University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Computer Science

Argumentation mining using neuro-probabilistic answer set programs

Jonas Rodrigues Lima Gonçalves

Final Essay

mac 499 — Capstone Project

Supervisor: Prof. Dr. Denis Deratani Mauá

São Paulo
2025

Ficha catalográfica elaborada com dados inseridos pelo(a) autor(a)
Biblioteca Carlos Benjamin de Lyra
Instituto de Matemática e Estatística
Universidade de São Paulo

Lima Gonçalves, Jonas Rodrigues
Argumentation mining using neuro-probabilistic answer set programs / Jonas Rodrigues Lima Gonçalves; orientador, Denis Maua. - São Paulo, 2025.
98 p.: il.

Dissertação (Mestrado) - Programa de Pós-Graduação em Ciência da Computação / Instituto de Matemática e Estatística / Universidade de São Paulo.
Bibliografia
Versão original

1. Inteligência Artificial. 2. Representação de Conhecimento. 3. Programação Lógico Probabilística. 4. Compilação de Conhecimento. 5. Circuitos Probabilísticos. I. Maua, Denis. II. Título.

Bibliotecárias do Serviço de Informação e Biblioteca
Carlos Benjamin de Lyra do IME-USP, responsáveis pela
estrutura de catalogação da publicação de acordo com a AACR2:
Maria Lúcia Ribeiro CRB-8/2766; Stela do Nascimento Madruga CRB 8/7534.

Acknowledgements

I am profoundly grateful to my advisor, Professor Denis Deratani Mauá, for his invaluable guidance and insightful feedback throughout this work. His mentorship has been instrumental in shaping my academic trajectory and fostering my intellectual growth. I extend my heartfelt thanks to my family, whose unwavering encouragement, patience, and belief in me have been a constant source of strength. I am also deeply appreciative of my friends and colleagues at IME-USP for their camaraderie, stimulating discussions, and the inspiration they have provided along the way. All this support has made this journey not only rewarding but also truly memorable.

Abstract

Jonas Rodrigues Lima Gonçalves. **Argumentation mining using neuro-probabilistic answer set programs**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2025.

Argumentation mining is a complex task that has been approached more recently using purely connectionist methods. These methods aim to extract arguments from text data and represent them in a structured format. However, these methods often lack the ability to handle uncertainty and probabilistic reasoning, and usually require large amounts of labeled data for training, which often are not available in many real-world scenarios.

In order to address these limitations, frameworks for modeling and reasoning about arguments have been developed. These frameworks model the problem via a Neuro-Symbolic approach, either using Integer Linear Programming or Probabilistic Logic Programming (PLP). While Integer Linear Programming is a well-known mathematical optimization technique that can be used to model and solve complex decision-making problems, the integration of such methods to constrain learning and inference is a challenging task. On the other hand, Probabilistic Logic Programming stands itself as a powerful tool with declarative semantics and more "readable" syntax for non-experts.

As of the time of writing, current PLP frameworks for modeling Argumentation Mining focused on using stratified programs, which largely restrict the expressiveness of the different argumentation problems one may desire to represent. Thus, we propose to model this problem using Probabilistic Answer Set Programming (PASP), a framework that combines the expressiveness of ASP with the probabilistic reasoning capabilities of PLP. Furthermore, in order to be able to use this PASP framework for scalable Neuro-Symbolic learning, we explore different state-of-the-art Knowledge Compilation (KC) techniques of the language, which are able to encode PASP programs into circuits that can be encoded as computational graphs in a variety of autodifferentiable frameworks, such as PyTorch or Jax.

Keywords: Probabilistic Logic Programming. Knowledge Compilation. Probabilistic Circuits. Argument Mining.

Resumo

Jonas Rodrigues Lima Gonçalves. **Mineração de argumentos usando programas de conjuntos de respostas probabilísticas**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2025.

A mineração de argumentação é uma tarefa complexa que tem sido abordada mais recentemente utilizando métodos puramente conexionistas. Esses métodos têm como objetivo extrair argumentos de dados textuais e representá-los em um formato estruturado. No entanto, esses métodos frequentemente carecem da capacidade de lidar com incertezas e raciocínio probabilístico, além de geralmente exigirem grandes quantidades de dados rotulados para treinamento, os quais muitas vezes não estão disponíveis em muitos cenários do mundo real.

Para lidar com essas limitações, foram desenvolvidos frameworks para modelar e realizar raciocínio sobre argumentos. Esses frameworks modelam o problema por meio de uma abordagem Neuro-Simbólica, utilizando Programação Linear Inteira ou Programação Lógica Probabilística (PLP). Enquanto a Programação Linear Inteira é uma técnica de otimização matemática bem conhecida que pode ser usada para modelar e resolver problemas complexos de tomada de decisão, a integração de tais métodos para restringir aprendizado e inferência é uma tarefa desafiadora. Por outro lado, a Programação Lógica Probabilística se destaca como uma ferramenta poderosa com semântica declarativa e uma sintaxe mais "legível" para não especialistas.

No momento da redação deste trabalho, os frameworks atuais de PLP para modelar a Mineração de Argumentação focaram no uso de programas estratificados, o que restringe amplamente a expressividade dos diferentes problemas de argumentação que se deseja representar. Assim, propomos modelar esse problema utilizando a Programação Lógica com Conjuntos de Respostas Probabilística (PASP), um framework que combina a expressividade da ASP com as capacidades de raciocínio probabilístico da PLP. Além disso, para ser capaz de usar esse framework PASP para aprendizado Neuro-Simbólico escalável, exploramos diferentes técnicas modernas de Compilação de Conhecimento (KC) da linguagem, que são capazes de codificar programas PASP em circuitos que podem ser representados como grafos computacionais em uma variedade de frameworks autodiferenciáveis, como PyTorch ou Jax.

Palavras-chave: Programação Lógico Probabilística. Compilação de Conhecimento. Circuito Probabilísticos. Mineração de Argumentos.

Lista of Abbreviations

2AMC	Second Level Algebraic Model Counting
AASC	Algebraic Answer Set Counting
AC	Arithmetic Circuit
AMC	Algebraic Model Counting
ASC	Answer Set Counting
BDD	Binary Decision Diagram
DNNF	Decomposable Negation Normal Form
d-DNNF	Deterministic Decomposable Negation Normal Form
EDLP	Extended Disjunctive Logic Programming
ELP	Extended Logic Programming
FOL	First Order Logic
KC	Knowledge Compilation
KR	Knowledge Representation
LC	Logic Circuit
LP	Logic Programming
NNF	Negation Normal Form
NP	Nondeteriministic Polynomial Time
OBDD	Ordered Binary Decision Diagram
PASP	Probabilistic Answer Set Programming
PC	Probabilistic Circuit
PLP	Probabilistic Logic Programming
PSDD	Probabilistic Sentential Decision Diagram
SDD	Sentential Decision Diagram
sd-DNNF	Smooth Decomposable Negation Normal Form
SDNNF	Structured Decomposable Negation Normal Form
WMC	Weighted Model Counting

List of Symbols

$X, Y, Z, \ldots$	Sets of variables
$x, y, z, \ldots$	Sets of assignments
$\langle f \rangle$	Semantics of Boolean formula f
$f \equiv g$	Equivalence between Boolean formulae f and g (i.e. $\langle f \rangle = \langle g \rangle$)
N, S, P, L	Sets of nodes
$Ch(N)$	Set of all children of node N
$Pa(N)$	Set of all parents of node N
$Desc(N)$	Set of all descendants of node N
$head(r)$	Head of a rule r
$body(r)$	The set of atoms inside the body of a rule r

List of Figures

List of Tables

List of Programs

Contents

Chapter 1

Introduction

Argumentation is a fundamental aspect of human communication, arising in contexts ranging from everyday discussions to formal debates and legal reasoning. The importance of argumentation extends beyond mere persuasion; it is central to critical thinking, decision-making, and the construction of knowledge. Hence, understading how arguments are structured, how they interact and how they can be evaluated is a key challenge for both social sciences and Artificial Intelligence (AI)

An area of growing interest within AI is *argumentation mining*, which seeks to automatically identify and extract argumentative structures from natural language texts (Stab and Gurevych, 2017). Therefore, this field can be greatly enhanced by Neuro-Symbolic AI techniques, that combine the strengths of symbolic reasoning and connectionist learning. More specifically, connectionist models excel at processing unstructured data, such as text, extracting argumentative components and their relations. On the other hand, symbolic reasoning provides a framework for robustly constraining and interpreting these components, either in inference or learning.

Core tasks in argumentation mining may include: separating argumentative from non-argumentative spans, classifying argument components, and predicting the support or attack relations between them. These tasks share tight interectios, where errors in earlier stages, such as claim identification, can easily propagate and degrade the coherence of the recovered argumentative graph (Stab and Gurevych, 2017). By optimizing these tasks jointly within a structured framework, with global constraints, we can mitigate this error propagation and improve the system's overall performance.

One of the main approaches for this task was proposed in (Stab and Gurevych, 2017), where the authors introduce a joint model for argumentation mining based on Integer Linear Programming (ILP). Their model combines local classifiers for argument component identification and relation prediction with global constraints that enforce the well-formedness of the argumentative structure. This joint model demonstrates improved performance over pipeline approaches, highlighting the benefits of integrating local predictions with global reasoning. On the other hand, the ILP-based approach struggles to naturally express uncertainty in the predictions, and its reliance on discrete optimization can limit its scalability and flexibility.

More recent approaches focused on leveraging Probabilistic Logic Programming (PLP) systems, such as ProbLog (**fierens2015**), to model argumentation mining. These approaches benefit from the declarative nature of PLP, allowing for more interpretable models that can naturally incorporate uncertainty and logical reasoning, without the need of expert knowledge to define complex ILP constraints. For instance, the framework proposed by (**cerveira2020joint**) leverages DeepProbLog (Manhaeve *et al.*, 2018) to combine neural networks to handle the connectionist part of the task, with ProbLog to model the symbolic reasoning about argumentative structures. This hybrid approach allows for end-to-end learning while maintaining the interpretability and expressiveness of the symbolic component.

Another relevant work is (Totis *et al.*, 2023), which extends the ProbLog framework by incorporating the Stable Model semantics from Answer Set Programming (ASP). This extension, known as smProbLog, enables the modeling of more complex argumentative patterns, such as negative cycles, which may arise in contradictory or even vague argumentative contexts. By adopting stable model semantics, smProbLog provides a more expressive framework for argumentation, allowing for richer representations of argumentative structures and their interactions.

This dissertation examines how probabilistic logic programming, through the usage of knowledge compilation compilation techniques, can build end-to-end Neuro-Symbolic pipelines to learn rich argumenatative structures from data, detecting how claims, premises, and attacks interplay in text. We aim to explore how Deep Probabilistic Answer Set Programming can bridge the gap between symbolic argumentation theory and data-driven argumentation mining. By embedding argumentative structures directly in probabilistic circuits, we aim to support transparent reasoning, quantified uncertainty, and efficient integration with neural learning.

This works aims to explore the following topics:

- How to formalize argumentation mining tasks as PASP programs that capture the mutual influence of claims, premises, and attacks while retaining probabilistic semantics for uncertain or conflicting evidence.
- The development of knowledge compilation pipeline that encodes PASP programs into structured probabilistic circuits amenable to tractable inference.
- Techniques for converting such circuits into differentiable structure that may be used in neuro-symbolic learning strategies that exploit the compiled circuits to align neural predictions with argumentative constraints, paving the way for scalable argumentation mining under uncertainty.

The remainder of this document is organized as follows. The first three chapters Chapters chapter 2, chapter 3, and chapter 4 reviews basic concepts in propositional satisfiability, model counting over semirings, and logic programming foundations. After, in Chapter **??**, we introduce how PASP can used to encode argumentative structures. Building on this, we introduce basic concepts related to Knowledge Compilation and the proposed pipeline for end-to-end Neuro-Symbolic learning for argumentation mining. Finally, we situate our approach within existing work, and discuss future directions.

Chapter 2

Appendix Satisfiability and Proof Theory

Logic concerns two main interdefinable ideas: *consistency* and *validity* (Franco and Martin, 2009). Moreover, this strongly interconnected relationship can be described either syntactically or semantically, where the former approach results in definitions related to *proof theory*; and the latter one to *model theory*.

When considering the first approach, it is necessary to take into consideration that the *syntax* is restricted to definitions that refer to the syntactic form of the sentences in question, the grammatical structure of the language. For *proofs* to be carried out, it is necessary to define an axiom system, with a set of axioms and inference rules that allows the derivation of theorems, where *derivability* is the syntatic notion of *validity* (Franco and Martin, 2009). Similarly, the notion of *consistency* under a syntactic approach is defined as the impossibility of deriving a contradiction from a set of related sentences (Franco and Martin, 2009). Since, *consistency* is defined with respect to *derivability*, and the latter is defined entirely syntactically, the notion of both *consistency* and *validity* are also syntactic.

On the other hand, a semantic approach, that gives rise to *model theory*, is concerned with the concept of *truth*, with respect to the interpretation of some algebraic structure, usually referred to as *the world*, where the sentences are interpreted as *true* or *false* via a well-defined function (Franco and Martin, 2009). Similarly to the syntatic approach, both the semantic versions of *validity* and *consistency* are defined under the same concept, this one being the concept of *structure*. In *model theory*, the *validity* ideia is also called *validity*, where an argument is valid if, whenever the premises are true, so is the conclusion. As for the semantic version of the *consistency* concept, on the other hand, it is called *satistiability*. A set of formulas is said to be *satisfiable* if there is some structure in which all of its components are *true*; if such structure does not exists, the set is said to be *unsatisfiable* (Franco and Martin, 2009).

Even though both the syntactic and semantic versions of *validity* and *consistency*, *derivability* and *consistency*, respectively, have vastly different definitions, the concepts from these two branches of logic are closely intertwined. For a *complete* axiom system, it is true that the syntactic and semantic concepts match up so as to *validity* and *consistency*

coincide exactly, and the same for *derivability* and *satisfiability*.

A very explicit relationship between *satistiability* and *proof theory* resides on the next chapter about Answer Set Programming (ASP - Chapter4), in which the need for efficient algorithms to solve *satistiability* problems is crucial for the development of ASP solvers. Moreover, this intrinsic relationship between *satisfiability* and many other Computer Science problems is also reflected by the work of Shannon, who paved the way for the connection between Logic and Circuits (Shannon, 1938), which enabled advancements such as Binary Decision Diagrams (BDDs) (Akers, 1978; Bryant, 1986), that were thought to be (during the 80's and 90's) the best way to tackle real-world applications related to logical problems, such as SAT and *model counting* (Akers, 1978; Bryant, 1986).

This relation between SAT and Circuits Theory, despite being a well-studied subject, is also what motivates the development of this dissertation. Since the complexity relationships that carry over from *satistiability* deeply influence the complexity of developing efficient methods for solving Probabilistic Answer Set Programming (PASP) and other Probabilistic Logic Programming (PLP) problems; what makes the development of efficient algorithms both challenging and open to new research opportunities.

2.1 Notation

We borrow definitions and notation from the work of (Arora and Barak, 2009) and (Kimmig *et al.*, 2017). Since this chapter is focused on logical SAT-like problems, we only introduce basic logic concepts, such as *Boolean formulas*, CNFs and *Logical Propositional Theories*.

Definition 2.1.1 (Boolean Formulas)**.** A *Boolean formula* is the combination of *Boolean variables*, that can be either $\top$ or $\bot$, and logical operators, such as $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT, also denoted by an overline) (Arora and Barak, 2009).

Example 2.1.1. *For example, the formula ϕ defined by*

$$\phi = (u_1 \wedge u_2) \vee \neg(u_3 \wedge \overline{u}_4)$$

is a Boolean formula over variables u_1, u_2, u_3, u_4. We denote by $\phi(a)$ the evaluation of ϕ over an assignment a of the variables in the formula.

Definition 2.1.2 (Conjunctive Normal Form)**.** A Boolean formula over variables $u_1, \dots u_n$ is in CNF if it is a conjunction of clauses, where each clause is a disjunction of literals, and a literal is either a variable or its negation (Arora and Barak, 2009). Hence, a CNF is a Boolean formula of the form

$$\bigwedge_i \left(\bigvee_j v_{ij} \right),$$

where each v_{ij} is a literal over variables $u_1, \dots u_n$.

Example 2.1.2. *For example, the following formula is a 3CNF (a CNF where each clause has*

at most 3 literals) over variables u_1, u_2, u_3, u_4*:*

$$(u_1 \vee \overline{u}_2 \vee u_3) \wedge (u_2 \vee \overline{u}_3 \vee u_4) \wedge (\overline{u}_1 \vee u_3 \vee \overline{u}_4).$$

Definition 2.1.3 (Disjunctive Normal Form)**.** A Boolean formula over variables $u_1, \dots u_n$ is in DNF if it is a disjunction of clauses, where each clause is a conjunction of literals. Hence, a DNF is a Boolean formula of the form:

$$\bigvee_i \left(\bigwedge_j v_{ij} \right),$$

where each v_{ij} is a literal over variables $u_1, \dots u_n$.

Definition 2.1.4 (Logical Propositional Theory)**.** A *Logical Theory* is set of logical sentences, Boolean-valued formulas with no free variables. Hence, a *Propositional Theory* is Logical Theory where the sentences are propositional formulas, i.e., formulas with no quantifiers.

2.2 Satisfiability

The SAT problem is a fundamental problem in both Artificial Intelligence and Theory of Computation. This importance is due not only by the logical nature of SAT, but also by the fact that all NP problems can be reduced to SAT in polynomial time, thus its NP-completeness (Cook, 1971; Levin, 1973). In this manner, the SAT problem is a natural candidate for encoding not only arbitrary a wide range of logical theories, but also for encoding any NP problem in general. Therefore, methods to solve SAT fast are of great interest.

Later in this work, when we discuss probabilistic inferences related to Probabilistic Argumentation (via Probabilistic Answer Set Programming), we will see that being able to solve numerous logical queries in polynomial time is a desirable feature, in order to scale an argumentation mining problem over learning Neural Networks. Therefore, our goal will not be to solve SAT only once; the necessity of solving it many times creates the need to develop tractable methods for solving multiple queries.

Thus, we begin this thesis with a formal definition of the SAT problem and its variations, such as Max-SAT and Sharp-SAT (#SAT - also known as SAT-Counting).

2.2.1 Satisfiability

The Boolean Satisfiability Problem is the problem of determining whether a given CNF is *satisfiable*, i.e., whether there is an assignment of $True$ or $False$ to the variables that makes the formula $True$ (Arora and Barak, 2009). If such an assignment does not exist, the formula is said to be *unsatisfiable*.

Definition 2.2.1 (SAT - Satisfiability)**.** Let SAT be the language of all satisfiable CNFs. Then, we say that for a CNF ϕ, the SAT problem is defined as

$$\text{SAT}(\phi) = \{\phi \mid \phi \text{ is a satisfiable propositional formula}\}.$$

For example, $(True, False, False, False)$ is an example of assignment that makes the 3CNF of definition 2.1.2 satisfiable. On the other hand, the CNF $x \wedge \overline{x}$ is unsatisfiable, since it is a *contradiction* (the opposite of a tautology).

2.2.2 MAX-SAT

The Max-SAT problem is the problem of finding an assignment that satisfies the maximum number of clauses in a given CNF (Krentel, 1988).

Definition 2.2.2 (MAX-SAT)**.** Formally, let ϕ be a CNF over variables $x_1, \ldots, x_n$. Then, the Max-SAT problem is to find an assignment a that maximizes the sum of the Iverson brackets of the clauses in C, such that:

$$\phi(a) = \bigwedge_{c \in C} c(a),$$

where $c(a)$ is the evaluation of c over the assignment a, and $[c(a)]$ is the Iverson bracket, which is equal to 1 if $c(a)$ is $True$, and 0 otherwise.

Sometimes, the Max-SAT problem refers to the weighted version of the problem, where each clause has a non-negative weight, and the goal is to maximize the sum of the weights of the satisfied clauses.

Definition 2.2.3 (Weighted MAX-SAT)**.** Formally, let ϕ be a CNF over variables $x_1, \ldots, x_n$, and let $w : C \to \mathbb{R}^+$ be the weight function over the clauses of ϕ. Then, this variant of the Max-SAT problem, called *Weighted* Max-SAT, is defined as the problem of finding an assignment a such that it maximizes the following:

$$\sum_{c \in C} w(c) \cdot [c(a)],$$

subject to the constraint that

$$\phi(a) = \bigwedge_{c \in C} c(a).$$

An interesting difference between the SAT and Max-SAT is that the former is a decision problem, while the latter is an optimization problem. Another relation between them is that SAT is a special case of the Max-SAT, where it is possible to obtain a solution where the maximum number of possible clauses satisfied is the total number of clauses. Thus, there is a reduction from SAT to Max-SAT, which is polynomial in time. Moreover, because of this reduction, one can claim that the Max-SAT problem is NP-hard, since SAT is NP-complete.

The reduction from Max-SAT to weighted Max-SAT is simple too: just set all weights to 1 (or any other constant), and the problem is reduced to the unweighted version. Therefore, the weighted version of the problem is also NP-hard, as one would expect. Moreover, the Max-SAT problem is also OptP-complete (Optimization Polynomial Time), similar to the *Traveling Salesman* and *Knapsack* problems (Krentel, 1988).

2.2.3 Sharp-SAT

Differently from the SAT and Max-SAT problems, the $\#SAT$ (or Sharp-SAT) problem is a counting problem, where the goal is to count the number of solutions of a given CNF has (VALIANT, 1979). Therefore, one can see the Sharp-SAT problem closer to Levin studied problems (LEVIN, 1973) than Cook's (COOK, 1971), in the sense that the goal is to retrieve a solution, because there will always be one. A way to perform this computation would be a search problem over **all** possible assignments over the variables of the CNF, and count the number of them that satisfy the formula. Note that this is different from search problems, where finding one solution solves the problem, and the search can be stopped.

This relation between Sharp-SAT and SAT variations does not stop here. The Sharp-SAT is clearly NP-hard, since it is possible to reduce SAT to Sharp-SAT in polynomial time, because knowing the number of solutions of a CNF is equivalent to knowing whether it has at least one solution. Moreover, Sharp-SAT is a Sharp-P-complete problem (or $\#P-$complete), which means that it is capable of reducing the set of the counting problems associated with the decision problems in the set NP.

Definition 2.2.4 (#SAT)**.** Formally, let ϕ be a CNF over variables $x_1, \ldots, x_n$. Then, the Sharp-SAT problem is to find the natural number n such that

$$\#SAT(\phi) = |\{a \in \{0,1\}^n \mid a \text{ is an assignment of } \phi\}|.$$

Similar to the Max-SAT problem, there is a weighted version of the Sharp-SAT problem, called Weighted Model Counting (WMC), where a non-negative weight is associated to each assignment of values to variables (CHAKRABORTY *et al.*, 2015). The classical version of the problem, usually called Model Counting (MC), can be seen as a special case of the weighted version, where all weights are equal to 1, and is sometimes referred to as *unweighted model counting*.

Definition 2.2.5 (Weighted Model Counting)**.** The Weighted Model Counting problem is the problem of finding the sum of the weights of all assignments that satisfy a given CNF. Formally, let ϕ be a CNF over variables $x_1, \ldots, x_n$, and let $w : \{0,1\}^n \to \mathbb{R}^+$ be the weight function over the assignments of the variables in ϕ. Then, the WMC problem is

$$\text{WMC}(\phi) = \sum_{a \in \{0,1\}^n} w(a) \cdot [\phi(a)].$$

Chapter 3

Probabilistic Logical Inference

The SAT problem can be coined as the cornerstone of almost all research relating intractable problems in Computer Science. Its pure logical nature arises many problems related to Combinatorial or decision problems (Cook, 1971; Levin, 1973). While SAT itself is a decision problem, its optimization counterpart, the Max-SAT problem, is also of great importance in many fields, from Artificial Intelligence to Operations Research (Kolokolov *et al.*, 2013; Gomes *et al.*, 2006) and many other fields. On the other hand, the Sharp-SAT problem elevates the combinatorial nature of SAT to a counting problem, which increases its complexity and allows it as a reduction to model many other counting problems (valiant, 1979).

Another relevant generalization of SAT is the WMC, a probabilistic generalization of Sharp-SAT, where each assignment of the variables in the propositional logic theory has an associated weight, and the problem is to compute the sum of weights of all satisfying assignments.

The great importance of WMC, defined in Chapter 2, is due to the fact that it provides a framework for performing probabilistic inference. This result stems from the possibility of reducing probabilistic inference calls to WMC on a propositional knowledge base Chavira and Darwiche, 2008. Specifically, this approach derives from the possibility of encoding the target probabilistic model, usually represented as a Bayesian Network, into a set of propositional clauses, where each clause is a knowledge base in CNF, and then assign weights to the CNF variables based on the network probabilities (Darwiche, 2002; Chavira, Darwiche, and Jaeger, 2006; Chavira and Darwiche, 2008; Sang *et al.*, 2005; Costa *et al.*, 2012).

This possibility of reducing general probabilistic inference queries on Bayesian Networks to WMC not only provides a declarative method for encoding local structure (specific properties of network parameters) and a powerful way for exploiting available evidence (Chavira and Darwiche, 2008), but also enables the use of highly optimized SAT solvers or Knowledge Compilation techniques (Kimmig *et al.*, 2017; Chavira and Darwiche, 2008), which renders some of today's most efficient techniques for probabilistic inference (Kimmig *et al.*, 2017).

Given the importance of all problems described above, a unified framework capable of modeling all of them was proposed, called Algebraic Model Counting (AMC) (Kimmig *et al.*, 2017). Moreover, a second level of this framework, naturally called Second-Level Algebraic Model Counting (2AMC), was proposed recently (Kiesel *et al.*, 2022), which is capable of reducing inference in PASP.

3.1 Algebraic Model Counting

Before defining the AMC problem, we need to define the type algebraic structure that is used to generalize tasks such as SAT, Sharp-SAT and WMC. More specifically, by generalizing these SAT-like problems to a more abstract WMC over a different algebraic structure, a *comutative semiring*, we can define the AMC problem as follows (Kimmig *et al.*, 2017):

3.2 Semirings

Definition 3.2.1 (Semiring)**.** A semiring is an algebraic structure $(\mathcal{A}, \oplus, \otimes, e_\oplus, e_\otimes)$, where addition $\oplus$ and multiplication $\otimes$ are associative binary operations over the set $\mathcal{A}$, $\oplus$ is commutative, $\otimes$ distributes over $\oplus$, $e_\oplus \in \mathcal{A}$ is the neutral element of (the sum operator) $\oplus$, $e_\otimes \in \mathcal{A}$ is the neutral element of (the product operator) $\otimes$, and for all $a \in \mathcal{A}$, $e_\oplus \otimes a = a \otimes e_\oplus = e_\oplus$. In a commutative semiring, $\otimes$ is commutative as well.

One particular notable semiring is the *Tropical Semiring*, in the context of *idempotent analysis*, where the sum operator $\oplus$ is defined as the minimum operator, and the product $\otimes$ operator is defined as the usual real sum. The set $\mathcal{A}$ in this case is the set of extended real numbers, $\{\mathbb{R} \cup \{+\infty\}\}$. This important semiring has various applications, with a special focus in *Tropical Analysis* and *Tropical Geometry*, and is named after Imre Simon's extensive work on this specific semiring (Simon, 1988). Within the context of AMC, the *Tropical Semiring* can be used to perform the Max-SAT problem, where the sum operator is responsible for counting the number of satisfied clauses, and the maximum operator is responsible for finding the maximum number of satisfied clauses.

3.3 Algebraic Model Counting

Definition 3.3.1 (Algebraic Model Counting)**.** Given

- A propositional logic theory T over a set of variables $\mathcal{V}$;
- A commutative semiring $(\mathcal{A}, \oplus, \otimes, e_\oplus, e_\otimes)$; and
- A labeling function $\alpha : \mathcal{L} \to \mathcal{A}$, mapping literals $\mathcal{L}$ of the variables in $\mathcal{V}$ to elements of the semiring set $\mathcal{A}$.

The AMC problem is now defined as the computation of the following expression:

$$A(T) = \bigoplus_{I \in \mathcal{M}(T)} \bigotimes_{i \in I} \alpha(i),$$

where $\mathcal{M}(T)$ denotes the set of models of T.

To anyone familiar with introductory algebra, it is easy to see that the definition above for the AMC problem indeed reduces many of the presented SAT like problems. For example, by setting the semiring to

$$(\mathcal{A}, \oplus, \otimes, e_{\oplus}, e_{\otimes}) = (\{true, false\}, \vee, \wedge, false, true),$$

and α maps positive literals to *true* and negative literals to *false*, we can see that the AMC problem is capable of solving the SAT problem. Moreover, by setting a similar semiring

$$(\mathcal{A}, \oplus, \otimes, e_{\oplus}, e_{\otimes}) = (\mathbb{N}, +, \times, 0, 1),$$

and α to map literals to 1 and 0, if they are positive or negative, respectively, we can see that the AMC models model counting.

Similarly, by only changing the set $\mathcal{A}$ of the semiring to $\mathbb{R}^+$ (non-negative real numbers) and α to also map literals to $\mathbb{R}^+$, we can see that the AMC problem is capable of solving the WMC problem.

Moreover, by the results cited previously about Bayesian Networks, one can see that this also means that AMC is also capable of reducing general probabilistic inference (Darwiche, 2002; Chavira, Darwiche, and Jaeger, 2006; Chavira and Darwiche, 2008; Sang *et al.*, 2005). However, the algebraic structure of AMC makes it easy to show that it is capable of modeling probabilistic inference: by using the same semiring used to model the WMC problem and just by changing the α function to represent a probability distribution over the literals, setting $a(v) \in [0, 1]$ and $a(\bar{v})$ to be its complement.

In fact, AMC is capable of modeling many other interesting problems, such as $EXPEC$ (expectation), which allows one to infer parameters in a Finite State Transducer model w.r.t. to a given dataset. Hence, the elements of the respective expectation semiring are tuples of the form (p, v), where $p \in [0, 1]$ is a probability of an arc of the Finite State Transducer and $v \in \mathbb{R}$ is the value of this respective arc. Then, the operations $\oplus, \otimes$, neutral elements $e_{\oplus}, e_{\otimes}$, and the labeling function α are defined as follows:

$$\begin{aligned}
(p_1, v_1) \oplus (p_2, v_2) &= (p_1 + p_2, v_1 + v_2), \\
(p_1, v_1) \otimes (p_2, v_2) &= (p_1 \cdot p_2, p_1 \cdot v_2 + p_2 \cdot v_1), \\
e_{\oplus} &= (0, 0), \\
e_{\otimes} &= (1, 0), \\
\alpha(a) &= \begin{cases} (p, 1) & \text{if } i = k, \\ (p, 0) & \text{else.} \end{cases},
\end{aligned}$$

where a is the respective arc of the Finite State Transducer, in relation to the tuple (p, v).

Adapted from (Kimmig *et al.*, 2017), Table 3.1 summarizes the AMC problem for different

logical and probabilistic inference tasks, describind in depth the respective sets, operations and labeling functions associated with each semi-ring.

Task	$\mathcal{A}$	$\oplus$	$\otimes$	$e^{\oplus}$	$e^{\otimes}$	$\alpha(v)$	$\alpha(\neg v)$
SAT	$\{true, false\}$	$\vee$	$\wedge$	$false$	$true$	$true$	$true$
#SAT	$\mathbb{N}$	$+$	$\times$	0	1	1	1
WMC	$\mathbb{R}_{\geq 0}$	$+$	$\times$	0	1	$\in \mathbb{R}^+$	$\in \mathbb{R}^+$
PI	$\mathbb{R}_{\geq 0}$	$+$	$\times$	0	1	$\in [0,1]$	$1-\alpha(v)$

Table 3.1: *Examples of logical and probabilistic inference tasks that can be modeled as instances of the AMC, and their corresponding semirings and labeling functions.*

Due to the importance of the semiring associated with SAT and probabilistic inference, we give a special name to them: the *Boolean* and the *Probability* semi-rings.

Other useful problems that can be modeled by the AMC task are: *sensitivity analysis* (SENS), *probability of most likely states* (MPE), *shortest* and *widest path* (S-PATH and W-PATH, respectively), *fuzzy* and *k-weighted* constraints (FUZZY and k-WEIGHT, respetively), and $OBDD_<$ *construction* (Kimmig *et al.*, 2017).

3.4 Second Level of Algebraic Model Counting

As mathematical are prone to do, there is a generalization of AMC, called 2AMC, which is capable of modeling problems where there is the need of a third operation, besides $\oplus$ and $\otimes$. This third operation appears in many applications, such as the Maximum a Posteriori (MAP) probabilistical query, but is also essential when modeling PASP problems, as we will see in more detail in the next chapter 4.

Similarly to the AMC, follows the definition of the 2AMC problem (Kiesel *et al.*, 2022):

Definition 3.4.1 (Second-Level Algebraic Model Counting)**.** Given

- A propositional logic theory T over a set of variables $\mathcal{V}$;
- A partition of the variables in T, $(\mathbb{X}_I, \mathbb{X}_O)$;
- Two commutative semiring $S_j = (\mathcal{A}_j, \oplus_j, \otimes_j, e_{\oplus_j}, e_{\otimes_j})$, for $j \in \{I, O\}$;
- Two labeling function $\alpha_j : X_j \to \mathcal{A}_j$, for $j \in \{I, O\}$, mapping literals of the variables in X_j to elements of the semiring set $\mathcal{A}_j$; and
- A weight transformation function $t : \mathcal{A}_I \to \mathcal{A}_O$ that respects $t(e_{\oplus_I}) = e_{\oplus_O}$.

Then the 2AMC problem is defined as the computation of the following expression:

$$2AMC(T) = \bigoplus_{\mathbf{a} \in A(X_O)}^{O} \bigotimes_{a \in \mathbf{a}}^{O} \alpha_O(a) \otimes_O t\left(\bigoplus_{I \in \mathcal{M}(T|\mathbf{a})}^{I} \bigotimes_{i \in I}^{I} \alpha_I(i) \right),$$

where $A(X)$ denotes the set of assignments of x to $X \in \mathcal{V}$, and $\mathcal{M}(T|a)$ denotes the set of models of T given an assignment a.

It is easy to see that AMC is an instance 2AMC, where the first partition is empty, $X_O = \emptyset$, and the weight transformation is the identity function. Thus, the leftmost half of the equation (all elements outside the parentheses) is equal applying the identity function, and the rightmost half is equivalent to the AMC, because the assignment $\mathbf{a}$ inside the summation is empty, therefore, $M(T|\mathbf{a}) = M(T)$. This implies that 2AMC tries to solve an instance of AMC over the variables present in the partition X_I for each possible assignment of the variables in X_O, applying a weight transformation to the result. From the algebraic point of view, this weight transformation function maps the result of the AMC inside the parentheses to the semiring S_j, used on the left side of the equation. Thus, this transformation enables one to solve a second AMC instance over the variables in X_O.

Therefore, analyzing the 2AMC problem, we can see that it partitions variables in $\mathcal{V}$ into two sets, X_I and X_O, and then solves an inner AMC instance over the variables presents in X_I for each assignment to X_O. Then, uses the results of this inner AMC instance to solve a second (level) AMC instance over the variables in X_O.

Perhaps, a non-trivial example of a problem that can be modeled as an instance of 2AMC is the MAP query, which is defined to be the most probable assignment q given an evidence e (the assignment of the remaining variables that is more likely given the evidence). Formally, given a propositional logic theory T over variables $\mathcal{V}$, a joint probability distribution P over T, a conjunction e of observed literals for the set of evidence atoms E, and a set of ground query atoms Q: find the most probable assignment q to Q given the evidence e, with $R = \mathcal{V} \setminus (Q \cup E)$

$$MAP(Q|e) = \arg\max_{q} P(Q = q|e) = \arg\max_{q} \sum_{\mathbf{r} \in \mathcal{A}(R)} P(Q = q, e, R = r),$$

where $\mathcal{A}(R)$ denotes the set of assignments of r to R.

Under the assumption that the distribution the Random Variables respective to the variables in Q are independent (the distribution is factorized w.r.t. Q), we can see that MAP query consists of two steps:

1. A first AMC task of solving the sum over the truth values of the atoms in R, when considering fixed assignments to the atoms both in E and Q; and
2. A second AMC task of determining the assignments to the variables in Q that maximize the inner sum of the first AMC task.

This intuitive description of the MAP query illustrates how it could be modeled as an instance of the 2AMC problem. By having the partitions of variables to be $X_O = Q$ and $X_I = \mathcal{V} \setminus Q = R \cup E$, we have a compatible partition of the variables in $\mathcal{V}$ w.r.t. the MAP problem.

With the partition of variables defined, there is still the need to define the semirings, labeling functions, and the weight transformation function t. Since the rightmost half of the equation is w.r.t. to summing the probabilities over assignments r to R, the semiring

used for this part of the equation is $S_I = ([0,1], +, \times, 0, 1)$ (the same semiring used for probabilistic inference); and the labeling function α_I maps literals present in e to 1 and 0 to their negation, $P(r)$ and $1 - P(r)$ to the positive and negative literals present in R (thus, the probability of assignments that are not "compatible" with e is zero).

On the other hand, the leftmost half of the equation represents a maximization of the truth values of the atoms in Q, given the result of the inner AMC instance. Therefore, the semiring used for this part of the equation is $S_O = (R^+ \times 2^{|Q|}, \oplus_{\text{argmax}}, \otimes_{\text{argmax}}, (0, \emptyset), (1, \emptyset))$, where

$$(p_1, q_1) \oplus_{\text{argmax}} (p_2, q_2) = \begin{cases} (p_1, q_1) & p_1 > p_2 \\ (p_2, q_2) & p_1 < p_2 \\ (p_1, q_{\text{min}}) & p_1 = p_2 \end{cases}$$

and q_{min} represents the smallest assigment of literals, w.r.t. a predefinied lexycographic order of the variables Q; and $\otimes_{\text{argmax}}$ is defined as the product between the probabilities of the assignments, and the union of the assignments of the literals, $(p_1, q_1) \otimes_{\text{argmax}} (p_2, q_2) = (p_1 \cdot p_2, q_1 \cup q_2)$. Moreover, to complete the definition of the this leftmost AMC instance, the labeling function $\alpha_O(l)$ is defined to be: $(p, \{l\})$ for the positive literals; and $(1 - p, \{l\})$ for the negative ones.

Finally, to completely describe MAP as an instance of the 2AMC problem, we construct the function $t(p)$, where p is the resulting probability of the inner (rightmost) AMC task, as being $t(p) = (p, \emptyset)$. This weight transformation function takes the sum of the probabilities as input and just returns the Cartesian Product of the probability and the empty set, which is the neutral element of the $\oplus_{\text{argmax}}$ operator. Next, the $\otimes_{\text{argmax}}$ will take the product of the probabilities an assignment of a variable in Q and the result probability of the inner AMC. In other words, the probability $P(q)$, where $q \in \mathbf{q}$ and $\mathbf{q}$ is the assignment of the variables in Q, is multiplied by the sum of probabilities conditioned on the assignment q and evidence e. The second part of the tuple is the union between q and $\emptyset$, which is equal to q. Thus, the $\otimes_{\text{argmax}}$ applies the Bayes Rule to the result of the inner AMC, multiplying the probability of the assignment of the variables. Therefore, the last step, $\oplus_{\text{argmax}}$, takes the assignment that maximizes the probability of the computation performed by the $\otimes_{\text{argmax}}$ operator, which results in the most probable assignment of the variables in Q.

Chapter 4

Answer Set Programming

The study of Knowledge Represenation (KR) aspires to find compromises between expressiveness and efficiency; so that one of the motivating reasons for the research in this area is the well-known fact that calculating entailments of arbitrary first-order logical Knowledge Base (KB) can be intractable (H. J. Levesque, 1986). Even when considering the usage of heuristics and refinements, the Resolution procedure can still be exponential (H. J. Levesque, 1986).

Due to this intrinsic complexity of First-Order Logic (FOL), the main alternative to Knowledge Base representation is the use of less expressible representations, such as propositional logic. This idea of restricting the expressiveness of a logic formalism in order to obtain a more efficient reasoning procedure is a fundamental trade-off that characterizes the Knowledge Representation research (H. J. Levesque and R. J. Brachman, 1987).

In the previous appendix 2, we talked about the complexity class of SAT-like encompasses several hard computational problems, with focus on logical ones. A not so distant problem, from a logical standpoint, is the Resolution procedure, which used to calculate entailments of any first-order logical KB (R. Brachman and H. Levesque, 2004). In other words, Resolution is an inference procedure that leads to a refutation-complete theorem-proving technique for Propositional Logic and FOL.

As one could imagine, by its relationship with SAT, in its most general form, Resolution ran into serious computational difficulties (R. Brachman and H. Levesque, 2004). Although refinements to Resolution may lead to more efficient routines, the complexity of this problem is intrinsically high due to its nature (R. Brachman and H. Levesque, 2004). This is a consequence of the fundamental computational intractability of first-order entailment.

Therefore, in this chapter, we will explore the idea of limiting ourselves to only a certain interesting subset of FOL, where the Resolution procedure becomes much more manageable. We will also see that from a representation standpoint, that the subset in question, Answer Set Programming (ASP) is still sufficiently expressive for many real-world applications purposes, such as Argumentation (Toni and Sergot, 2011).

4.1 Foundations of Logic Programming

4.1.1 Definite Logic Programs

Before diving on the semantics of Probabilistic Answer Set Programming (PASP), it is important to understand less expressive formalisms, such as *definite* and *propositional* programs. These formalisms are not only the basis where Answer Set Programming (ASP) is built upon, but also hint at the underlying complexity of the problems that Answer Set Programming (ASP) can solve and the efficiency of the algorithms that can be used in this context. Thus, we start by defining the simplest form of Logic Programming (LP), called *definite* (or *positive*) programs:

Definition 4.1.1 (Definite Logic Programs)**.** A definite logic program P is a finite set of clauses (rules) in the form

```
a :- b1, ..., bM.
```

where $a, b_1, \ldots, b_m$ are atoms of a function-free FOL L; and this rule can be seen as material implication restricted to Horn clauses, where `a :- b1, ..., bM` is read as $B \supset A$ or $B \rightarrow A$ (Eiter *et al.*, 2009). The atom a is called the *head* of the rule, while $b_1, \ldots, b_m$ are called the rule's *body*. When a rule has an empty body, it is called a fact and can be shortened as a.

The following program is an example of a definite program:

```
happy(turing) :- friends(turing, vonNeumann).
```

Programs without variables, like the one above, are called *propositional* programs.

We could skip the definition of propositional programs, if we were not interested in the complexity of this problem, that is not only $P - complete$, but can be solved in linear time (testing satisfiability of propositional (Horn) formulas) (Dowling and Gallier, 1984).

4.1.2 Herbrand Universe, Base and Interpretation

A natural question that arises after the definition of this new class of program is about its complexity. This answer largely depends on the type of normal program that we are dealing with. In search of a more formal explanation, we have to introduce more concepts to capture a class of programs that we are comfortable between the trade-off of expressiveness and efficiency. We do this by firstly defining the concept of *Herbrand Universe* and *Base*:

Definition 4.1.2 (Herbrand Universe)**.** The *Herbrand Universe* $HU(P)$ of a logic program P is the set of all terms that can be formed from constants and functions in P (w.r.t. a predefined vocabulary L). Moreover, the *Hebrand Base* $HB(P)$ of P is the set of all ground atoms that can be formed from terms and predicates occurring $HU(P)$. Finally, ta *Herbrand Interpretation* is a subset of $HB(P)$, an interpretation I (a set denoting ground *truths*) over $HU(P)$.

Example 4.1.1. *Consider the following logic program P (Eiter et al., 2009):*

```
h(0, 0).
```

```
t(a, b, r).
p(0, 0, b).
p(f(X), Y, Z) :- p(X, Y, Z'), h(X, Y), t(Z, Z', r).
h(f(X), f(Y)) :- p(X, Y, Z'), h(X, Y), t(Z, Z', r).
```

Then, the Herbrand Universe $HU(P)$ is the union of the set containing all constants of P, $\{0, a, b, r\}$, and the set of terms that can be formed from these constants $\{f(0), f(a), f(b), f(r), f(f(0)), f(f(a)), f(f(b)), f(f(r)), ...\}$. Whereas, the Herbrand Base, $HB(P)$ is given by the set of all ground atoms assertions, $\{p(0,0,0), p(a,a,a), \dots, h(0,0), \dots, t(0,0,0), t(a,a,a), ...\}$.

Finally, we list a few Herbrand Interpretations over $HU(P)$:

- $I_1 = \emptyset$;
- $I_2 = HB(P)$;
- $I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}$.

Note that not all interpretations are consistent with the program P. For instance, the interpretation I_1 is contradictory, because it contains does not contain any of the facts of P.

4.1.3 Grounding of a Logic Program

With the notion of Herbrand Universe and Base, we can now have a moral formal definition of *grounding* (Eiter *et al.*, 2009):

Definition 4.1.3 (Grounding)**.** We define a ground instance of a clause C, of a logic program P, as any clause C' obtained from C by applying a substitution

$$\theta : Var(C) \rightarrow HU(P),$$

where $Var(C) \in \mathcal{V}(P)$ is the set of variables in C. Moreover, the grounding of a program P is the set of all possible ground instances of the clauses in P and is denoted by $ground(P) = \bigcup_{C \in P} ground(C)$ (the union of all ground instances for all the clauses in P).

4.1.4 Interpreation of a Logic Program

As we are already formalizing the concepts like grounding, the Interpretation of a logic program is also a concept that needs more attention. Thus, follows its definition:

Definition 4.1.4 (Interpretation)**.** The interpretation I of a logic program P is a model of P that is compatible with the assertions in P. That is, I is a model of

- A ground clause $C =$ `a :- b1, ..., bM, not c1, ..., cN`, denoted $I \models C$, if either $\{ a, c_1, \dots, c_N \} \cap I \neq \emptyset$ or $\{ b_1, \dots, b_M \} \nsubseteq I$;
- A clause C, denoted $I \models C$, if $I \models C'$ for every $C' \in ground(C)$;
- A program P, denoted $I \models P$, if $I \models C$ for every clause $C \in P$.

That is: an interpretation I is a model of a program P if it is compatible with all the ground instances of the clauses of P.

We call a model I of a program P a *minimal model*, if there is no other model J of P such that $J \subset I$. Although, Normal logic programs can have multiple minimal models for a program, it is true that Definite Logic Programs only have one minimal model (Eiter *et al.*, 2009).

4.1.5 Loop Formulas

Definition 4.1.5 (Loops)**.** Let P be a normal logic program. Any nonempty subset L of the atoms of P is called a *loop* if for any two atoms p, q in L, there is a path in the dependency graph of P from p to q of length greater than zero.

We also associate two sets of rules with a loop L:

- The set of rules $R^+(L, P)$ contains rules of P whose heads and bodies are in L; and
- The set of rules $R_L^-(L, P)$ contains rules about atoms in L that are out of the loop L.

Definition 4.1.6 (Loop Formulas)**.** Let P be a normal logic program and L be a loop of P. Then, the *loop formula* $LF(L, P)$ is the following implication:

$$\bigvee_{p \in L} \neg p \subset \neg \bigwedge_{r \in R-(L,P)} \text{body}(r),$$

where $\text{body}(r)$ is the body of the rule r.

The main idea behind loop formulas is that they can be used to transform logic programs under stable model semantics to propositional theories. This is particularly useful when these propositional theories are fed into SAT solvers to compute stable models. The entire process of translating a logic program by this approach includes: using Clark's completion to create a propositional theory and augment it by using additional loop formulas, that guarantees that this theory admits only stable models (Lin and Zhao, 2004). One problem with this approach is that the number of loop formulas must be controlled in some way, because there can be an exponential number of loop formulas for a given program (Eiter *et al.*, 2009).

4.2 Answer Set Programming

Finally, after defining the main semantics for logic programs that include negation, we introduce the concept of ASP, an extension of normal logic programming that allows for the use of: integrity constraints, strong negation, disjunctive rules, and choice rules (Eiter *et al.*, 2009; Mauá and Cozman, 2020).

4.2.1 Extended Logic Programs

We start this ASP definition by first introducing the concept of Extended Logic Programs (ELP) and Extended Disjunctive Logic Programs (EDLP), programs that use the three extensions defined above.

Integrity Constraints

Integrity constraints are a way of checking admissibility of models, by adding rules that must be satisfied by all models. These constraints can be written as a rule of the form

```
:- b1, ..., bM, not c1, ..., not cN.
```

But, this rules without heads can also be written with heads when using auxiliary predicates, such as

```
falsity :- b1, ..., bM, not falsity, not c1, ..., not cN,
```

where the auxiliary $falsity$ is a fresh propositional atom (Eiter *et al.*, 2009).

Strong Negation

Strong negation is not a necessity for the definition of ELP, because one can emulate this concept by using integration constraints and (default, also called *weak*) negation. But, the introduction of this concept is at least enlightening, because it allows to encode knowledge that something is known to be false.

Usually, this notion of knowing that a is false is denoted by $\neg a$. But, in the context of ASP, the common notation is $-a$.

Extended Logic Programs

By combining normal logic programs with integrity constraints and strong negation, we are capable of defining ELP:

Definition 4.2.1 (Extended Logic Programs)**.** An extended logic program P is a finite set of rules of the form:

```
a :- b1, ..., bM, not c1, ..., not cN,
```

where $M, N \geq 0$; a, b_i, c_i are atoms or *strongly negated* atoms of a FOL.

Since we showed how one could represent integrity constraints by using auxiliary predicates, and that strong negation can be described by using default negation in combination with integration constraints, there is no need to define a new semantic for this type of logic programs.

The only main "difference" is given by the fact that stable models of ELPs are called ***answer sets***.

Disjunctive Logic Programs

The last extension that we define for normal logic programs is: the addition of disjunctions on the head of rules:

Definition 4.2.2 (Extended Disjunctive Logic Programs)**.** An extended disjunctive logic program P is a finite set of rules of the form:

```
a1 ; ... ; aK :- b1, ..., bM, not c1, ..., not cN,
```

where $K, M, N \geq 0$; a_i, b_i, c_i are atoms or *strongly negated* atoms of a FOL; and the symbol ; denotes disjunction (similar to how commas denote conjunctions).

The semantics for an EDLP can be defined similarly to the ones for an ELP or normal logic program, with two main difference: instead of choosing a stable model M of P (i.e., M being the least model of the reduct P^M), we define the *answer sets* M of P as the *minimal model* of the reduct P^M (since P^M might have multiple minimal models); and there is the need to adjust the completion to suit heads with disjunctions. Completion of Disjunctive Logic Programs, as defined by Alviano, Dodaro, *et al.*, 2016, is a generalization of Clark's completion, where one has to consider

Finally, we define Intepretations for EDLPs, that unifies different types of extensions that lead from (Wielemaker *et al.*, 2012) to the development of Answer Set Programming.

Definition 4.2.3 (Interpretation (of an EDLP))**.** Let P be an EDLP. An interpretation I is a model of:

1. A ground clause C = `a1; ...; aK :- b1, ..., bM, not c1, ..., cN`, denoted $I \models C$, if either $\{ a_1, \ldots, a_K, c_1, \ldots, c_N \} \cap I \neq \emptyset$ or $\{ b_1, \ldots b_M \} \not\subseteq I$;
2. A clause C, denoted $I \models C$, if $I \models C$ for every $C' \in ground(C)$; and
3. A program P, denoted $I \models P$, if $I \models C$ for every clause $C \in P$.

4.2.2 Cardinality Constraints

The last extension that characterizes Answer Set Programming is called *cardinality constraint*, which are constructs of the form: `L \{l1, ..., ln \} U`, that are satisfied whenever the number of satisfied literals l_i is between the integral bounds L and U, inclusive (Syrjänen and Niemelä, 2001). As Syrjänen and Niemelä described, a cardinality constraint in a rule head imposes a non-deterministic choice over the literals in it when the rule body is satisfied.

Choice Rules

An special case of cardinality constraints are *choice rules*: rules where the head is enclosed in brackets and represent the idea that the head can be included in a stable model only if the body holds; but it can be left out, too (Syrjänen and Niemelä, 2001). This type of rule can be expressed through normal rules by introducing a new atom. For example,

Example 4.2.1. *The following program P*

```
{a} :- b, not c.
```

is equivalent to the Normal logic program P':

```
a :- not aa, b, not c.
aa:- not a.
```

References

[Akers 1978] Akers. "Binary decision diagrams". *IEEE Transactions on computers* 100.6 (1978), pp. 509–516 (cit. on p. 4).

[Alviano, Dodaro, *et al.* 2016] Mario Alviano, Carmine Dodaro, *et al.* "Completion of disjunctive logic programs." In: *IJCAI*. Vol. 16. 2016, pp. 886–892 (cit. on p. 20).

[Arora and Barak 2009] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009 (cit. on pp. 4, 5).

[R. Brachman and H. Levesque 2004] Ronald Brachman and Hector Levesque. *Knowledge representation and reasoning*. Elsevier, 2004 (cit. on p. 15).

[Bryant 1986] Randal E Bryant. "Graph-based algorithms for boolean function manipulation". *Computers, IEEE Transactions on* 100.8 (1986), pp. 677–691 (cit. on p. 4).

[Chakraborty *et al.* 2015] Supratik Chakraborty, Dror Fried, Kuldeep S Meel, and Moshe Y Vardi. "From weighted to unweighted model counting." In: *IJCAI*. 2015, pp. 689–695 (cit. on p. 7).

[Chavira and Darwiche 2008] Mark Chavira and Adnan Darwiche. "On probabilistic inference by weighted model counting". *Artificial Intelligence* 172.6-7 (2008), pp. 772–799 (cit. on pp. 9, 11).

[Chavira, Darwiche, and Jaeger 2006] Mark Chavira, Adnan Darwiche, and Manfred Jaeger. "Compiling relational bayesian networks for exact inference". *International Journal of Approximate Reasoning* 42.1-2 (2006), pp. 4–20 (cit. on pp. 9, 11).

[Cook 1971] Stephen A. Cook. "The complexity of theorem-proving procedures". In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC '71. New York, NY, USA: Association for Computing Machinery, 1971, pp. 151–158. isbn: 9781450374644. doi: 10.1145/800157.805047. url: https://doi.org/10.1145/800157.805047 (cit. on pp. 5, 7, 9).

[Costa *et al.* 2012] Vitor Santos Costa, David Page, Maleeha Qazi, and James Cussens. "Clp (bn): constraint logic programming for probabilistic knowledge". *arXiv preprint arXiv:1212.2519* (2012) (cit. on p. 9).

[Darwiche 2002] Adnan Darwiche. "A logical approach to factoring belief networks". *KR* 2 (2002), pp. 409–420 (cit. on pp. 9, 11).

[Dowling and Gallier 1984] William F Dowling and Jean H Gallier. "Linear-time algorithms for testing the satisfiability of propositional horn formulae". *The Journal of Logic Programming* 1.3 (1984), pp. 267–284 (cit. on p. 16).

[Eiter *et al.* 2009] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. *Answer set programming: A primer*. Springer, 2009 (cit. on pp. 16–19).

[Franco and Martin 2009] John Franco and John Martin. "A history of satisfiability." *HandBook of satisfiability* 185 (2009), pp. 3–74 (cit. on p. 3).

[Gomes *et al.* 2006] Carla P Gomes, Willem-Jan Van Hoeve, and Lucian Leahu. "The power of semidefinite programming relaxations for max-sat". In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2006, pp. 104–118 (cit. on p. 9).

[Kiesel *et al.* 2022] Rafael Kiesel, Pietro Totis, and Angelika Kimmig. "Efficient knowledge compilation beyond weighted model counting". *Theory and Practice of Logic Programming* 22.4 (2022), pp. 505–522 (cit. on pp. 10, 12).

[Kimmig *et al.* 2017] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. "Algebraic model counting". *Journal of Applied Logic* 22 (2017), pp. 46–62 (cit. on pp. 4, 9–12).

[Kolokolov *et al.* 2013] Alexander Kolokolov, Alexander Adelshin, and Darya Yagofarova. "Analysis and solving sat and max-sat problems using an l-partition approach". *Journal of Mathematical Modelling and Algorithms in Operations Research* 12.2 (2013), pp. 201–212 (cit. on p. 9).

[Krentel 1988] Mark W. Krentel. "The complexity of optimization problems". *Journal of Computer and System Sciences* 36.3 (1988), pp. 490–509. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(88)90039-6. URL: https://www.sciencedirect.com/science/Article/pii/0022000088900396 (cit. on p. 6).

[H. J. Levesque 1986] Hector J Levesque. "Knowledge representation and reasoning". *Annual review of computer science* 1.1 (1986), pp. 255–287 (cit. on p. 15).

[H. J. Levesque and R. J. Brachman 1987] Hector J Levesque and Ronald J Brachman. "Expressiveness and tractability in knowledge representation and reasoning 1". *Computational intelligence* 3.1 (1987), pp. 78–93 (cit. on p. 15).

[Levin 1973] Leonid Anatolevich Levin. "Universal sequential search problems". *Problemy peredachi informatsii* 9.3 (1973), pp. 115–116 (cit. on pp. 5, 7, 9).

[Lin and Zhao 2004] Fangzhen Lin and Yuting Zhao. "Assat: computing answer sets of a logic program by sat solvers". *Artificial Intelligence* 157.1-2 (2004), pp. 115–137 (cit. on p. 18).

[Manhaeve *et al.* 2018] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. "Deepproblog: neural probabilistic logic programming". *Advances in neural information processing systems* 31 (2018) (cit. on p. 2).

[Mauá and Cozman 2020] Denis Deratani Mauá and Fabio Gagliardi Cozman. "Complexity results for probabilistic answer set programming". *International Journal of Approximate Reasoning* 118 (2020), pp. 133–154 (cit. on p. 18).

[Sang *et al.* 2005] Tian Sang, Paul Beame, and Henry A Kautz. "Performing bayesian inference by weighted model counting". In: *AAAI*. Vol. 5. 2005, pp. 475–481 (cit. on pp. 9, 11).

[Shannon 1938] Claude E Shannon. "A symbolic analysis of relay and switching circuits". *Electrical Engineering* 57.12 (1938), pp. 713–723 (cit. on p. 4).

[Simon 1988] Imre Simon. "Recognizable sets with multiplicities in the tropical semiring". In: *International Symposium on Mathematical Foundations of Computer Science*. Springer. 1988, pp. 107–120 (cit. on p. 10).

[Stab and Gurevych 2017] Christian Stab and Iryna Gurevych. "Parsing argumentation structures in persuasive essays". *Computational Linguistics* 43.3 (2017), pp. 619–659 (cit. on p. 1).

[Syrjänen and Niemelä 2001] Tommi Syrjänen and Ilkka Niemelä. "The smodels system". In: *International Conference on Logic Programming and NonMonotonic Reasoning*. Springer. 2001, pp. 434–438 (cit. on p. 20).

[Toni and Sergot 2011] Francesca Toni and Marek Sergot. "Argumentation and answer set programming". *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday* (2011), pp. 164–180 (cit. on p. 15).

[Totis *et al.* 2023] Pietro Totis, Luc De Raedt, and Angelika Kimmig. "Smproblog: stable model semantics in problog for probabilistic argumentation". *Theory and Practice of Logic Programming* 23.6 (2023), pp. 1198–1247 (cit. on p. 2).

[valiant 1979] l.g. valiant. "The complexity of computing the permanent". *theoretical computer science* 8.2 (1979), pp. 189–201. issn: 0304-3975. doi: https://doi.org/10.1016/0304-3975(79)90044-6. url: https://www.sciencedirect.com/science/Article/pii/0304397579900446 (cit. on pp. 7, 9).

[Wielemaker *et al.* 2012] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. "Swi-prolog". *Theory and Practice of Logic Programming* 12.1-2 (2012), pp. 67–96 (cit. on p. 20).