyright}.$$

Proposition 40.
Body-complement commutes with head-reducts, that is, for any program P and interpretation I,

$${}^I(P^{\copyright}) = ({}^I P)^{\copyright}. \tag{73}$$

Proof.
An immediate consequence from the definition of body-complement and head-reducts. To illustrate some of the identities obtained above, we additionally compute

$${}^I(P^{\copyright}) \overset{47}{=} I \sqcup P^{\copyright} \overset{71}{=} (I^{\copyright} \sqcap P)^{\copyright} \overset{64}{=} ({}^I(A^{\oplus}) \sqcap P)^{\copyright} \overset{58}{=} ({}^I P)^{\copyright}.$$ □

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Theorem 41.
For all programs P, Q, R, we have

$$(P \sqcup R) \sqcap Q \subseteq (P \sqcap Q) \sqcup (R \sqcap Q) \tag{74}$$
$$(P \sqcap R) \sqcup Q \subseteq (P \sqcup Q) \sqcap (R \sqcup Q), \tag{75}$$

and in case M is a minimalist program, we have the identities

$$(P \sqcup R) \sqcap M = (P \sqcap M) \sqcup (R \sqcap M) \tag{76}$$
$$(P \sqcap R) \sqcup M = (P \sqcup M) \sqcap (R \sqcup M). \tag{77}$$

Proof.
Applying the well-known distributivity laws of the set operations, we compute

$$\begin{aligned}
&(P \sqcup R) \sqcap Q \\
&\quad = \{a \leftarrow ((B \cup C) \cap D) \mid a \leftarrow B \in P,\ a \leftarrow C \in R,\ a \leftarrow D \in Q\} \\
&\quad = \{a \leftarrow ((B \cap D) \cup (C \cap D)) \mid a \leftarrow B \in P,\ a \leftarrow C \in R,\ a \leftarrow D \in Q\} \\
&\quad \subseteq \{a \leftarrow ((B \cap D) \cup (C \cap D')) \mid a \leftarrow B \in P,\ a \leftarrow C \in R,\ a \leftarrow D \in Q,\ a \leftarrow D' \in Q\} \\
&\quad = (P \sqcap Q) \sqcup (R \sqcap Q)
\end{aligned}$$

and, analogously,

$$\begin{aligned}
&(P \sqcap R) \sqcup Q \\
&\quad = \{a \leftarrow ((B \cap C) \cup D) \mid a \leftarrow B \in P,\ a \leftarrow C \in R,\ a \leftarrow D \in Q\} \\
&\quad = \{a \leftarrow ((B \cup D) \cap (C \cup D)) \mid a \leftarrow B \in P,\ a \leftarrow C \in R,\ a \leftarrow D \in Q\} \\
&\quad \subseteq \{a \leftarrow ((B \cup D) \cap (C \cup D')) \mid a \leftarrow B \in P,\ a \leftarrow C \in R,\ a \leftarrow D \in Q,\ a \leftarrow D' \in Q\} \\
&\quad = (P \sqcup Q) \sqcap (R \sqcup Q).
\end{aligned}$$

In both computations, the inclusion is an identity in case Q is minimalist, since then D and D' are both the body of the unique rule in Q with head a — which proves equations (76) and (77). □

Observation 42.
The inclusions in equations (74) and (75) can in general not be turned into identities. For example, for the non-minimalist program $Q := \{a \leftarrow b,\ a \leftarrow c\}$ and $P := \{a \leftarrow b\}$ and $R := \{a \leftarrow c\}$, we have

$$(P \sqcup R) \sqcap Q = (P \sqcap R) \sqcup Q = \left\{ \begin{array}{l} a \leftarrow b \\ a \leftarrow c \end{array} \right\} \subsetneq \left\{ \begin{array}{l} \underline{a} \\ a \leftarrow b \\ a \leftarrow c \\ \underline{a \leftarrow b, c} \end{array} \right\} = (P \sqcap Q) \sqcup (R \sqcap Q)$$
$$= (P \sqcup Q) \sqcap (R \sqcup Q).$$

6 Body-subtraction

In this section, we introduce an operation on programs which mimics set difference:

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Definition 43.
Define the *body-subtraction* of P and R by

$$P \boxminus R := \{a \leftarrow (B - C) \mid a \leftarrow B \in P,\ a \leftarrow C \in R\}.$$

For example, we have

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

This operation satisfies the following identities (among others):

$$P \boxminus \emptyset = \emptyset \boxminus P = \emptyset$$
$$P \boxminus A = P$$
$$P \boxminus A^{\oplus} = h(P).$$

Moreover, we have

$$M \boxminus M = h(M)$$

for every minimalist program M. Notice that the identity may fail for non-minimalist programs as is witnessed by the counterexample:

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

We thus have

$$P \boxminus P \supseteq h(P).$$

Fact 44.
The head operator is a homomorphism $h : (\mathbb{P}_A, \boxminus) \to (\mathbb{I}_A, \cap)$ thus satisfying

$$h(P \boxminus R) = h(P) \cap h(R).$$

Recall that for sets X and Y,

$$X - Y = X \cap Y^c.$$

Analogously, we have:

Proposition 45.
For any programs P and R,

$$P \boxminus R = P \sqcap R^{\copyright}. \tag{78}$$

Proof.
Using De Morgan's law, we compute

$$\begin{aligned} P \sqcap R^{\copyright} &= \{a \leftarrow (B \cap C) \mid a \leftarrow B \in P,\ a \leftarrow C \in R^{\copyright}\} \\ &= \{a \leftarrow (B \cap C) \mid a \leftarrow B \in P,\ a \leftarrow (A - C) \in R\} \\ &= \{a \leftarrow (B \cap (A - C)) \mid a \leftarrow B \in P,\ a \leftarrow C \in R\} \end{aligned}$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

$$= \{a \leftarrow (B - C) \mid a \leftarrow B \in P,\ a \leftarrow C \in R\}$$
$$= P \boxminus R. \qquad \square$$

Corollary 46.
For any programs P, Q, R, we have

$$(P \sqcup R) \boxminus Q \subseteq (P \boxminus Q) \sqcup (R \boxminus Q), \tag{79}$$

with equality in case Q is minimalist.

Proof.
Notice first that Q is minimalist iff $Q^{\copyright}$ is minimalist, since body-complement preserves rule heads and neither adds nor removes rules. We can therefore compute

$$(P \sqcup R) \boxminus Q \overset{78}{=} (P \sqcup R) \sqcap Q^{\copyright} \overset{74}{\subseteq} (P \sqcap Q^{\copyright}) \sqcup (R \sqcap Q^{\copyright}) \overset{78}{=} (P \boxminus Q) \sqcup (R \boxminus Q),$$

where the inclusion is an identity by equation (76) in case Q — and hence $Q^{\copyright}$ — is minimalist. $\square$

Observation 47.
The inclusion in equation (79) can in general not be turned into an identity, not even modulo subsumption equivalence. For example, for $P := \{a \leftarrow b\}$ and $R := \{a \leftarrow c\}$ and the non-minimalist program $Q := \{a \leftarrow b,\ a \leftarrow c\}$, we have

$$(P \sqcup R) \boxminus Q = \left\{ \begin{array}{l} a \leftarrow b \\ a \leftarrow c \end{array} \right\} \subsetneq \left\{ \begin{array}{l} \underline{a} \\ a \leftarrow b \\ a \leftarrow c \\ \underline{a \leftarrow b, c} \end{array} \right\} = (P \boxminus Q) \sqcup (R \boxminus Q),$$

and since the program on the right contains the fact a whereas the program on the left contains no fact at all, Equation (10) yields

$$[(P \sqcup R) \boxminus Q]\emptyset = \emptyset \neq \{a\} = [(P \boxminus Q) \sqcup (R \boxminus Q)]\emptyset.$$

Of course, Equation (79) does imply

$$(P \sqcup R) \boxminus Q \lesssim (P \boxminus Q) \sqcup (R \boxminus Q).$$

Interestingly enough, we can compute the body-complement of P via body-subtraction according to the formula

$$P^{\copyright} = A^{\oplus} \boxminus P.$$

We shall now investigate the interaction between reducts and body-subtraction. We can express the head-reduct in terms of body-subtraction as (compare with equation (47))

$$P \boxminus I = {}^{I}P, \tag{80}$$

for every program P and interpretation I. Moreover, we can simplify the computation of body-subtraction using left reducts as

$$P \boxminus R = {}^{h(R)}P \boxminus {}^{h(P)}R.$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Proposition 48.
The head-reduct is an endomorphism ${}^I\!.\colon(\mathbb{P}_A,\boxminus)\to(\mathbb{P}_A,\boxminus)$ for every interpretation I, that is, for any programs P and R,

$$^I(P\boxminus R)={}^IP\boxminus{}^IR.$$

Proof.

$$^I(P\boxminus R)\overset{47}{=}I\sqcup(P\boxminus R)\overset{78}{=}I\sqcup(P\sqcap R^{\copyright})\overset{77}{=}(I\sqcup P)\sqcap(I\sqcup R^{\copyright})\overset{47}{=}{}^IP\sqcap{}^I(R^{\copyright})$$
$$\overset{73}{=}{}^IP\sqcap({}^IR)^{\copyright}\overset{78}{=}{}^IP\boxminus{}^IR.$$ □

We now turn our attention to the interaction between composition and body-subtraction.

Observation 49.
There is a minimalist Krom program K and an interpretation I such that

$$(K\boxminus K)I\not\approx KI\boxminus KI.$$

For example, we have

$$(\{a\leftarrow b\}\boxminus\{a\leftarrow b\})\emptyset=\{a\}\emptyset=\{a\}$$

whereas

$$\{a\leftarrow b\}\emptyset\boxminus\{a\leftarrow b\}\emptyset=\emptyset\boxminus\emptyset=\emptyset.$$

Hence, in general we have

$$(P\boxminus R)Q\not\approx PQ\boxminus RQ.$$

Observation 50.
There are Krom programs K and L and an interpretation I such that

$$K(L\boxminus I)\not\approx KL\boxminus KI.$$

For example, for

$$K:=\left\{\begin{array}{l}a\leftarrow b\\a\leftarrow c\end{array}\right\}\quad\text{and}\quad L:=\{b\leftarrow b\}\quad\text{and}\quad I:=\{c\},$$

we have

$$\left\{\begin{array}{l}a\leftarrow b\\a\leftarrow c\end{array}\right\}(\{b\leftarrow b\}\boxminus\{c\})=\emptyset$$

whereas

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

Hence, in general

$$Q(P\boxminus R)\not\approx QP\boxminus QR.$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Fact 51.
For any programs P and R with $h(P) \subseteq h(R)$,

$$P^\omega \subseteq (P \boxminus R)^\omega. \tag{81}$$

7 Body-symmetric-difference

Recall that the set-theoretic symmetric difference of two programs is given by

$$P \triangle R = (P - R) \cup (R - P).$$

We shall introduce analogous set-like operations on programs in two different ways corresponding to the two sides of the above equation:

Definition 52.
Define the *body-symmetric-difference* of P and R by

$$P \boxtimes R := \{a \leftarrow (B \triangle C) \mid a \leftarrow B \in P,\, a \leftarrow C \in R\},$$

and define the *black-body-symmetric-difference* of P and R by

$$P \blacksquare R := (P \boxminus R) \sqcup (R \boxminus P).$$

The two notions of "symmetric difference" coincide on minimalist programs containing at most one rule for each rule head, that is, for any minimalist programs M and N we clearly have

$$M \boxtimes N = M \blacksquare N.$$

However, for non-minimalist programs, the two notions may differ as is witnessed by the following example:

Example 53.
For

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

we have

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

whereas

$$P \blacksquare R = \left\{ \begin{array}{l} a \\ a \leftarrow b \\ a \leftarrow c \end{array} \right\} \sqcup \left\{ \begin{array}{l} a \\ a \leftarrow b \\ a \leftarrow d \end{array} \right\} = \left\{ \begin{array}{l} a \\ \underline{a \leftarrow b} \\ \underline{a \leftarrow c} \\ \underline{a \leftarrow d} \\ a \leftarrow b,c \\ a \leftarrow b,d \\ a \leftarrow c,d \end{array} \right\}.$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Proposition 54.
For any programs P and R,

$$P \boxtriangle R \subseteq P \blacktriangle R.$$

Proof.

$$\begin{aligned}
P \boxtriangle R &= \{a \leftarrow (B \triangle C) \mid a \leftarrow B \in P,\, a \leftarrow C \in R\} \\
&= \{a \leftarrow ((B-C) \cup (C-B)) \mid a \leftarrow B \in P,\, a \leftarrow C \in R\} \\
&\subseteq \{a \leftarrow ((B-C) \cup (D-E)) \mid a \leftarrow B \in P,\, a \leftarrow C \in R,\, a \leftarrow D \in R,\, a \leftarrow E \in P\} \\
&= (P \boxminus R) \sqcup (R \boxminus P) \\
&= P \blacktriangle R.
\end{aligned}$$

□

8 Body-power-set

This section introduces and studies a set-like operation on programs inspired by the analogous power set operation on sets:

Definition 55.
Define the *body-power-set* of P by

$$P^{\mathfrak{p}} := \{a \leftarrow B \mid a \leftarrow C \in P,\ B \subseteq C\}.$$

For example, we have

$$\emptyset^{\mathfrak{p}} = \emptyset, \quad \{a\}^{\mathfrak{p}} = \{a\}, \quad \left\{ \begin{array}{l} a \\ b \leftarrow a \\ c \leftarrow a, b \end{array} \right\}^{\mathfrak{p}} = \left\{ \begin{array}{l} a \\ b \\ c \\ b \leftarrow a \\ c \leftarrow a \\ c \leftarrow b \\ c \leftarrow a, b \end{array} \right\}.$$

Interestingly, we have

$$P^{\mathfrak{p}} = P \boxminus F_A, \tag{82}$$

where F_A is the full program defined in equation (70).

Fact 56.
The body-power-set operator is an endomorphism on $(\mathbb{P}_A, \cup, \emptyset)$[4] thus satisfying

$$\begin{aligned}
(P \cup R)^{\mathfrak{p}} &= P^{\mathfrak{p}} \cup R^{\mathfrak{p}} \\
\emptyset^{\mathfrak{p}} &= \emptyset.
\end{aligned}$$

[4] Recall that F_A is the full program over A defined in (70).

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Moreover, we have

$$(P \cap R)^{\mathfrak{p}} \subseteq P^{\mathfrak{p}} \cap R^{\mathfrak{p}}$$
$$F_A^{\mathfrak{p}} = F_A.$$

Computing the least model of $P^{\mathfrak{p}}$ is easy as it contains a fact for each rule head of P and therefore

$$(P^{\mathfrak{p}})^{\omega} = h(P).$$

The body-power-set operator is idempotent and the identity on interpretations, that is,

$$(P^{\mathfrak{p}})^{\mathfrak{p}} = P^{\mathfrak{p}} \tag{83}$$
$$I^{\mathfrak{p}} = I. \tag{84}$$

Moreover, it can be written in terms of composition as

$$P^{\mathfrak{p}} \overset{20}{=} PA^*. \tag{85}$$

In particular, for any Krom program K, we can derive that identity as

$$K^{\mathfrak{p}} = K \cup h(K) \overset{11}{=} K \cup KA \overset{15}{=} K(1 \cup A) \overset{20}{=} KA^*.$$

As a consequence, we have

$$(KL)^{\mathfrak{p}} \overset{85}{=} (KL)A^* \overset{14}{=} K(LA^*) \overset{85}{=} KL^{\mathfrak{p}},$$

where the second identity follows by the associativity of composition with respect to Krom programs (recall that composition is in general not associative).

Similarly,

$$(PR)^{\mathfrak{p}} \overset{85}{=} (PR)A^* \approx P(RA^*) \overset{85}{=} PR^{\mathfrak{p}},$$

where we have to invoke Proposition 6 instead of equation (14) as for Krom programs above since composition is in general associative only modulo subsumption equivalence.

An interesting identity is

$$1_A^{\mathfrak{p}} = A^*.$$

For the program $A^{\oplus}$, neutral to body-intersection (55), we have

$$(A^{\oplus})^{\mathfrak{p}} = F_A.$$

Hence, as a consequence of equation (82),

$$P^{\mathfrak{p}} = P \boxminus (A^{\oplus})^{\mathfrak{p}}.$$

We now analyze the body-power-set operator in combination with body-union. The next result shows that the body-power-set operator is a homomorphism for body-union:

Proposition 57.
The body-power-set operator is an endomorphism on the monoid $(\mathbb{P}_A, \sqcup, A)$, that is,

$$(P \sqcup R)^{\mathfrak{p}} = P^{\mathfrak{p}} \sqcup R^{\mathfrak{p}}$$
$$A^{\mathfrak{p}} \overset{84}{=} A.$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Proof.
Since

$$A^* \sqcup A^* \overset{20}{=} (1_A \cup A) \sqcup (1_A \cup A) = 1_A \cup A \overset{20}{=} A^*,$$

we may apply equation (45) and compute

$$(P \sqcup R)^{\mathfrak{p}} \overset{85}{=} (P \sqcup R)A^* \overset{45}{=} PA^* \sqcup RA^* \overset{85}{=} P^{\mathfrak{p}} \sqcup R^{\mathfrak{p}}.$$ □

9 Homomorphisms

This section is concerned with homomorphisms, which are structure-preserving mappings between algebras.

First, we state that interpretations under the usual set operations can be embedded into programs under the set-like operations as follows:

Proposition 58.
The mapping $^{\square} : \mathbb{I}_A \to \mathbb{P}_A$ given by

$$I^{\square} := \{a \leftarrow I \mid a \in A\}$$

is a monomorphism $(\mathbb{I}_A, \cup, \cap, ^c, \emptyset, A) \to (\mathbb{M}_A, \sqcup, \sqcap, ^{©}, A, A^{\oplus})$, that is, $^{\square}$ maps every interpretation $I \in \mathbb{I}_A$ to a minimalist program $I^{\square} \in \mathbb{M}_A$ such that

$$(I \cup J)^{\square} = I^{\square} \sqcup J^{\square} \tag{86}$$
$$(I \cap J)^{\square} = I^{\square} \sqcap J^{\square} \tag{87}$$
$$(I^c)^{\square} = (I^{\square})^{©} \tag{88}$$
$$\emptyset^{\square} = A \tag{89}$$
$$A^{\square} = A^{\oplus}. \tag{90}$$

Proof.
Follows immediately from the definitions. □

Remark 59.
Equations (89) and (90) mean that $\emptyset^{\square}$ and $A^{\square}$ are the neutral elements A and $A^{\oplus}$ with respect to $\sqcup$ and $\sqcap$, respectively.

The following theorem gives us a way to construct homomorphisms bottom-up from single-element interpretations representing single atoms:

Theorem 60.
Let A and A' be alphabets and let $^{\bullet} : \mathbb{I}_A \to \mathbb{I}_{A'}$ be a mapping on interpretations satisfying

$$I^{\bullet} = \bigcup_{a \in I} \{a\}^{\bullet}, \tag{91}$$
$$\{a\}^{\bullet} \neq \emptyset, \quad \text{for every } a \in A, \tag{92}$$
$$\{a\}^{\bullet} \cap \{b\}^{\bullet} = \emptyset, \quad \text{for all } a, b \in A \text{ with } a \neq b. \tag{93}$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

We extend $^\bullet$ to a mapping $\mathbb{P}_A \to \mathbb{P}_{A'}$ on programs as

$$P^\bullet := \{a' \leftarrow B^\bullet \mid a \leftarrow B \in P,\ a' \in \{a\}^\bullet\}.$$

Then

$$^\bullet : (\mathbb{P}_A, \cup, \sqcup, \emptyset) \to (\mathbb{P}_{A'}, \cup, \sqcup, \emptyset)$$

is a homomorphism satisfying

$$P \in \mathbb{I}_A \quad \Leftrightarrow \quad P^\bullet \in \mathbb{I}_{A'},$$

that is, a program $P \in \mathbb{P}_A$ is an interpretation iff $P^\bullet \in \mathbb{P}_{A'}$ is an interpretation.

Proof.
Instantiating equation (91) with $I = \emptyset$ yields $\emptyset^\bullet = \emptyset$, and $^\bullet$ is a homomorphism with respect to $\cup$ since

$$\begin{aligned}(P \cup R)^\bullet &= \{a' \leftarrow B^\bullet \mid a \leftarrow B \in P \cup R,\ a' \in \{a\}^\bullet\} \\ &= \{a' \leftarrow B^\bullet \mid a \leftarrow B \in P,\ a' \in \{a\}^\bullet\} \cup \{a' \leftarrow B^\bullet \mid a \leftarrow B \in R,\ a' \in \{a\}^\bullet\} \\ &= P^\bullet \cup R^\bullet.\end{aligned}$$

To show that $^\bullet$ is a homomorphism with respect to $\sqcup$, we compute

$$\begin{aligned}(P \sqcup R)^\bullet &= \{a \leftarrow (B \cup C) \mid a \leftarrow B \in P,\ a \leftarrow C \in R\}^\bullet \\ &= \{a' \leftarrow (B \cup C)^\bullet \mid a \leftarrow B \in P,\ a \leftarrow C \in R,\ a' \in \{a\}^\bullet\} \\ &\overset{91}{=} \{a' \leftarrow (B^\bullet \cup C^\bullet) \mid a \leftarrow B \in P,\ a \leftarrow C \in R,\ a' \in \{a\}^\bullet\} \\ &\overset{93}{=} P^\bullet \sqcup R^\bullet,\end{aligned}$$

where the last identity holds since $P^\bullet \sqcup R^\bullet$ consists of all rules $a' \leftarrow (D \cup E)$ with $a' \leftarrow D \in P^\bullet$ and $a' \leftarrow E \in R^\bullet$: the inclusion from left to right is immediate from the definition of $^\bullet$, and for the inclusion from right to left notice that by equation (93) the head a' determines the unique atom a satisfying $a' \in \{a\}^\bullet$, which means that $D = B^\bullet$ and $E = C^\bullet$ for rules $a \leftarrow B \in P$ and $a \leftarrow C \in R$ with one and the same head a.

It remains to prove the stated equivalence. Every interpretation $I \in \mathbb{I}_A$ is a program consisting of facts only, which by equation (91) gives

$$\{a \leftarrow \emptyset \mid a \in I\}^\bullet = \{a' \leftarrow \emptyset^\bullet \mid a \in I,\ a' \in \{a\}^\bullet\} = \left\{ a' \leftarrow \emptyset \;\middle|\; a' \in \bigcup_{a \in I} \{a\}^\bullet \right\} \overset{91}{=} I^\bullet,$$

which shows that the extension of $^\bullet$ to programs is consistent with the given mapping on interpretations and, in particular, that $P^\bullet$ is an interpretation whenever P is one. For the other direction, assume that P is not an interpretation and pick a proper rule $a \leftarrow B \in p(P)$ together with an atom $b \in B$. By equation (92), we can pick atoms $a' \in \{a\}^\bullet$ and $b' \in \{b\}^\bullet$, and equation (91) yields $b' \in B^\bullet$. Hence, $a' \leftarrow B^\bullet$ is a proper rule in $P^\bullet$ and therefore $P^\bullet$ is not an interpretation. □

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Example 61.
Let $A := \{a, b\}$ and $A' := \{a', b', c'\}$. Define the mapping $^\bullet$ on interpretations in $\mathbb{I}_A$ by

$$\emptyset^\bullet := \emptyset$$
$$\{a\}^\bullet := \{a', c'\}$$
$$\{b\}^\bullet := \{b'\}.$$

This forces us to put

$$\{a, b\}^\bullet := \{a\}^\bullet \cup \{b\}^\bullet = \{a', b', c'\}.$$

Informally speaking, $^\bullet$ "blows up" the atom a to a', c' and leaves the atom b "unchanged". By Theorem 60, extending $^\bullet$ to programs as described above we automatically get a homomorphism satisfying, for example,

$$\left(\left\{\begin{array}{l} a \\ b \leftarrow a \end{array}\right\} \sqcup \left\{\begin{array}{l} a \\ b \leftarrow b \end{array}\right\}\right)^\bullet = \left\{\begin{array}{l} a \\ b \leftarrow a, b \end{array}\right\}^\bullet$$
$$= \left\{\begin{array}{l} a' \\ c' \\ b' \leftarrow a', b', c' \end{array}\right\}$$
$$= \left\{\begin{array}{l} a' \\ c' \\ b' \leftarrow a', c' \end{array}\right\} \sqcup \left\{\begin{array}{l} a' \\ c' \\ b' \leftarrow b' \end{array}\right\}$$
$$= \left\{\begin{array}{l} a \\ b \leftarrow a \end{array}\right\}^\bullet \sqcup \left\{\begin{array}{l} a \\ b \leftarrow b \end{array}\right\}^\bullet.$$

10 Decomposition

This section is concerned with syntactic and semantic decompositions of programs.

Our first observation is as follows:

Fact 62.
We can syntactically decompose every program P into single-rule Krom programs as

$$P = f(P) \cup \bigcup_{a \leftarrow B \in p(P)} \bigsqcup_{b \in B} \{a \leftarrow b\}. \tag{94}$$

For example,

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

In what follows, we are interested in decomposing programs with respect to body-union.

We shall now show that composition can be represented via $\sqcup$, $\cup$, f, and p which is useful given that $\sqcup$ and $\cup$ satisfy nice algebraic laws (see Theorems 11 and 14):

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Theorem 63.
For any programs P and R,

$$PR = f(P) \cup \bigcup_{a \leftarrow B \in p(P)} \bigsqcup_{b \in B} \bigcup_{b \leftarrow C \in R} \bigsqcup_{c \in C} \{a \leftarrow c\}, \tag{95}$$

where we use the convention $\bigsqcup_{c \in C} \{a \leftarrow c\} = \{a \leftarrow C\}$ for every atom a and every set of atoms C, so that the empty body-union of rules with head a is the fact a.

Proof.

$$\begin{aligned}
PR &= f(P) \cup \bigcup_{a \leftarrow B \in p(P)} \{a \leftarrow B\}R \\
&= f(P) \cup \bigcup_{a \leftarrow B \in p(P)} \bigsqcup_{b \in B} \{a \leftarrow b\}R \\
&\overset{15}{=} f(P) \cup \bigcup_{a \leftarrow B \in p(P)} \bigsqcup_{b \in B} \bigcup_{c \leftarrow C \in R} \{a \leftarrow b\}\{c \leftarrow C\} \\
&= f(P) \cup \bigcup_{a \leftarrow B \in p(P)} \bigsqcup_{b \in B} \bigcup_{b \leftarrow C \in R} \{a \leftarrow C\} \\
&= f(P) \cup \bigcup_{a \leftarrow B \in p(P)} \bigsqcup_{b \in B} \bigcup_{b \leftarrow C \in R} \bigsqcup_{c \in C} \{a \leftarrow c\},
\end{aligned}$$

where the first identity uses $f(P)R = f(P)$ together with equation (9). For the second identity, we first write $\{a \leftarrow B\} = \bigsqcup_{b \in B} \{a \leftarrow b\}$ and then apply equation (45) iteratively. This is legitimate since the bodies $\{b\}$, $b \in B$, are pairwise disjoint and therefore satisfy case (i). □

Recall from equation (15) that Krom programs have the nice property that composition distributes over union from both sides which is not the case in general (see equation (12)). Decomposing programs into Krom programs is thus desirable. We shall now prove the main results of the paper showing that every program can be $\sqcup$-decomposed into Krom programs (Theorems 66 and 68), but before we can give the formal proofs we first introduce some auxiliary notions and then illustrate the main idea with an example:

Definition 64.
Define the *body-or* of P as

$$P^{\vee} := f(P) \cup \{a \leftarrow b \mid a \leftarrow B \in p(P),\ b \in B\}.$$

Notice that the body-or of a program is always a Krom program consisting only of rules with at most one body atom. For example, we have

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

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Example 65.
Now consider the minimalist program

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

We show that we can decompose M with width $w(M) = 3$ into three Krom programs K_1, K_2, K_3, where

$$K_1, K_2, K_3 \subseteq M^\vee = \left\{ \begin{array}{l} a \\ b \\ c \leftarrow a \\ c \leftarrow b \\ d \leftarrow a \\ d \leftarrow b \\ d \leftarrow d \end{array} \right\},$$

such that

$$M = K_1 \sqcup K_2 \sqcup K_3 \tag{96}$$

and

$$M^\omega = K_1^\omega \sqcup K_2^\omega \sqcup K_3^\omega.$$

For this, define

$$K_1 := \left\{ \begin{array}{l} a \\ b \\ c \leftarrow a \\ d \leftarrow a \end{array} \right\}, \quad K_2 := \left\{ \begin{array}{l} a \\ b \\ c \leftarrow b \\ d \leftarrow b \end{array} \right\}, \quad K_3 := \left\{ \begin{array}{l} a \\ b \\ c \leftarrow b \\ d \leftarrow d \end{array} \right\}.$$

We have equation (96) and

$$M^\omega = \{a, b, c\} = \{a, b, c, d\} \cap \{a, b, c, d\} \cap \{a, b, c\} = K_1^\omega \cap K_2^\omega \cap K_3^\omega \overset{48}{=} K_1^\omega \sqcup K_2^\omega \sqcup K_3^\omega.$$

We are now ready to prove the following decomposition theorem for minimalist programs containing at most one rule for each rule head by generalizing the reasoning in Example 65:

Theorem 66 (Minimalist decomposition theorem).
Let M be a minimalist program and put

$$m := \max\{1, w(M)\}.$$

Then there are Krom programs

$$K_1, \ldots, K_m \subseteq M^\vee$$

such that

$$M = K_1 \sqcup \ldots \sqcup K_m \tag{97}$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

and

$$M^\omega = K_1^\omega \sqcup \ldots \sqcup K_m^\omega \overset{48}{=} K_1^\omega \cap \cdots \cap K_m^\omega. \tag{98}$$

Proof.
Since M is minimalist, for every $a \in h(M)$ there is a unique rule

$$r_a = a \leftarrow B_a$$

in M.

We first choose, for every non-empty body B_a, a distinguished atom $b_a^* \in B_a$. If $a \notin M^\omega$, choose

$$b_a^* \in B_a - M^\omega.$$

Such an atom exists. Indeed, if $B_a \subseteq M^\omega$, then

$$a \in MM^\omega = M^\omega,$$

contrary to $a \notin M^\omega$. If $a \in M^\omega$, choose b_a^* arbitrarily from B_a.

Since $|B_a| \leq m$, for every non-empty B_a we can choose atoms

$$b_{a,1}, \ldots, b_{a,m} \in B_a$$

such that

$$b_{a,1} = b_a^* \quad \text{and} \quad B_a = \{b_{a,1}, \ldots, b_{a,m}\}.$$

If $|B_a| < m$, the remaining positions are filled by repeating arbitrary atoms of B_a.

For $1 \leq i \leq m$, define

$$K_i := f(M) \cup \{a \leftarrow b_{a,i} \mid a \leftarrow B_a \in p(M)\}.$$

Every K_i is a Krom program. Moreover, every rule $a \leftarrow b_{a,i}$ belongs to $M^\vee$, and hence

$$K_i \subseteq M^\vee.$$

For every $a \in h(M)$, the unique rules with head a in the programs $K_1, \ldots, K_m$ have bodies whose union is B_a. Consequently,

$$K_1 \sqcup \ldots \sqcup K_m = \{a \leftarrow B_a \mid a \in h(M)\} = M.$$

This proves equation (97).

We next compare the least models. For every interpretation I and every i, we have

$$MI \subseteq K_i I.$$

Indeed, whenever $B_a \subseteq I$, we also have $b_{a,i} \in I$. Facts are present in both M and K_i. By induction on the iterations of the immediate consequence operators, it follows that

$$M^\omega \subseteq K_i^\omega \qquad (1 \leq i \leq m).$$

It remains to show that the first component introduces no new atoms. We claim that M^ω is a model of K_1. Consider a rule in K_1. If it is a fact, then its head belongs to $f(M) \subseteq M^\omega$. Otherwise, it has the form

$$a \leftarrow b_{a,1}.$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

If $a \in M^\omega$, this rule is satisfied by M^ω. If $a \notin M^\omega$, then the construction of $b_{a,1}$ gives

$$b_{a,1} = b_a^* \notin M^\omega,$$

so the body of the rule is false in M^ω. Thus every rule of K_1 is satisfied by M^ω, and therefore

$$K_1^\omega \subseteq M^\omega.$$

Together with the opposite inclusion proved above, this yields

$$K_1^\omega = M^\omega.$$

Finally,

$$M^\omega = K_1^\omega \subseteq K_i^\omega \qquad (1 \leq i \leq m),$$

and hence

$$K_1^\omega \cap \ldots \cap K_m^\omega = M^\omega.$$

Since body-union coincides with intersection on interpretations by equation (48), we obtain

$$K_1^\omega \sqcup \ldots \sqcup K_m^\omega = K_1^\omega \cap \ldots \cap K_m^\omega = M^\omega.$$

This proves equation (98). □

Remark 67.
The decomposition in equation (97) is not unique. The condition in equation (97) is essential since without it, we can simply choose M^ω — which is an interpretation and thus trivially Krom — as our "decomposition."

Theorem 68 (Decomposition theorem).
Let P be a program and put

$$m := \max\{1, w(P)\}.$$

Then there are Krom programs

$$K_1, \ldots, K_m \subseteq P^\vee$$

such that

$$P \subseteq K_1 \sqcup \cdots \sqcup K_m \tag{99}$$

and

$$P^\omega \subseteq K_1^\omega \sqcup \cdots \sqcup K_m^\omega \overset{48}{=} K_1^\omega \cap \cdots \cap K_m^\omega. \tag{100}$$

Proof.
Since $|B| \leq m$ holds for every rule $a \leftarrow B \in p(P)$, we can choose for every such rule $r = a \leftarrow B$ atoms

$$b_{r,1}, \ldots, b_{r,m} \in B \quad \text{such that} \quad B = \{b_{r,1}, \ldots, b_{r,m}\},$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

where in case $|B| < m$ the remaining positions are filled by repeating arbitrary atoms of B. For $1 \leq i \leq m$, define

$$K_i := f(P) \cup \{a \leftarrow b_{r,i} \mid r = a \leftarrow B \in p(P)\}.$$

Every K_i is a Krom program and every one of its rules occurs in $P^\vee$, which gives $K_i \subseteq P^\vee$.

We first prove equation (99). Let $a \leftarrow B \in P$. In case $B = \emptyset$, we have $a \in f(P) \subseteq K_i$ for every i and hence $a \in K_1 \sqcup \cdots \sqcup K_m$. In case $r = a \leftarrow B \in p(P)$, the rule $a \leftarrow b_{r,i}$ belongs to K_i for every i, and the body-union of these m rules is

$$a \leftarrow (\{b_{r,1}\} \cup \ldots \cup \{b_{r,m}\}) = a \leftarrow B,$$

which shows $a \leftarrow B \in K_1 \sqcup \cdots \sqcup K_m$. Notice that in contrast to Theorem 66 we cannot expect equality here, since the body-union also combines rules originating from *different* rules of P with the same head, thus producing rules not occurring in P (see Example 69).

We now prove equation (100). Let I be an arbitrary interpretation and let $a \in PI$, that is, there is a rule $a \leftarrow B \in P$ with $B \subseteq I$. In case $B = \emptyset$, we have $a \in f(P) \subseteq K_i$ and hence $a \in K_i I$; and in case $r = a \leftarrow B \in p(P)$, we have $b_{r,i} \in B \subseteq I$ and therefore $a \in K_i I$ as well. This shows

$$P \lesssim K_i \quad (1 \leq i \leq m),$$

which by equations (17) and (22) yields $P^\omega \subseteq K_i^\omega$ for every i and hence

$$P^\omega \subseteq K_1^\omega \cap \cdots \cap K_m^\omega \overset{48}{=} K_1^\omega \sqcup \cdots \sqcup K_m^\omega. \qquad \square$$

The next example provides an intuition why we cannot turn the inclusion in Theorem 68 into an equality:

Example 69.
The non-minimalist program

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

cannot be $\sqcup$-decomposed into Krom programs. Consider, for instance,

$$\left\{ \begin{array}{l} a \leftarrow b \\ a \leftarrow d \end{array} \right\} \sqcup \left\{ \begin{array}{l} a \leftarrow c \\ a \leftarrow e \end{array} \right\} = \left\{ \begin{array}{l} a \leftarrow b, c \\ a \leftarrow \underline{b, e} \\ a \leftarrow \underline{d, c} \\ a \leftarrow d, e \end{array} \right\},$$

where the body-union produces the two underlined rules not occurring in P. This is not an artifact of the particular choice. Suppose $P = K_1 \sqcup \ldots \sqcup K_n$ for Krom programs $K_1, \ldots, K_n$. Since every rule in P has head a, each K_i contains at least one rule with head a, and by the definition of body-union there are bodies $B_1, \ldots, B_n$ and $C_1, \ldots, C_n$ of rules with head a in $K_1, \ldots, K_n$ satisfying

$$B_1 \cup \ldots \cup B_n = \{b, c\} \quad \text{and} \quad C_1 \cup \ldots \cup C_n = \{d, e\}.$$

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

As each B_i contains at most one atom, we have $B_i = \{b\}$ for some i, and replacing C_i by B_i in the second selection yields a rule in $K_1 \sqcup \ldots \sqcup K_n = P$ whose body contains b and is contained in $\{b, d, e\}$ — contradicting the fact that the only rule in P with b occurring in its body is $a \leftarrow b, c$.

11 Towards applications to automatic logic programming

Automatic logic programming studies the construction of logic programs from specifications and background knowledge (Deville and Lau 1994). A key challenge in this setting is to provide principled mechanisms for combining and transforming programs while preserving their semantics.

In this section, we sketch how the set-like operations introduced in this paper can contribute to this goal by enabling structured manipulations of programs and supporting decomposition-based approaches to program construction.

Beyond these examples, algebraic operations on logic programs are also relevant to program transformation and optimization (Fuchs and Fromherz 1992; Partsch and Steinbrüggen 1983).

11.1 Algebraic semantics

A central requirement for algebraic reasoning about logic programs is the ability to decompose complex programs into simpler components in such a way that their semantics can be derived from the semantics of these components.

Theorems 66 and 68 provide two related structural results: minimalist programs admit exact representations in terms of Krom components, whereas arbitrary programs admit corresponding syntactic and semantic upper approximations. In the minimalist case, the least model is the intersection of the least models of the components.

Krom programs satisfy algebraic laws such as associativity equation (14) and left-distributivity equation (15), which fail for arbitrary programs. The decomposition results therefore provide structural representations in terms of algebraically simpler components. Since the construction used in Theorem 66 depends on the least model of the original program, however, the theorem does not by itself yield an algorithmic reduction of reasoning to the components.

11.2 Schema-based transformation

Program schemes provide a common mechanism for representing reusable program fragments that can be instantiated to obtain concrete programs (Greibach 1975; Flener 1995; Fuchs and Fromherz 1992).

Within an algebraic setting, such schemes can be naturally understood as terms (or polynomials) over logic programs, where programs act as constants, algebraic operations as function symbols, and program variables as placeholders (see e.g. Burris and Sankappanavar (1981), Section II.1). For example, the term $\{a \leftarrow b\} \sqcup X$ defines a schema that can be instantiated with a program P to yield $\{a \leftarrow b\} \sqcup P$, thus inducing a transformation on programs.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Crucially, the expressive power of such schemes depends on the underlying algebra of programs. Introducing new algebraic operations expands the space of definable transformations. In particular, the set-like operations considered in this paper enable schemes that are not expressible in a natural way without them, such as those based on body-union.

11.3 Inductive logic programming

In ILP (Shapiro 1981; Muggleton 1991; Nienhuys-Cheng and de Wolf 1997; Cropper and Dumančić 2022), (partial) specifications consist of sets of positive and negative examples in the form of interpretations I_+ and I_-, respectively, and the task is, given some background program R, to find a program P such that

$$I_+ \subseteq (P \cup R)^\omega \quad \text{and} \quad I_- \cap (P \cup R)^\omega = \emptyset.$$

From an algebraic perspective, this task amounts to searching the space of programs under the least model operator and its interaction with program composition. A key difficulty is that this search space is typically large and poorly structured.

Algebraic operations on programs provide a way to impose structure on this space. In particular, the decomposition results established in Theorems 66 and 68 allow one to reduce the search for a solution to reasoning about simpler components, since they make explicit how the behavior of a program emerges from the behavior of its parts.

11.4 Learning from interpretation transitions

Closely related to ILP Section 11.3 is the following learning setting introduced in Inoue *et al.* (2014).

Every logic program P is a state transition system that maps, via its immediate consequence van Emden-Kowalski operator T_P, every state of the world represented as an interpretation I to a next state $T_P(I)$ (Inoue 2011; Inoue and Sakama 2012).

The learning setting now is as follows. The positive examples are given by pairs of interpretations (I, J) and the task is to find a program P such that $T_P(I) = J$ holds for each such pair. Notice that by equation (16), this is equivalent to finding a program P satisfying $PI = J$ via sequential composition of P with I.

We are thus interested in the transition behavior of programs with respect to the composition with interpretations and in this paper, we have derived numerous results in that direction. In fact, every result showing identities (modulo subsumption equivalence) between programs can be instantiated with interpretations to yield results concerning the behavior of programs with respect to interpretations. For example, Theorem 17 shows

$$(P \sqcup R)I = PI \sqcup RI \overset{48}{=} PI \cap RI,$$

which means that if we have programs P and R, then $P \sqcup R$ is a solution to the state transition $(I, PI \cap RI)$. Here the benefit of representing the solution program as the body-union of its two factors P and R is that it provides a better understanding of the inner workings of the program.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

11.5 Analogical logic programming

In analogical logic programming — recently introduced in Antić (2025) — the task is as follows.

Given a proportional equation of the form $P : Q :: R : X$ — read as "P is to Q what R is to X" — where P, Q, R are programs and X is a program variable, we wish to find solutions $S_1, S_2, \ldots$ such that $P : Q :: R : S_i$ for all i, in which case we say that P, Q, R, S_i are in an analogical proportion.[5] That is, we generate novel programs from "known" ones via analogical transfer instead of starting with a partial specification as in ILP Section 11.3.

Analogical proportions formalize the idea that analogy-making is the task of transforming different objects from the source to the target domain in "the same way" (which is why "copycat" is the name of a prominent model of analogy-making (Hofstadter and Mitchell 1995); and see Correa *et al.* (2012)).

At the formal level, we instantiate a recently introduced abstract algebraic framework of analogical proportions (Antić 2022) in our setting of logic programming, where we will restrict ourselves to the functorial fragment (Antić 2025):[6]

Definition 70 (Logic program proportions).
Let P, Q, R, S be programs. We say that P *transforms into* Q *as* R *transforms into* S — in symbols, $P \to Q :: R \to S$ — if there is a schema $\mathbf{F}(X)$ such that $Q = \mathbf{F}(P)$ and $S = \mathbf{F}(R)$, in which case we call $\mathbf{F}$ a *justification* of $P \to Q :: R \to S$. We call an expression of the form $P \to Q :: R \to S$ an *arrow proportion*.

We now say that P *is to* Q *as* R *is to* S — in symbols, $P : Q :: R : S$ — if $P \to Q :: R \to S$ and $Q \to P :: S \to R$, that is, if there are schemes $\mathbf{F}$ and $\mathbf{G}$ such that $Q = \mathbf{F}(P)$, $S = \mathbf{F}(R)$, $P = \mathbf{G}(Q)$, and $R = \mathbf{G}(S)$, in which case we call the pair $(\mathbf{F}, \mathbf{G})$ a *justification* of $P : Q :: R : S$. We call an expression of the form $P : Q :: R : S$ a *(logic program) proportion*.

Since Definition 70 uses program schemes, the underlying algebra of programs is essential and, importantly, the set-like operations introduced in this paper allow us to derive (plausible) proportions which would otherwise not be expressible.

We shall now illustrate logic program proportions with some simple examples:

Example 71.
We have the proportion

$$\{a\} : \{a, b\} :: \{c\} : \{c, b\}$$

between interpretations as justified by the schemes

$$\mathbf{F}(X) := X \cup \{b\} \quad \text{and} \quad \mathbf{G}(X) := X - \{b\}.$$

Example 72.
The proportion

$$\{a\} : \{a, b\} :: \{c\} : \{c\}$$

[5] Analogical proportions between logic programs can be seen as analogical proportions between formulas (Herzig *et al.* 2024).
[6] The abstract framework has been studied in 2-valued boolean (Antić 2024b) and monounary algebras (Antić 2024a), where analogical proportions admit particularly transparent structural descriptions.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

is justified by

$$\mathbf{F}(X) := \left\{ \begin{array}{l} a \leftarrow a \\ b \leftarrow a \\ c \leftarrow c \end{array} \right\} X \quad \text{and} \quad \mathbf{G}(X) := \left\{ \begin{array}{l} a \leftarrow a \\ a \leftarrow b \\ c \leftarrow c \end{array} \right\} X.$$

Proposition 73.
For any programs P and R,

$$P : P^{\copyright} :: R : R^{\copyright}. \tag{101}$$

Proof.
An immediate consequence of equation (59). □

Example 74.
Let $A := \{a, b, c\}$. As a consequence of equation (101) we have, for example,

$$\left\{ \begin{array}{l} a \leftarrow b \\ b \leftarrow a \\ c \leftarrow c \end{array} \right\} : \left\{ \begin{array}{l} a \leftarrow a, c \\ b \leftarrow b, c \\ c \leftarrow a, b \end{array} \right\} :: \left\{ \begin{array}{l} a \leftarrow a \\ b \leftarrow b \\ c \leftarrow c \end{array} \right\} : \left\{ \begin{array}{l} a \leftarrow b, c \\ b \leftarrow a, c \\ c \leftarrow a, b \end{array} \right\} \tag{102}$$

which can be written more concisely as follows.

Every permutation π of A gives rise to a *permutation program* over A — again denoted by π — via

$$\pi := \{\pi(a) \leftarrow a \mid a \in A\}. \tag{103}$$

We will use the well-known cycle notation.[7] For example, the permutation $\pi_{(a\,b\,c)}$, which is defined as the mapping $a \mapsto b$, $b \mapsto c$, $c \mapsto a$, gives rise to the permutation program

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

Notice that every permutation program is minimalist and Krom. In the cycle notation used below, cycles of length one are included, so that the atoms occurring in the notation are exactly the elements of the underlying alphabet.

Moreover, define the *diagonal program* over A as the minimalist program

$$\Delta_A := \{a \leftarrow (A - \{a\}) \mid a \in A\}. \tag{104}$$

For example, we have

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

We can now rewrite equation (102) more compactly as

$$\pi_{(a\,b)(c)} : \pi^{\copyright}_{(a\,b)(c)} :: 1_A : \Delta_A,$$

where $\pi_{(a\,b)(c)}$ is a permutation program, 1_A is a unit program defined in equation (6), and Δ_A is a diagonal program.

[7] https://en.wikipedia.org/wiki/Permutation#Cycle_notation

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Proposition 75.
Let P, Q, R be programs such that Q is minimalist and

(i) $B \cap C = \emptyset$, for all $a \leftarrow B \in P$ and $a \leftarrow C \in Q$;
(ii) $D \cap C = \emptyset$, for all $a \leftarrow D \in R$ and $a \leftarrow C \in Q$;
(iii) $h(P) \cup h(R) \subseteq h(Q)$.

Then

$$P : P \sqcup Q :: R : R \sqcup Q.$$

Proof.
Since Q is minimalist, for every $a \in h(Q)$ there is a unique rule $a \leftarrow C_a \in Q$. We show $(P \sqcup Q) \boxminus Q = P$; the identity $(R \sqcup Q) \boxminus Q = R$ is proved analogously using (ii) instead of (i). By (iii), every rule head of P occurs in Q, which by minimalism of Q gives

$$P \sqcup Q = \{a \leftarrow (B \cup C_a) \mid a \leftarrow B \in P\}$$

and therefore, again by minimalism of Q,

$$(P \sqcup Q) \boxminus Q = \{a \leftarrow ((B \cup C_a) - C_a) \mid a \leftarrow B \in P\} = \{a \leftarrow (B - C_a) \mid a \leftarrow B \in P\} = P,$$

where the last identity holds by (i). Consequently, the schemes

$$\mathbf{F}(X) := X \sqcup Q \quad \text{and} \quad \mathbf{G}(X) := X \boxminus Q$$

satisfy $\mathbf{F}(P) = P \sqcup Q$, $\mathbf{F}(R) = R \sqcup Q$, $\mathbf{G}(P \sqcup Q) = P$, and $\mathbf{G}(R \sqcup Q) = R$, which by Definition 70 means that $(\mathbf{F}, \mathbf{G})$ is a justification of the stated proportion. □

Example 76.
As a consequence of Proposition 75, we have

$$\{a\} : \{a \leftarrow c\} :: \{b\} : \{b \leftarrow c\},$$

justified by

$$\mathbf{F}(X) := X \sqcup \left\{ \begin{array}{l} a \leftarrow c \\ b \leftarrow c \end{array} \right\} \quad \text{and} \quad \mathbf{G}(X) := X \boxminus \left\{ \begin{array}{l} a \leftarrow c \\ b \leftarrow c \end{array} \right\}.$$

12 Conclusion

In this paper, we introduced a family of algebraic operations on propositional Horn logic programs that generalize classical set-theoretic constructions, including union, intersection, complement, subtraction, symmetric difference, and power set (see Sections 3–8). While inspired by set theory, these operations exhibit fundamentally different behavior due to the rule-based structure of logic programs, in particular through non-idempotence and non-trivial interactions between rules sharing the same head.

These operations provide a structured means of decomposing and recomposing logic programs, which allowed us to establish the Decomposition Theorems 66 and 68. As illustrated in Section 11, they offer a basis for approaching problems in automatic logic programming from an algebraic perspective, including program construction, transformation, and analysis.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

A natural direction for future work is the extension of this framework beyond the propositional Horn setting. In particular, lifting these operations to first-order and higher-order logic programs (Lloyd 1987; Apt 1990; Chen *et al*. 1993; Miller and Nadathur 2012), as well as to answer set programs with negation as failure (Gelfond and Lifschitz 1991; Marek and Truszczyński 1999; Brewka *et al*. 2011; Lifschitz 2019), raises substantial technical challenges. For instance, the definition of a body-complement operation in the first-order setting is non-trivial, as rule bodies are required to be finite, preventing a direct generalization of the propositional case. Developing suitable generalizations that preserve the algebraic structure identified in this paper remains an open problem.

More broadly, the results highlight an algebraic structure of the class of logic programs considered here: minimalist programs can be decomposed into and constructed from Krom components while preserving their least-model semantics, whereas arbitrary programs admit corresponding syntactic and semantic approximations. In this respect, the results complement earlier algebraic approaches to logic programming and may support the further development of compositional methods for program synthesis and transformation.

Data availability statement

The manuscript has no data associated.

Competing interests

The author declares none.

Use of artificial intelligence tools

OpenAI Codex (GPT-5, accessed on 4 August 2026, https://openai.com/codex/) was used to analyze an earlier proof of the Minimalist Decomposition Theorem 66, identify a gap in that argument, and assist in drafting a corrected proof.

Anthropic's Claude (Opus 5, accessed on 7 August 2026, https://claude.ai) was subsequently used to check the manuscript for mathematical and expository errors. It assisted in testing algebraic identities, identifying counterexamples, missing hypotheses, and an omitted empty-body case, and in drafting and verifying corrected statements, proofs, and examples. This included the results on distributivity and subsumption equivalence in Theorems 17, 41, and Lemma 19, as well as the Decomposition Theorems 66 and 68.

Both systems also assisted in revising the abstract and the introduction. The author reviewed and independently verified all resulting statements and arguments and takes full responsibility for the correctness and content of the manuscript.

References

Antić, C. 2014. On cascade products of answer set programs. *Theory and Practice of Logic Programming* 14, 4–5, 711–723. https://doi.org/10.1017/S1471068414000301.

Antić, C. 2022. Analogical proportions. *Annals of Mathematics and Artificial Intelligence* 90, 6, 595–644. https://doi.org/10.1007/s10472-022-09798-y.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Antić, C. 2024a. Analogical proportions in monounary algebras. *Annals of Mathematics and Artificial Intelligence* 92, 1663–1677. https://doi.org/10.1007/s10472-023-09921-7.

Antić, C. 2024b. Boolean proportions. *Logical Methods in Computer Science* 20, 2, 2:1–2:20. https://doi.org/10.46298/lmcs-20(2:2)2024.

Antić, C. 2024c. Sequential composition of answer set programs. https://arxiv.org/pdf/2104.12156.pdf

Antić, C. 2024d. Sequential composition of propositional logic programs. *Annals of Mathematics and Artificial Intelligence* 92, 2, 505–533. https://doi.org/10.1007/s10472-024-09925-x.

Antić, C. 2025. Logic program proportions. *Annals of Mathematics and Artificial Intelligence* 93, 321–342. https://doi.org/10.1007/s10472-023-09904-8.

Apt, K. R. 1990. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B, Elsevier, Amsterdam, 493–574.

Apt, K. R. 1997. *From Logic Programming to Prolog*. C.A.R. Hoare Series. Prentice Hall, Englewood Cliffs, NJ.

Backus, J. 1978. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the ACM* 21, 8, 613–641.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge.

Bossi, A., Bugliesi, M., Gabbrielli, M., Levi, G. and Meo, M. 1996. Differential logic programs: Programming methodologies and semantics. *Science of Computer Programming* 27, 217–262.

Brewka, G., Eiter, T. and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54, 12, 92–103.

Brogi, A., Lamma, E. and Mello, P. 1992a. Compositional model-theoretic semantics for logic programs. *New Generation Computing* 11, 1–21.

Brogi, A., Mancarella, P., Pedreschi, D. and Turini, F. 1990. Composition operators for logic theories. In *Computational Logic, Symposium Proceedings 1990*, Springer-Verlag, New York, 117–134.

Brogi, A., Mancarella, P., Pedreschi, D. and Turini, F. 1992b. Meta for modularising logic programming. In *META 1992*, A. Pettorossi, Ed. Vol. 649, Springer-Verlag, Berlin, Heidelberg, 105–119.

Brogi, A., Mancarella, P., Pedreschi, D. and Turini, F. 1994. Modular logic programming. *ACM Transactions on Programming Languages and Systems* 16, 4, 1361–1398.

Brogi, A. and Turini, F. 1995. Fully abstract compositional semantics for an algebra of logic programs. *Theoretical Computer Science* 149, 2, 201–229.

Bugliesi, M., Lamma, E. and Mello, P. 1994. Modularity in logic programming. *The Journal of Logic Programming* 19–20, 1, 443–502.

Burris, S. and Sankappanavar, H. P. 1981. *A Course in Universal Algebra*. Graduate Texts in Mathematics, vol. 78. Springer-Verlag, New York.

Ceri, S., Gottlob, G. and Tanca, L. 1989. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering* 1, 1, 146–166.

Ceri, S., Gottlob, G. and Tanca, L. 1990. *Logic Programming and Databases*. Surveys in Computer Science. Springer-Verlag, Berlin/Heidelberg.

Chen, W., Kifer, M. and Warren, D. S. 1993. HiLog: A foundation for higher-order logic programming. *The Journal of Logic Programming* 15, 3, 187–230.

Correa, W., Prade, H. and Richard, G. 2012. When intelligence is just a matter of copying. In *ECAI 2012*, L. D. Raedt, C. Bessiere, D. Dubois, P. Doherty, P. Frasconi, F. Heintz and

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

P. Lucas, Eds. Vol. 242 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, Amsterdam, 276–281.

Cropper, A. and Dumančić, S. 2022. Inductive logic programming at 30: A new introduction. *Journal of Artificial Intelligence Research* 74, 765–850.

Deville, Y. and Lau, K.-K. 1994. Logic program synthesis. *The Journal of Logic Programming* 19–20, 1, 321–350.

Dong, G. 1988. On the Composition and Decomposition of Datalog Program Mappings. Ph.D. thesis, University of Southern California, Los Angeles, CA.

Dong, G. and Ginsburg, S. 1990. On the decomposition of datalog program mappings. *Theoretical Computer Science* 76, 1, 143–177.

Droste, M., Stüber, T. and Vogler, H. 2010. Weighted finite automata over strong bimonoids. *Information Sciences* 180, 1, 156–166.

Faber, W., Pfeifer, G. and Leone, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence* 175, 1, 278–298.

Flener, P. 1995. *Logic Program Synthesis from Incomplete Information*. Kluwer.

Fuchs, N. E. and Fromherz, M. P. J. 1992. Schema-based transformations of logic programs. In *LOPSTR 1991*, T. P. Clement and K.-K. Lau, Eds. Springer-Verlag, London, 111–125.

Gaifman, H. and Shapiro, E. 1989. Fully abstract compositional semantics for logic programs. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages*, ACM Press, 134–142.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, 3–4, 365–385.

Golan, J. S. 1999. *Semirings and Their Applications*. Springer Science+Business Media.

Greibach, S. A. 1975. *Theory of Program Structures: Schemes, Semantics, Verification*. Springer-Verlag, Berlin/Heidelberg.

Herzig, A., Lorini, E. and Prade, H. 2024. A novel view of analogical proportion between formulas. In *ECAI 2024*, IOS Press, Amsterdam, 1270–1277.

Hill, P. and Lloyd, J. W. 1994. *The Gödel Programming Language*. The MIT Press.

Hirani, A. N. and Subrahmanian, V. S. 1990. Algebraic foundations of logic programming, I: The distributive lattice of logic programs. *Fundamenta Informaticae* 13, 3, 317–332.

Hodges, W. 1994. Logical features of Horn clauses. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. M. Gabbay, C. J. Hogger and J. A. Robinson, Eds. Vol. 1. Clarendon Press, Oxford/NY, 449–503.

Hofstadter, D. and Mitchell, M. 1995. The copycat project: A model of mental fluidity and analogy-making. In *Fluid Concepts and Creative Analogies. Computer Models of the Fundamental Mechanisms of Thought*. Basic Books, NY, 205–267.

Howie, J. M. 2003. *Fundamentals of Semigroup Theory*. London Mathematical Society Monographs New Series. Oxford University Press, Oxford.

Inoue, K. 2011. Logic programming for Boolean networks. In *IJCAI 2011*. AAAI Press, 924–930.

Inoue, K., Ribeiro, T. and Sakama, C. 2014. Learning from interpretation transition. *Machine Learning* 94, 51–79. https://link.springer.com/content/pdf/10.1007/s10994-013-5353-8.pdf

Inoue, K. and Sakama, C. 2012. Oscillating behavior of logic programs. In *Correct Reasoning — Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, E. Erdem, J. Lee, Y. Lierler and D. Pearce, Eds. Springer, Berlin.

Ioannidis, Y. E. and Wong, E. 1991. Towards an algebraic theory of recursion. *Journal of the ACM* 38, 2, 329–381.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696

Krom, M. R. 1967. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly* 13, 1–2, 15–20.

Lifschitz, V. 2019. *Answer Set Programming*. Springer Nature Switzerland AG, Cham, Switzerland.

Lifschitz, V. and Turner, H. 1994. Splitting a logic program. In *Proceedings 11th International Conference on Logic Programming 1994*, P. Van Hentenryck, Ed. MIT Press, Cambridge MA, 23–37.

Lloyd, J. W. 1987. *Foundations of Logic Programming*, 2nd ed. Springer-Verlag, Berlin, Heidelberg.

Maher, M. J. 1988. Equivalences of logic programs. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed., chapter 16, Morgan Kaufmann Publishers, 627–658.

Mancarella, P. and Pedreschi, D. 1988. An algebra of logic programs. In *Proceedings of the 5th International Conference on Logic Programming 1988*, R. Kowalski and K. A. Bowen, Eds. The MIT Press, Cambridge MA, 1006–1023.

Marek, V. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective*, K. R. Apt, V. Marek, M. Truszczyński and D. S. Warren, Eds. Springer, Berlin, 375–398.

Miller, D. and Nadathur, G. 2012. *Programming with Higher-Order Logic*. Cambridge University Press.

Muggleton, S. 1991. Inductive logic programming. *New Generation Computing* 8, 4, 295–318.

Nienhuys-Cheng, S.-H. and de Wolf, R. 1997. Foundations of inductive logic programming. In *LNAI 1228*. Springer-Verlag, Berlin/Heidelberg.

O'Keefe, R. A. 1985. Towards an algebra for constructing logic programs. In *Proceedings 2nd IEEE Symposium on Logic Programming 1985*. IEEE Computer Society Press, Boston, MA, 152–160.

Partsch, H. and Steinbrüggen, R. 1983. Program transformation systems. *ACM Computing Surveys* 15, 3, 199–236.

Shapiro, E. 1981. Inductive inference of theories from facts. Technical Report Research Report 192, Yale University.

Sterling, L. and Shapiro, E. 1994. *The Art of Prolog: Advanced Programming Techniques*, 2nd ed. The MIT Press, Cambridge, MA.

van Emden, M. H. and Kowalski, R. 1976. The semantics of predicate logic as a programming language. *Journal of the ACM* 23, 4, 733–742.

van Hoorn, W. G. and van Rootselaar, B. 1967. Fundamental notions in the theory of seminearrings. *Compositio Mathematica* 18, 1–2, 65–78.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 12 Nov 2026 at 08:00:27, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1471068426100696