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.

Tese de Conclusão de Curso - Programa de Graduação em Ciência da Computação / Instituto de Matemática e Estatística / Universidade de São Paulo.
Bibliografia
Versão original

1. Inteligência Artificial. 2. Mineração de Argumentos. 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

5.1 Knowledge compilation pipeline with grounding, compilation, and inference stages. 27
5.2 Illustrative bipolar argumentation fragment with mutual support and derived defeat chains. 28
5.3 Smooth, decomposable, and deterministic circuit fragment used for PASP inference. 29

List of Tables

3.1 Examples of logical and probabilistic inference tasks that can be modelled as AMC instances, together with their semirings and labelling functions. 9

List of Programs

Contents

1 Introduction **1**

2 Appendix Satisfiability and Proof Theory **3**
2.1 Notation . . . 3
2.2 Satisfiability . . . 4
2.2.1 Satisfiability . . . 5
2.2.2 MAX-SAT . . . 5
2.2.3 Sharp-SAT . . . 6

3 Probabilistic Logical Inference **7**
3.1 From SAT to AMC . . . 7
3.2 Semirings . . . 7
3.3 Algebraic Model Counting . . . 8
3.4 Second Level of Algebraic Model Counting . . . 10

4 Answer Set Programming **13**
4.1 Foundations of Logic Programming . . . 13
4.1.1 Definite Logic Programs . . . 13
4.1.2 Normal Logic Programs . . . 14
4.1.3 Herbrand Universe, Base and Interpretation . . . 14
4.1.4 Grounding of a Logic Program . . . 15
4.1.5 Interpretation of a Logic Program . . . 15
4.1.6 Loop Formulas . . . 16
4.1.7 Immediate Consequence Operator . . . 16
4.1.8 Negation and Stratification . . . 16
4.1.9 Stable Model Semantics . . . 17
4.1.10 Reasoning . . . 18
4.2 Answer Set Programming . . . 18
4.2.1 Extended Logic Programs . . . 18

4.2.2 Cardinality Constraints 20
4.3 Probabilistic Logic Programming 20
4.3.1 Probabilistic Answer Set Programming 21
4.3.2 PASP Inference as 2AMC 23

5 Contribution **25**
5.1 Scope and Roadmap 25
5.2 Problem Setting 25
5.2.1 From Argumentation Mining to PASP 25
5.2.2 Knowledge Compilation Pipeline 26
5.3 Foundational Building Blocks 26
5.3.1 Argumentation Frameworks 26
5.3.2 Knowledge Compilation 27
5.4 Experimental Results 29
5.4.1 Experimental Outlook 29
5.4.2 Expected Outcomes 31
5.4.3 Empirical Analysis 31

6 Conclusion **33**

References **35**

Chapter 1

Introduction

Argumentation is a fundamental aspect of human communication, surfacing in contexts that range from informal discussions to legal reasoning. Beyond persuasion, argumentative exchanges support critical thinking, collective decision-making, and the construction of shared knowledge. Understanding how arguments are structured, how they interact, and how they can be evaluated is therefore a central challenge for both the social sciences and Artificial Intelligence (AI).

Recent advances in argumentation mining attempt to automatically detect the building blocks of argumentative discourse (claims, premises, supports, and attacks) directly from text (Stab and Gurevych, 2017). The quality of these pipelines depends on coordinating several interdependent subtasks: deciding whether a span of text is argumentative, classifying its role, and predicting how it relates to the rest of the discourse. Errors in the early stages easily propagate to later ones, eroding the coherence of the recovered argumentative graph. Neuro-symbolic methods are especially attractive in this setting because they combine neural networks, which excel at processing unstructured language, with symbolic reasoning, which enforces global consistency and domain constraints.

The foundational joint model of Stab and Gurevych (2017) couples local classifiers with Integer Linear Programming (ILP) constraints that capture the shape of well-formed argumentative structures. ILP-based approaches improved consistency over pipeline baselines, yet they struggle to represent uncertainty and incur a combinatorial cost when the search space grows. Probabilistic Logic Programming (PLP) systems, such as ProbLog (Fierens *et al.*, 2015), mitigate these limitations by natively handling stochastic information while retaining the declarative and explainable nature of logic programs. Frameworks like DeepProbLog (Manhaeve *et al.*, 2018) and its application to argumentation mining (Cerveira do Amaral *et al.*, 2023) demonstrate how neural predictions can be wired into symbolic models that reason about argumentative structure.

Extending this line of work, smProbLog adopts the stable model semantics of Answer Set Programming (ASP), unlocking the ability to represent negative cycles and other non-monotonic phenomena that occur in real-world argumentation (Totis *et al.*, 2023). Stable models are expressive enough to encode the bipolar argumentation frameworks we study in this thesis, but their probabilistic extensions quickly lead to intractable inference when treated naïvely.

This dissertation examines how probabilistic ASP (PASP) can be compiled into structured probabilistic circuits that support tractable inference and learning. By embedding argumentative constraints into compiled circuits we seek to align neural predictions with the semantics of bipolar argumentation, enabling transparent reasoning, calibrated uncertainty, and scalable training loops. The central questions we explore are:

- How can argumentation mining tasks be encoded as PASP programs that capture the mutual influence of claims, premises, supports, and attacks while preserving probabilistic semantics for uncertain or conflicting evidence?
- Which knowledge compilation strategies can translate these PASP encodings into circuits that admit efficient marginal and conditional inference?
- How can the resulting circuits interface with neuro-symbolic learning, allowing gradients or other learning signals to flow through compiled structure without sacrificing explainability?

The remainder of this document is organized as follows. Chapters chapter 2, chapter 3, and chapter 4 review background on propositional satisfiability, algebraic model counting, and logic programming with stable models. Chapter chapter 5 then introduces our PASP encoding of bipolar argumentation, the knowledge compilation pipeline that supports it, and the circuit properties required for efficient inference. We later position our approach with respect to the literature and discuss the experimental questions that guide our evaluation.

Chapter 2

Appendix Satisfiability and Proof Theory

Logic revolves around two tightly linked notions: *consistency* and *validity* (Franco and Martin, 2009). Proof theory studies these notions from a syntactic perspective by manipulating symbols using axioms and inference rules; model theory examines them semantically by interpreting sentences over structures that encode the “world”. Although the two views emphasise different objects, they coincide for complete axiom systems: a statement that can be derived syntactically is precisely one that is true in every model, and unsatisfiable theories are those that entail a contradiction.

The bridge between proof theory and model theory is central to this thesis. Efficient algorithms for propositional satisfiability (SAT) allow us to reason about the models of logic programs, while proofs about circuit properties ensure that the compiled artefacts faithfully capture the semantics of the original theories. Shannon’s connection between Boolean logic and circuits (Shannon, 1938) sparked a line of work that culminated in modern knowledge compilation techniques, such as Binary Decision Diagrams (BDDs) (Akers, 1978; Bryant, 1986) and Sentential Decision Diagrams (SDDs). These structures will later support our probabilistic Answer Set Programming encodings.

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

Propositional satisfiability (SAT) is one of the cornerstones of theoretical computer science. Every problem in NP polynomially reduces to SAT, which establishes its NP-completeness (Cook, 1971; Levin, 1973). Because of this universality, SAT provides a convenient target for reasoning about diverse logical theories. Later chapters will rely on SAT when translating PASP encodings into propositional form so that we can leverage mature SAT and model counting machinery.

In neuro-symbolic settings we seldom solve SAT just once; instead we evaluate many related queries that differ by evidence or parameter values. This calls for solving SAT-like problems in a way that enables reuse, motivating the knowledge compilation techniques studied throughout this thesis. We therefore start by recalling baseline notions of satisfiability and its counting and optimization variants.

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

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

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

Max-SAT generalises satisfiability by searching for the assignment that optimises how many clauses evaluate to true (Krentel, 1988). The classical (unweighted) variant simply counts satisfied clauses.

Definition 2.2.2 (MAX-SAT)**.** Let ϕ be a CNF whose clause set is C. The Max-SAT objective is

$$\max_{a \in \{0,1\}^n} \sum_{c \in C} [\![c(a)]\!],$$

where $[\![c(a)]\!]$ is the Iverson bracket that yields 1 when clause c evaluates to *True* under assignment a and 0 otherwise, and n is the number of variables in ϕ.

We can additionally attach a non-negative weight $w(c)$ to each clause and seek the maximum weighted sum of satisfied clauses.

Definition 2.2.3 (Weighted MAX-SAT)**.** The weighted Max-SAT problem is

$$\max_{a \in \{0,1\}^n} \sum_{c \in C} w(c)\, [\![c(a)]\!].$$

SAT is a special case of Max-SAT: a CNF is satisfiable iff the optimum equals $|C|$. Consequently, Max-SAT is NP-hard. Setting all weights to 1 reduces the weighted variant to the unweighted one, confirming that the weighted problem inherits NP-hardness. Krentel further shows that Max-SAT is OptP-complete, placing it alongside optimisation problems such as TSP and Knapsack (Krentel, 1988).

2.2.3 Sharp-SAT

Unlike SAT and Max-SAT, the $\#SAT$ (Sharp-SAT) problem is a counting task: it asks how many satisfying assignments a CNF admits (VALIANT, 1979). This places $\#SAT$ closer to the counting problems studied by Levin (LEVIN, 1973) than to Cook's decision formulation (COOK, 1971): there is always an answer, but enumerating it may require exploring all assignments rather than stopping after the first model is discovered.

Counting solutions is at least as hard as deciding their existence. Every SAT instance can be solved by verifying whether the corresponding $\#SAT$ count is non-zero, so $\#SAT$ is NP-hard. Valiant further proved that it is #P-complete, meaning that any counting problem associated with an NP decision problem can be reduced to $\#SAT$.

Definition 2.2.4 (#SAT)**.** Formally, let ϕ be a CNF over variables $x_1, \dots, x_n$. The Sharp-SAT problem asks for the quantity

$$\#SAT(\phi) = \left|\{a \in \{0,1\}^n \mid \phi(a) = True\}\right|.$$

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 $x_1, \dots, x_n$, 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

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

Weighted model counting subsumes the previous variants: setting all weights to 1 recovers model counting, while restricting weights to $\{0,1\}$ captures ordinary SAT. In Chapter chapter 3 we will further generalise this view using algebraic structures that allow us to express probabilistic inference and other reasoning tasks within a single framework.

Chapter 3

Probabilistic Logical Inference

Chapter chapter 2 introduced SAT and its counting and optimisation variants. Weighted model counting (WMC) is especially important for us because it reduces many probabilistic inference tasks to summing weights over satisfying assignments (Adnan Darwiche, 2002; Chavira and Adnan Darwiche, 2008). By encoding, for example, a Bayesian network as a propositional theory whose literals carry probabilities, WMC answers marginal queries through purely logical machinery.

Knowledge compilation builds on this reduction: once a theory has been compiled into a circuit that supports fast WMC, evidence and query evaluation become linear in the circuit size (Chavira, Adnan Darwiche, and Jaeger, 2006; Kimmig *et al.*, 2017). To reason about PASP we need a framework that encompasses classical SAT, WMC, and other algebraic tasks. Algebraic Model Counting (AMC) provides precisely that unifying perspective (Kimmig *et al.*, 2017). A recent extension, second level AMC (2AMC), further captures probabilistic ASP inference (Kiesel, Totis, *et al.*, 2022). This chapter introduces these abstractions and prepares the ground for the compilation pipeline described later.

3.1 From SAT to AMC

Before defining AMC we recall the algebraic structure that underpins it. Generalising from simple numeric addition and multiplication allows us to model counting, optimisation, and probability computations within the same template.

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 on $\mathcal{A}$, $\oplus$ is commutative, $\otimes$ distributes over $\oplus$, $e_\oplus$ and $e_\otimes$ are neutral elements for $\oplus$ and $\otimes$, respectively, and $e_\oplus$ annihilates multiplication: $e_\oplus \otimes a = a \otimes e_\oplus = e_\oplus$ for every $a \in \mathcal{A}$. In a *commutative* semiring, multiplication $\otimes$ is also commutative.

Semirings appear throughout computer science. The *tropical* semiring, for instance, replaces addition with the minimum operator and multiplication with the standard addition over extended real numbers (Simon, 1988). When used with AMC it captures optimisation tasks like Max-SAT: summations aggregate clause scores while products accumulate costs along a derivation.

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 labelling 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 construction above subsumes many 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 mapping every literal to *true* through α, the AMC expression collapses to Boolean reasoning and therefore decides SAT. Moreover, by choosing

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

and mapping both positive and negative literals to 1, the resulting AMC computation counts how many models satisfy the theory.

Similarly, by only changing the set $\mathcal{A}$ of the semiring to $\mathbb{R}_{\geq 0}$ (non-negative real numbers) and letting α assign literal weights in $\mathbb{R}_{\geq 0}$, the AMC problem recovers weighted model counting.

Moreover, the results cited previously about Bayesian networks imply that AMC also captures probabilistic inference (Adnan Darwiche, 2002; Chavira, Adnan Darwiche, and Jaeger, 2006; Chavira and Adnan Darwiche, 2008; Sang *et al.*, 2005). The algebraic structure makes the connection explicit: using the same semiring and defining $\alpha(v) \in [0, 1]$ as the probability of literal v, with $\alpha(\neg v) = 1 - \alpha(v)$ for binary variables, yields the desired probability mass assignments.

Example 3.3.1. *Consider the theory* $T = (a \vee b)$ *and the probability semiring* $\left(\mathbb{R}_{\geq 0}, +, \times, 0, 1\right)$. *Let the labelling function assign weights* $\alpha(a) = 0.6$, $\alpha(\neg a) = 0.4$, $\alpha(b) = 0.3$, *and* $\alpha(\neg b) = 0.7$.

The models that satisfy T are all assignments except $(\neg a, \neg b)$. Applying the AMC formula yields

$$A(T) = 1 - \alpha(\neg a)\alpha(\neg b) = 1 - 0.4 \times 0.7 = 1 - 0.28 = 0.72,$$

which coincides with the probability that at least one of a or b is true. This simple computation mirrors the WMC task we will perform on compiled circuits.

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 relative to a given dataset. The elements of the respective expectation semiring are tuples of the form (p, v), where $p \in [0, 1]$ is the probability of traversing a particular arc and $v \in \mathbb{R}$ is the value contributed by that arc. The operations $\oplus, \otimes$, neutral elements $e_\oplus, e_\otimes$, and a suitable labelling function are defined as follows:

$$\begin{aligned}
(p_1, v_1) \oplus (p_2, v_2) &= (p_1 + p_2, v_1 + v_2), \\
(p_1, v_1) \otimes (p_2, v_2) &= (p_1 \cdot p_2, p_1 \cdot v_2 + p_2 \cdot v_1), \\
e_\oplus &= (0, 0), \\
e_\otimes &= (1, 0), \\
\alpha(a) &= (p_a, p_a \cdot v_a),
\end{aligned}$$

where each arc a carries probability p_a and reward v_a.

Adapted from (Kimmig *et al.*, 2017), Table 3.1 summarises AMC instances for different logical and probabilistic tasks, detailing the underlying sets, operations, and labelling functions for each semiring.

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 modelled as AMC instances, together with their semirings and labelling functions.*

The semirings associated with SAT and probabilistic inference are often called the *Boolean* and the *probability* semirings, respectively.

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, respectively), and *OBDD$_<$ construction* (Kimmig *et al.*, 2017).

3.4 Second Level of Algebraic Model Counting

A natural generalisation of AMC introduces a third operation alongside $\oplus$ and $\otimes$. The resulting framework, called 2AMC, captures tasks such as Maximum a Posteriori (MAP) inference and is essential when modelling PASP problems, as we will see in Chapter 4.

We now recall the definition of the 2AMC problem (Kiesel, Totis, *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, (X_I, X_O);
- Two commutative semirings $S_j = (\mathcal{A}_j, \oplus_j, \otimes_j, e_{\oplus_j}, e_{\otimes_j})$, for $j \in \{I, O\}$;
- Two labelling functions $\alpha_j : \mathcal{L}(X_j) \to \mathcal{A}_j$, for $j \in \{I, O\}$, mapping literals over 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:

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

where $\mathcal{A}(X)$ denotes the set of assignments to the variables in X, and $\mathcal{M}(T|\mathbf{a})$ denotes the set of models of T consistent with assignment $\mathbf{a}$.

It is easy to see that AMC is a special case of 2AMC. By choosing an empty outer partition ($X_O = \emptyset$) and letting the weight transformation be the identity, the outer summation collapses and we recover the original AMC expression. Intuitively, 2AMC first enumerates assignments to the *outer* variables and re-weights the result of an *inner* AMC computation on the remaining variables before aggregating the outcomes.

A canonical example is Maximum a Posteriori (MAP) inference in probabilistic models. Let T be a propositional theory over variables $\mathcal{V}$, partition its atoms into queries Q, evidence E, and the remaining hidden variables R. Given evidence e, MAP seeks the most likely assignment to Q:

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

where $\mathcal{A}(R)$ enumerates assignments to R. Within 2AMC we set $X_O = Q$ and $X_I = R \cup E$. The inner AMC sums over the hidden variables for a fixed choice of q and e, while the outer semiring combines these values with a maximisation operator that selects the best query assignment. The weight transformation t merely injects the inner probability mass into the outer semiring so that maximisation can be performed.

To complete the specification we instantiate the two semirings and the weight transformation. The inner semiring is the standard probability semiring $S_I = ([0,1], +, \times, 0, 1)$ with a labelling function α_I that assigns unit weight to evidence literals consistent with e and uses the distributional parameters of T for the remaining atoms. Inconsistent literals receive weight 0, ensuring that only models compatible with the evidence contribute to the sum.

The outer semiring performs maximisation. We use $S_O = (\mathbb{R}_{\geq 0} \times 2^{|Q|}, \oplus_{\text{argmax}}, \otimes_{\text{argmax}}, (0, \emptyset), (1, \emptyset))$, where

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

breaking ties by a fixed lexicographic order on assignments, and $(p_1, q_1) \otimes_{\text{argmax}} (p_2, q_2) = (p_1 \cdot p_2, q_1 \cup q_2)$. The labelling α_O associates each literal with its local probability and the singleton set containing that literal. Finally, the weight transformation $t(p) = (p, \emptyset)$ carries the inner probability mass into the outer semiring so that the maximisation can compare different assignments. With these choices, 2AMC reproduces MAP inference exactly.

Chapter 4

Answer Set Programming

Knowledge Representation (KR) research is driven by a familiar tension: expressive formalisms capture rich domains but typically incur intractable inference, while restricted languages admit efficient reasoning at the cost of what can be expressed. Full first-order logic, for example, allows us to encode complex knowledge bases but entailment in this setting is undecidable in general and, even for decidable fragments, often demands exponential resources (Levesque, 1986). Despite numerous refinements, classical resolution-based procedures inherit this worst-case behaviour.

One way to regain tractability is to move to less expressive yet carefully designed fragments. Propositional logic exemplifies this strategy: by grounding away quantifiers we can apply SAT technology, as discussed in Chapter 2. Answer Set Programming (ASP) follows the same philosophy. It restricts syntax in a way that keeps many modelling conveniences—default negation, recursion, and non-monotonic reasoning—while providing a semantics that supports efficient solver implementations. ASP has become a workhorse for applications ranging from planning to computational argumentation (Toni and Sergot, 2011).

This chapter reviews the logical foundations required for the probabilistic ASP encodings that we adopt later. We begin with definite programs and the associated semantic machinery, introduce stable-model semantics through loop formulas and related constructs, and then extend the language with features such as disjunction and cardinality constraints. These ingredients will be essential when we encode bipolar argumentation in Chapter 5.

4.1 Foundations of Logic Programming

4.1.1 Definite Logic Programs

Before diving into the semantics of Probabilistic Answer Set Programming (PASP), it is important to understand less expressive formalisms, such as *definite* and *propositional* programs. These formalisms underpin Answer Set Programming (ASP): they reveal the inherent complexity of the problems ASP can solve and guide the design of efficient algorithms. We therefore begin by defining the simplest form of logic programming, namely *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, Ianni, *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.

Although we could omit propositional programs, they are instructive from a complexity perspective: deciding satisfiability for propositional (Horn) programs is P-complete, yet it admits linear-time algorithms (Dowling and Gallier, 1984).

4.1.2 Normal Logic Programs

To reason about non-monotonic phenomena we allow default negation in rule bodies, obtaining *normal* programs:

Definition 4.1.2 (Normal Logic Program)**.** A normal logic program is a finite collection of rules written as

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

Here $M, N \geq 0$, each a, b_i, and c_i is an atom over a function-free first-order language, and the keyword `not` denotes negation as failure ($\neg c$). We refer to the b_i as positive subgoals and to the literals `not` c_i as negative subgoals. Empty bodies are omitted, yielding facts, and empty heads correspond to integrity constraints discussed later.

4.1.3 Herbrand Universe, Base and Interpretation

The symbols occurring in a program determine a canonical universe over which we evaluate rules. The two standard constructions are the *Herbrand universe* and the *Herbrand base.*

Definition 4.1.3 (Herbrand Universe)**.** For a logic program P, the Herbrand universe $HU(P)$ is the set of all ground terms that can be assembled from the constants and function symbols appearing in P (with respect to the background language L). The *Herbrand base* $HB(P)$ consists of every predicate symbol of P applied to tuples of terms from $HU(P)$. Any subset $I \subseteq HB(P)$ is a Herbrand interpretation; its members are precisely the ground atoms that I designates as true.

Example 4.1.1. *Consider the program P from* Eiter, Ianni, *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).
```

The constants $\{0, a, b, r\}$ *generate infinitely many ground terms through the unary symbol* f*, yielding a universe that contains* $\{0, a, b, r, f(0), f(a), f(b), f(r), f(f(0)), \ldots\}$*. Instantiating the predicates* $h, t,$ *and* p *with tuples drawn from* $HU(P)$ *forms* $HB(P)$*; for example,* `p(0,0,0)`*,* `h(f(0),f(0))`*, and* `t(a,a,a)` *are all included.*

Typical Herbrand interpretations include $\emptyset$*, the full base* $HB(P)$*, or intermediate sets such as* $\{h(0, 0), t(a, b, r), p(0, 0, b)\}$*. The empty set is inconsistent with* P *because it disregards the program's facts.*

4.1.4 Grounding of a Logic Program

Grounding replaces variables with ground terms taken from $HU(P)$. Eiter, Ianni, *et al.*, 2009 formalise the construction as follows.

Definition 4.1.4 (Grounding)**.** Let C be a clause in a program P. A ground instance of C is obtained by applying a substitution

$$\theta : \mathrm{Var}(C) \to HU(P).$$

The set of all such instances is denoted ground(C). The grounding of P is the union $\mathrm{ground}(P) = \bigcup_{C \in P} \mathrm{ground}(C)$, comprising every clause produced in this way.

4.1.5 Interpretation of a Logic Program

Ground programs are evaluated with respect to Herbrand interpretations.

Definition 4.1.5 (Interpretation)**.** Let P be a logic program. An interpretation $I \subseteq HB(P)$ is said to satisfy:

- A ground rule $C =$ `a :- b1, ..., bM, not c1, ..., not cN` when either one of $a, c_1, \ldots, c_N$ belongs to I or not all of the atoms $b_1, \ldots, b_M$ are contained in I;
- A (possibly non-ground) clause C if every ground instance $C' \in \mathrm{ground}(C)$ is satisfied by I; and
- The whole program P when each clause in P is satisfied.

Interpreting I as a model of P therefore amounts to checking compatibility with all ground rules. A model I is *minimal* if no proper subset of I is also a model. Definite programs have a unique minimal model, while normal programs may have several (Eiter, Ianni, *et al.*, 2009).

4.1.6 Loop Formulas

Definition 4.1.6 (Loops)**.** Given a normal logic program P, a non-empty set L of atoms is a *loop* when, for every pair $p, q \in L$, there exists a directed path in the dependency graph of P from p to q of positive length. Associated with L we consider two collections of rules:

- $R^+(L, P)$, comprising the rules in P whose heads and bodies mention only atoms from L; and
- $R^-_L(L, P)$, covering rules whose heads belong to L but whose bodies introduce atoms outside L.

Definition 4.1.7 (Loop Formulas)**.** For a loop L in P, the corresponding loop formula is

$$\bigvee_{p \in L} \neg p \;\rightarrow\; \neg \bigwedge_{r \in R^-_L(L,P)} \mathrm{body}(r),$$

where $\mathrm{body}(r)$ denotes the conjunction of body literals in rule r.

Loop formulas provide additional constraints that eliminate unfounded sets when encoding stable models as SAT problems. In the ASSAT approach (Lin and Zhao, 2004), Clark's completion yields a propositional theory whose models are filtered by iteratively adding loop formulas until only stable models remain. Since the number of loops can be exponential, practical systems must generate these formulas carefully (Eiter, Ianni, *et al.*, 2009).

4.1.7 Immediate Consequence Operator

Fixing notation for semantics, we employ the immediate consequence operator T_P.

Definition 4.1.8 (T_P Operator)**.** For a normal program P, the operator $T_P : 2^{HB(P)} \rightarrow 2^{HB(P)}$ maps a set of ground atoms I to

$$T_P(I) = \left\{ a \;\middle|\; \texttt{a :- b1, ..., bM, not c1, ..., not cN} \in \mathrm{ground}(P),\ \{b_1, \dots, b_M\} \subseteq I,\ \{c_1, \dots, c_N\} \cap I\right.$$

The operator is monotone (Bogaerts and Van den Broeck, 2015). Hence, by the Knaster–Tarski theorem it possesses a least fixed point $lfp(T_P)$, which coincides with the least model of P (Eiter, Ianni, *et al.*, 2009). Iterating T_P from the empty interpretation converges to this fixed point.

4.1.8 Negation and Stratification

Different families of logic programs balance expressiveness and computational cost, but structural aspects also matter—especially in the presence of negation. Negated literals introduce the possibility of multiple minimal models, so we require semantics that clarify how to interpret such programs. Two broad strategies are common:

- Look for a single canonical model. This succeeds for *stratified* programs and is exemplified by the well-founded semantics (Van Gelder *et al.*, 1991), which we do not cover in detail.

- Accept the existence of several models and reason with all of them, which leads to stable-model-based approaches such as ASP.

One can detect stratification by constructing a total order on rule evaluation so that negative literals only depend on strictly earlier strata (Eiter, Ianni, *et al.*, 2009). The notion can be expressed using dependency graphs.

Definition 4.1.9 (Dependency Graph)**.** For a ground program P, the dependency graph $dep(P) = (V, E)$ contains a vertex for every ground atom in P. There is a directed edge (v, w) when v occurs in the head of a rule whose body mentions w. If w appears negated, we annotate the edge with a mark $*\ (v, w)$ to indicate a negative dependency. A program is acyclic when this graph has no directed cycles.

Using the graph we formalise stratification.

Definition 4.1.10 (Stratification)**.** Let $G = (V, E)$ be the dependency graph of P. A stratification is a partition $\Sigma = \{S_1, \dots, S_n\}$ of the predicates of P into non-empty, pairwise disjoint sets satisfying:

1. If $(v, w) \in E$ is a positive edge with $v \in S_i$ and $w \in S_j$, then $i \leq j$; and
2. If $*\ (v, w) \in E$ is a negative edge, $v \in S_i$, and $w \in S_j$, then $i > j$.

Whenever such a partition exists we call P *stratified*, and the sets S_i are the program's strata.

4.1.9 Stable Model Semantics

As noted earlier, normal programs need not possess a unique minimal model once negation is allowed. The following example illustrates this behaviour.

Example 4.1.2. *Consider the program*

```
researcher(computability).
machine(X) :- researcher(X), not lambda(X).
lambda(X) :- researcher(X), not machine(X).
```

The dependency graph contains a negative cycle; hence the program is not stratified. Two minimal models exist: $M_1 = \{\texttt{researcher(computability)}, \texttt{machine(computability)}\}$ *and* $M_2 = \{\texttt{researcher(computability)}, \texttt{lambda(computability)}\}$.

To attach a semantics to such programs we employ the Gelfond–Lifschitz reduct.

Definition 4.1.11 (Reduct (GL-reduct))**.** Let P be a normal program and I an interpretation. The *reduct* of P with respect to I, written P^I, is constructed by

1. grounding P;
2. discarding every rule whose body contains `not` c with $c \in I$; and
3. removing all remaining negative literals.

Intuitively, the reduct treats atoms in I as true. Any rule whose body assumes their negation becomes unsatisfied and is removed. Rules that rely on negated atoms outside

I retain their head but lose those negated literals. The result P^I is a positive program and therefore has a unique least model $LM(P^I)$. If this least model coincides with I, the interpretation is stable.

Definition 4.1.12 (Stable Model)**.** An interpretation I of a ground normal program P is a stable model when I is the minimal model of the reduct P^I.

Stable models are fixed points of T_P: whenever I is stable, $T_P(I) = I$, though the converse does not hold in general (Eiter, Ianni, *et al.*, 2009). In this chapter we follow ASP terminology and refer to stable models as *answer sets*.

4.1.10 Reasoning

Reasoning tasks for stable models can be organised by increasing difficulty:

1. **Consistency**: determine whether the program admits at least one answer set.
2. **Brave reasoning**: given a ground literal Q, decide if there exists an answer set in which Q holds. Such literals are *brave consequences*.
3. **Cautious reasoning**: given Q, test whether every answer set entails Q. When this happens we say that Q is a *cautious consequence*.

4.2 Answer Set Programming

We now assemble the ingredients required for full Answer Set Programming. ASP extends normal logic programs with integrity constraints, strong negation, disjunctive heads, and cardinality constructs, while retaining stable-model semantics (Eiter, Ianni, *et al.*, 2009; Mauá and Cozman, 2020).

4.2.1 Extended Logic Programs

We begin with *extended logic programs* (ELPs) and *extended disjunctive logic programs* (EDLPs), which enrich normal programs with the new constructs.

Integrity Constraints

Integrity constraints forbid answer sets in which certain combinations of body literals hold. They are written as headless rules:

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

When convenient we introduce an auxiliary atom to encode the same restriction:

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

Here `falsity` is a fresh propositional symbol that never appears elsewhere (Eiter, Ianni, *et al.*, 2009).

Strong Negation

Although strong negation can be simulated using default negation plus integrity constraints, it is convenient to treat explicit falsity as a primitive. In ASP syntax, the assertion that a is provably false is written $-a$, mirroring the classical connective $\neg a$.

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 collection of rules of the form

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

Here $M, N \geq 0$ and every literal a, b_i, or c_i may be either an atom or its strong negation within the underlying first-order language.

Because integrity constraints and strong negation can be reduced to the basic semantics, we do not need a new notion of model: stable models of extended programs remain *answer sets*.

Disjunctive Logic Programs

We further enhance expressiveness by allowing disjunctions in rule heads.

Definition 4.2.2 (Extended Disjunctive Logic Programs)**.** An extended disjunctive logic program P comprises rules

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

with $K, M, N \geq 0$, where every literal a_i, b_i, and c_i may be a strongly negated atom. The semicolon denotes disjunction, contrasting with the comma used for conjunction in rule bodies.

EDLPs inherit stable-model semantics from extended programs. The reduct P^M remains central, though it may now have several minimal models. Moreover, Clark's completion needs a disjunctive generalisation to capture supported models (Alviano, Dodaro, *et al.*, 2016).

To conclude, we recall the notion of interpretation for EDLPs, bridging the language extensions and the semantics used by ASP solvers.

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

1. A ground rule C = `a1; ...; aK :- b1, ..., bM, not c1, ..., not cN` whenever either one of the head literals $a_1, \dots, a_K$ (or any negated literal c_j) belongs to I, or not all positive body atoms $b_1, \dots, b_M$ are in I;
2. A possibly non-ground clause C if I satisfies every ground instance $C' \in \text{ground}(C)$; and

3. The whole program P if each rule of P is satisfied.

4.2.2 Cardinality Constraints

The final construct we consider is the *cardinality constraint* (Syrjänen and Niemelä, 2001). Expressions of the form `L {l1, ..., ln} U` are true whenever exactly between L and U of the literals l_i hold. Placing such a constraint in a rule head enforces a non-deterministic selection of literals once the body is satisfied.

Choice Rules

A prominent instance is the *choice rule*, whose head is written in braces. Intuitively, whenever the body succeeds we may choose to include the head literal in the answer set, but omission remains possible (Syrjänen and Niemelä, 2001). Choice rules can be compiled away by adding auxiliary atoms:

Example 4.2.1. *The program*

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

behaves like the normal program

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

4.3 Probabilistic Logic Programming

To define Probabilistic Answer Set Programming, we first review how logic programs can be enriched to express probability distributions as generative models. Sato's distribution semantics (Sato, 1995) is one of the seminal works in this area: it associates probability to a set of *independent* events in such a way that a unique probability measure is induced over all interpretations of ground atoms (Cozman and Mauá, 2020).

Definition 4.3.1 (Probabilistic Fact)**.** A probabilistic fact pairs an atom A with a probability value α, a notation we abbreviate as $\alpha :: A$ (Cozman and Mauá, 2017).

The atom may contain variables—for instance $\alpha :: r(X_1, \ldots, X_n)$. In that case we interpret the declaration as shorthand for all of its ground instantiations. Moreover, probabilistic facts are required to be syntactically distinct from the heads of program rules: no substitution can turn a fact and a rule head into the same atom.

Under this scheme probabilities enter exclusively through probabilistic facts. Consequently, structural properties of the underlying logic program—such as stratification, acyclicity, or definiteness—carry over to the probabilistic setting.

To describe multi-valued random choices we use *annotated disjunctions*. Rather than assigning probability p to a single fact a (with $1 - p$ on its negation), we distribute mass across several mutually exclusive outcomes $a(1), \ldots, a(k)$ with probabilities $p(1), \ldots, p(k)$. Annotated disjunctions extend PLPs to the EDLP setting by placing probabilistic weights on rule heads.

Definition 4.3.2 (Annotated Disjunctive Rules)**.** In an annotated disjunctive program each rule has the form

```
p1::a(1); ...; pK::a(K) :- b1, ..., bM, not c1, ..., not cN.
```

The non-negative weights $p_1, \dots, p_K$ sum to at most one. The special case of a single annotated atom

```
p::a.
```

can be simulated by `p::a; (1-p)::dummy :- .`, where `dummy` is a fresh atom that never appears elsewhere (Geh *et al.*, 2024a).

4.3.1 Probabilistic Answer Set Programming

We now revisit Sato's distribution semantics with the adaptations necessary for stable-model reasoning. Probabilities are attached to the entire PLP, rather than to individual derivations, which keeps the approach compatible with both stable and well-founded semantics for non-stratified programs (Eiter, Ianni, *et al.*, 2009; Mauá and Cozman, 2020; Geh *et al.*, 2024b).

Definition 4.3.3 (Probabilistic Choice)**.** Suppose a PLP (P, PF) contains n probabilistic facts of the binary form $\alpha :: A$. Each combination of keeping or deleting these facts yields a different deterministic logic program, so there are 2^n possibilities in total (Cozman and Mauá, 2017). We refer to the decision taken for each fact as a *probabilistic choice*, and assume that all such choices are probabilistically independent.

Definition 4.3.4 (Total Choice)**.** A *total choice* θ for (P, PF) selects a subset of the grounded probabilistic facts to be kept; the remaining ground instances are dropped from the program (Cozman and Mauá, 2017). Independence implies that the probability of θ is simply the product of the individual contributions: each retained fact $\alpha :: A$ contributes α, while each omitted fact contributes $(1 - \alpha)$.

The program determined by θ, written $P \cap PF^{\downarrow\theta}$, is a normal logic program (Cozman and Mauá, 2017; Geh *et al.*, 2024a). In an annotated disjunctive rule, the choice of a head atom effectively selects the corresponding deterministic rule

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

If a total choice θ yields a stratified program $P \cup PF^{\downarrow\theta}$, the induced program has a unique stable model by Section 4.1.9. Sato's semantics therefore extends smoothly to this setting.

More subtle behaviour arises when some total choice leads to a non-stratified program with several stable models. We handle this situation through the credal and MaxEnt semantics introduced below.

First, we define the notion of *consistency* for PLPs:

Definition 4.3.5 (Consistency of Probabilistic Logic Programs)**.** A PLP (P, PF) is *consistent* when every total choice θ induces at least one stable model for $P \cup PF^{\downarrow\theta}$ (Cozman and Mauá, 2017).

Given that we have a definition to express PLPs that are able to have stable models, independent of the total choice, we are capable of formalizing the notion of a *probability model* for PLPs:

Definition 4.3.6 (Probability Model for Probabilistic Logic Programs)**.** A probability model for a consistent PLP (P, PF) is a probability measure $\mathbb{P}$ over interpretations of P satisfying (Cozman and Mauá, 2017):

1. $\mathbb{P}(I) > 0$ only when I is a stable model of the induced program $P \cup PF^{\downarrow\theta}$ for some total choice θ that agrees with I on the probabilistic facts; and
2. The mass assigned to each total choice θ factorises as the product of the probabilities of its selections:

$$\mathbb{P}(\{I \mid I \cap C = C\}) = \prod_{\alpha::A,\, A \in C} \alpha \prod_{\alpha::A,\, A \notin C} (1 - \alpha).$$

Credal Semantics

Our first probabilistic semantics is the *credal* approach, which represents uncertainty through probability intervals (Lukasiewicz, 2005; Lukasiewicz, 2007).

Definition 4.3.7 (Credal Semantics)**.** The credal semantics of a consistent PLP (P, PF) is the set of all its probability models (Cozman and Mauá, 2017). This set is closed and convex, and it coincides with the probability measures that dominate an infinitely monotone Choquet capacity (Cozman and Mauá, 2017; Cozman and Mauá, 2020).

For any collection of interpretations $\mathcal{M}$ we obtain an interval $[\underline{\mathbb{P}}(\mathcal{M}), \overline{\mathbb{P}}(\mathcal{M})]$, where

$$\underline{\mathbb{P}}(\mathcal{M}) = \sum_{\theta \in \Theta\,:\,\Gamma(\theta) \subseteq \mathcal{M}} \mathbb{P}(\theta), \qquad \overline{\mathbb{P}}(\mathcal{M}) = \sum_{\theta \in \Theta\,:\,\Gamma(\theta) \cap \mathcal{M} \neq \emptyset} \mathbb{P}(\theta), \tag{4.1}$$

Θ is the set of all total choices and $\Gamma(\theta)$ contains the stable models associated with θ.

An algorithmic view (Cozman and Mauá, 2017) for computing the interval of a ground query $\mathbf{Q}$ proceeds as follows:

1. Initialise $a = 0$ and $b = 0$.
2. For every total choice θ, enumerate all stable models S of $P \cap PF^{\downarrow\theta}$ and update
   (a) $a \leftarrow a + \mathbb{P}(\theta)$ if $\mathbf{Q}$ holds in every model of S; and
   (b) $b \leftarrow b + \mathbb{P}(\theta)$ if $\mathbf{Q}$ holds in some model of S.
3. Return $[a, b]$ as $[\underline{\mathbb{P}}(\mathbf{Q}), \overline{\mathbb{P}}(\mathbf{Q})]$.

Observe that $\overline{\mathbb{P}}(\mathbf{Q})$ corresponds to brave reasoning, whereas $\underline{\mathbb{P}}(\mathbf{Q})$ relies on cautious reasoning over answer sets (Cozman and Mauá, 2017).

Maximum-Entropy Semantics

Another widely used semantics is the Maximum Entropy (MaxEnt) approach. It distributes probability mass as uniformly as possible across stable models.

Definition 4.3.8 (Maximum Entropy (MaxEnt) Semantics)**.** For each total choice θ, let n be the number of stable models in $\Gamma(\theta)$. The MaxEnt semantics assigns to a stable model $\mathcal{M}$ the probability

$$\mathbb{P}(\mathcal{M}) = \sum_{\theta : \mathcal{M} \in \Gamma(\theta)} \frac{\mathbb{P}(\theta)}{n}.$$

In other words, the probability mass of every total choice is divided equally among the stable models that it induces (Cozman and Mauá, 2017).

4.3.2 PASP Inference as 2AMC

The credal and MaxEnt semantics align naturally with the two-level weighted model counting (2AMC) framework described in Chapter 3. Both can be encoded as 2AMC instances (Kiesel and Eiter, 2023; Azzolini and Riguzzi, 2023). For MaxEnt, the inner counting problem enumerates stable models for each total choice, while the outer aggregation sums their probabilities (Kiesel and Eiter, 2023). For the credal semantics, the inner level checks whether the query holds in some or all stable models—capturing brave or cautious reasoning—and the outer level again aggregates probabilities in a weighted model count (Azzolini and Riguzzi, 2023).

Chapter 5

Contribution

We pursue a neuro-symbolic pipeline for argumentation mining that leverages probabilistic Answer Set Programming (PASP) as an expressive representation language and compiles PASP programs into tractable circuits for efficient inference. By embedding argumentative constraints into compiled circuits we aim to align neural predictions with the semantics of bipolar argumentation, enabling transparent reasoning, calibrated uncertainty, and scalable learning loops.

5.1 Scope and Roadmap

This chapter is organised in four stages. Section 5.2 formalises the problem we study, first by explaining how textual argumentation signals are mapped into PASP programs and then by detailing the knowledge compilation pipeline that supports tractable inference. Section 5.3 gathers the foundational notions we rely on: we introduce the argumentation structures we model, recall Clark's completion, and discuss the circuit properties we require from downstream compilers. The chapter closes with Section 5.4, which anticipates the experimental plan that will ultimately assess the proposed approach.

5.2 Problem Setting

Before delving into the technical building blocks of our approach, we first outline the problem setting we target and summarise the proposed pipeline for scalable neuro-symbolic argumentation mining.

5.2.1 From Argumentation Mining to PASP

Argumentation mining systems must recognise argumentative spans, classify their roles, and recover the interaction graph that ties them together (Stab and Gurevych, 2017). Early neuro-symbolic pipelines coupled local neural predictions with Integer Linear Programming constraints, whereas probabilistic logic programming brought principled uncertainty handling to the task (Fierens *et al.*, 2015; Manhaeve *et al.*, 2018; Cerveira do Amaral *et al.*, 2023). Stable model semantics further broadens the expressiveness of these pipelines by accommodating negative cycles and other non-monotonic phenomena that arise in real discourse (Totis *et al.*, 2023).

We formalise an argumentation mining instance as a bipolar argumentation framework whose nodes represent candidate arguments extracted from text and whose labelled edges denote support or attack relations (Toni and Sergot, 2011; Toni, Potyka, *et al.*, 2023). Neural components provide noisy observations—e.g. span detection probabilities or relational scores—that we encode as probabilistic facts. The deterministic backbone of the PASP program asserts domain constraints, propagates support chains, and enforces well-formed argumentative structures. Inference tasks such as cautious or brave reasoning over accepted arguments then reduce to answering PASP queries, which we cast as instances of second-level algebraic model counting (2AMC) (Kiesel, Totis, *et al.*, 2022).

5.2.2 Knowledge Compilation Pipeline

Directly solving these probabilistic programs on demand would be prohibitively expensive. Instead, we adopt a knowledge compilation workflow that proceeds in three stages: grounding, propositional rewriting, and circuit construction. Grounding produces a finite propositional theory while retaining the structure of the original program; propositional rewriting augments the theory with loop formulas and other constraints that guarantee equivalence under stable-model semantics; and circuit construction compiles the resulting theory into a structured representation that supports efficient 2AMC evaluation.

The compiled artefact supports repeated 2AMC evaluations: evidence and query updates modify only the literal labels, after which the circuit can be re-evaluated in time linear in its size (Kiesel, Totis, *et al.*, 2022).

5.3 Foundational Building Blocks

In this section, we review the foundational concepts that underpin our proposed pipeline, starting with bipolar argumentation frameworks for modelling argumentative structures in PASP, and then discussing the knowledge compilation techniques that enable efficient inference in this setting.

5.3.1 Argumentation Frameworks

Bipolar argumentation frameworks extend Dung's abstract argumentation with explicit support relations that can interact with attacks in non-trivial ways.

Definition 5.3.1 (Bipolar Argumentation Framework)**.** A bipolar argumentation framework (BAF) is a triple $\langle A, R_d, R_s \rangle$ where A is a set of arguments, $R_d \subseteq A \times A$ is the defeat (attack) relation, and $R_s \subseteq A \times A$ is the support relation. Support chains can propagate attacks: a path of supports followed by a defeat induces a derived attack along the support path (Toni and Sergot, 2011; Toni, Potyka, *et al.*, 2023).

Encoding a BAF in PASP lets us reason about accepted arguments under probabilistic stable-model semantics while accounting for uncertainty in the edges or their textual

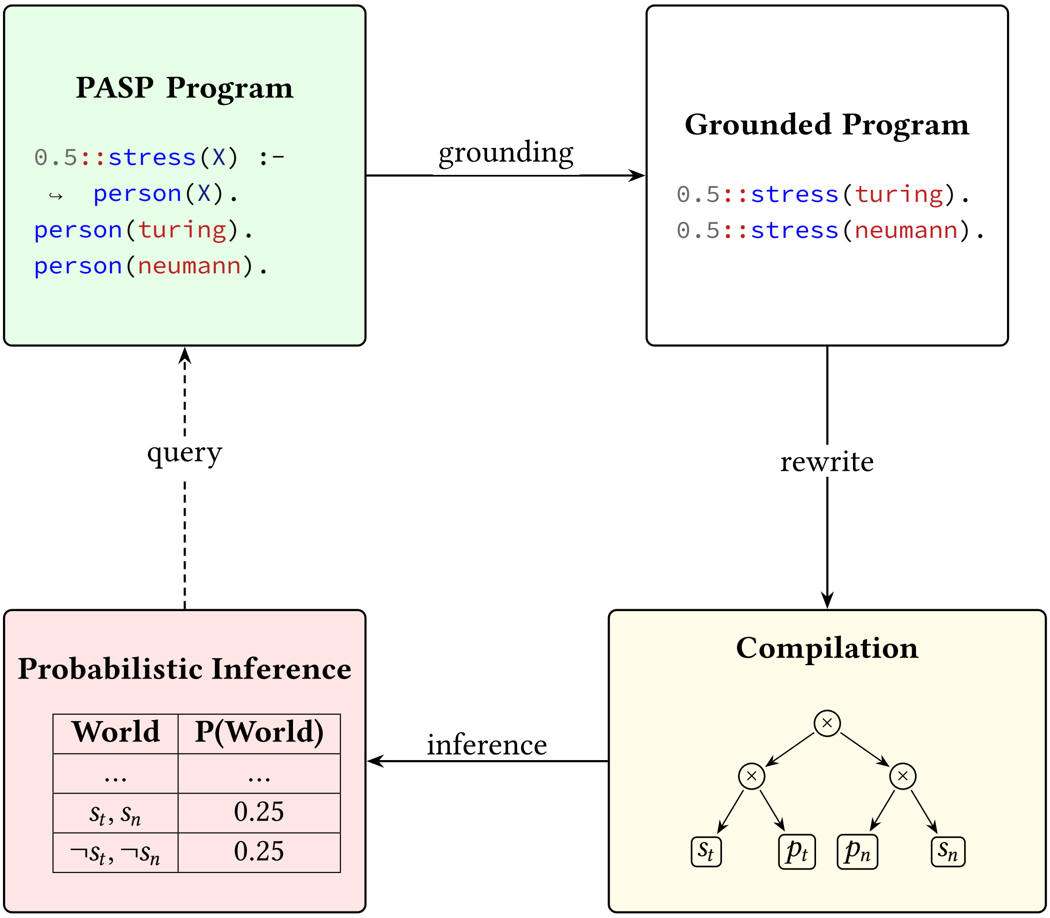

Figure 5.1: *Knowledge compilation pipeline with grounding, compilation, and inference stages.*

evidence.

5.3.2 Knowledge Compilation

Once the PASP program is grounded, we apply Clark's completion and generate a structured circuit that satisfies the properties required for tractable 2AMC inference.

Clark's Completion and CNF Generation

In order to obtain a propositional theory equivalent to the stable-model semantics of the grounded program, we first apply Clark's completion (Clark, 1978). For normal programs, the Clark's completion equates each atom with the disjunction of the bodies of the rules that derive it. For disjunctive programs we obtain

$$\textsc{Clark}(P) = \bigwedge_{a \in \mathcal{A}(P)} \left[a \iff \bigvee_{r \in \mathcal{R}(P,a)} \bigwedge_{b \in \text{body}(r)} b \bigwedge_{a' \in \text{head}(r) \setminus \{a\}} \neg a' \right], \tag{5.1}$$

which yields a conjunctive normal form (CNF) after distributing conjunctions and disjunctions over the finite grounding. Loop formulas supplement the completion to eliminate unsupported models, guaranteeing equivalence with the stable-model semantics (Lin

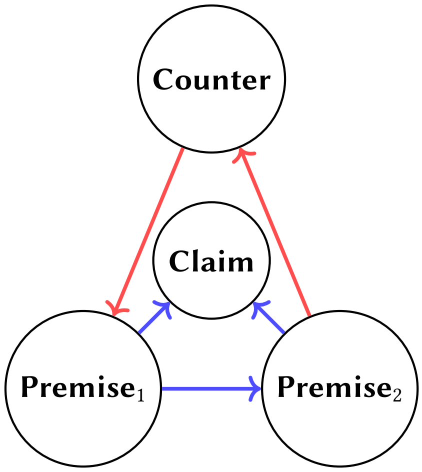

Figure 5.2: *Illustrative bipolar argumentation fragment with mutual support and derived defeat chains.*

and ZHAO, 2004; EITER, IANNI, *et al.*, 2009). Recent analyses tighten this translation by bounding the number of auxiliary variables introduced during completion, which is crucial for predictable compilation cost (EITER, HECHER, *et al.*, 2024).

Circuits

For the purposes of this work, Logic circuits are directed acyclic graphs that represent Boolean functions through internal AND and OR nodes, with literals labelling the leaves. This family of circuits are often called Negation Normal Form (NNF) circuits (Adnan DARWICHE and Pierre MARQUIS, 2002), and there are two key approaches to using them for representing complex boolean formulas: top-down compilation, which first translates the formula into a specific normal form (e.g., CNF) and then compiles it into a circuit; and bottom-up compilation, which constructs the circuit directly from the formula using a set of compilation rules (by applying operations such as conjunction, disjunction, and negation).

To support efficient 2AMC inference, we require circuits that satisfy three main properties that make algebraic model counting tractable (A. DARWICHE and P. MARQUIS, 2002; Adnan DARWICHE, 2011):

- **Decomposability**: sub-circuits combined by a product node refer to disjoint sets of variables.
- **Determinism**: the children of a sum node are mutually exclusive, preventing double counting.
- **Smoothness**: the children of a sum node mention the same variables, enabling weight sharing and differentiation.

Circuits satisfying these properties belong to the sd-DNNF class (smooth, deterministic, decomposable negation normal form). State-of-the-art compilers such as C2D, D4, and SHARPSAT-TD can target this class while preserving the structural guarantees required for efficient 2AMC inference (EITER, HECHER, *et al.*, 2024). Their output supports on-the-fly evaluation of credal and MAXENT semantics through algebraic model counting (WANG *et al.*,

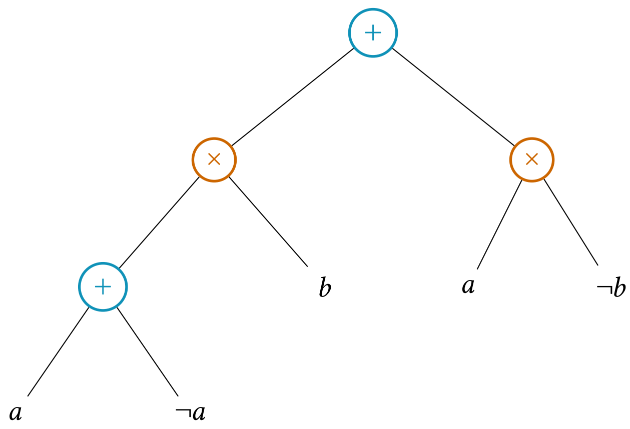

Figure 5.3: *Smooth, decomposable, and deterministic circuit fragment used for PASP inference.*

2025). Moreover, the recent translation scheme of MAENE *et al.* (2024) converts logic circuits into tensor-based representations compatible with automatic differentiation frameworks such as PyTorch or JAX, enabling gradient-based learning over the compiled model.

5.4 Experimental Results

The experimental section of the dissertation will benchmark the proposed pipeline against established sd-DNNF compilers. We plan to evaluate C2D, D4, and SHARPSAT-TD, three top-down compilers adapted by (EITER, HECHER, *et al.*, 2024) in order to handle the necessary constraints for tractable PASP inference as 2AMC problem (WANG *et al.*, 2025). The workloads will include the an argumentation program family, which captures the argumentation instances that motivated our pipeline.

First, we describe the experimental pipeline that will be used to evaluate the selected knowledge compilers on argumentation mining instances. This outlook clarifies that the compiled circuits will be assessed mainly for compilation efficiency, as compiled circuits can be reused for multiple inference tasks during neuro-symbolic learning. Then, we discuss the expected outcomes of the experiments, focusing on the research question that the evaluation aims to address. Finally, we analyse the experimental results obtained from running the proposed pipeline with the selected compilers in argumenatation mining instances ranging from 3 nodes to 50 nodes (a large scale when considering the complexity of exact neuro-symbolic inference).

5.4.1 Experimental Outlook

Program listing. The generator produces instantiations of the following ungrounded PASP template, which encodes argumentation constraints before grounding:

```
% Domain and probabilistic choices over components
node(ARG) :- argument(ARG).
0.5::aux_component(ARG,0) :- node(ARG).
0.5::aux_component(ARG,1) :- node(ARG).
```

```
component(ARG,0) :- aux_component(ARG,0).
component(ARG,1) :- aux_component(ARG,1), not aux_component(ARG,0).
component(ARG,2) :- node(ARG), not aux_component(ARG,1), not
 ↪  aux_component(ARG,0).

% Probabilistic relations between distinct arguments
0.5::relation(SRC,DST) :- node(SRC), node(DST), SRC != DST.

% Stratification constraints
fail :- node(SRC), node(DST), SRC != DST, component(SRC,2), not
 ↪  relation(SRC,DST).
fail :- node(SRC), node(DST), SRC != DST, component(SRC,1), not
 ↪  component(DST,2), not relation(SRC,DST).
fail :- node(SRC), node(DST), SRC != DST, component(SRC,0), component(DST,2),
 ↪  not relation(SRC,DST).
```

Grounding. The grounding step instantiates the above program over a set of arguments provided as input. For an input size of n arguments, the grounded program contains $3n$ probabilistic facts for the components and $n(n-1)/2$ probabilistic facts for the relations, resulting in a total of $3n + n(n-1)/2$ probabilistic facts. Note that the number of rules in the grounded program grows quadratically with n due to the pairwise relations between arguments. In order to compute such groundings, we use *clingo*, the state-of-the-art ASP grounder (Gebser *et al.*, 2014).

Knowledge compilation. Given the grounded program, we apply Clark's completion and generate the corresponding CNF with loop formulas, by following the algorithm present in (Lee and Lifschitz, 2003) (another possible approach would be to follow the algorithm present in (Eiter, Hecher, *et al.*, 2024), which uses the cycle-breaking method from (Eiter, Hecher, *et al.*, 2021)). The resulting CNF is then fed to the selected knowledge compilers, which produce sd-DNNF circuits that support efficient 2AMC inference.

Evaluation metrics. We will assess the performance of the knowledge compilers along two axes: compilation time and circuit size. Compilation time measures the duration required by each compiler to process the CNF and produce the sd-DNNF circuit. Circuit size is evaluated in terms of the number of nodes and edges in the resulting circuit, as well as the memory footprint. These metrics provide insights into the efficiency and scalability of each compiler when handling the argumentation instances generated by our program template.

Neuro-Symbolic Learning Given that the circuits produced by the compilers support efficient 2AMC inference, they do not need to be recompiled when evidence or queries change, i.e. then the probabilities estimated by neural components vary during training. This property enables the integration of the compiled circuits into end-to-end neuro-symbolic learning loops, where neural networks provide probabilistic inputs to the PASP program, and gradients can be backpropagated through the circuit to update the neural parameters. We plan to explore this integration in future work, building upon the tensor-based circuit representations introduced by Maene *et al.* (2024).

5.4.2 Expected Outcomes

The only variable factor in the proposed pipeline is the knowledge compiler, as the grounding and CNF generation steps are deterministic and the PASP program is fixed. Therefore, we expect the experimental evaluation to reveal differences in compilation time and circuit size among the selected compilers. These differences will inform the choice of the most suitable compiler for integrating into neuro-symbolic learning loops for argumentation mining. Ultimately, the goal is to identify a compilation strategy that balances efficiency and scalability, enabling practical applications of probabilistic argumentation mining in real-world scenarios.

Therefore, the only applicable research question for this experimental outlook is: which knowledge compiler among C2D, D4, and SharpSAT-TD produces the most efficient sd-DNNF circuits for argumentation instances, in terms of compilation time and circuit size?

Given the results from (Eiter, Hecher, *et al.*, 2024) on a variety of PASP programs, we expect SharpSAT-TD to outperform the other two compilers in both compilation time and circuit size, due to its advanced top-down compilation techniques, while C2D and D4 may exhibit longer compilation times and larger circuits. However, the actual performance may vary depending on the specific structure of the argumentation instances generated by our program template.

5.4.3 Empirical Analysis

Still pending (results are stored in my desktop and the energy crysis is delaying acess to the CSVs containing the data).

Chapter 6

Conclusion

The proposed neuro-symbolic pipeline for argumentation mining leverages probabilistic Answer Set Programming and knowledge compilation techniques to enable efficient inference over complex argumentative structures. By grounding PASP programs, applying Clark's completion, and compiling the resulting CNF into sd-DNNF circuits, we aim to facilitate scalable reasoning in neuro-symbolic learning loops, with potential applications in various domains requiring robust argumentation analysis.

The proposed methods for efficiente inference using circuits are amenable to integration with gradient-based learning frameworks, paving the way for future work that combines neural networks with symbolic reasoning in a unified architecture.

In our experimental evaluation 5.4.3, we observed that SharpSAT-TD consistently outperformed C2D and D4 in both compilation time and circuit size, across a range of argumentation instances. This finding is aligned with our expectations and previous studies in the literature (Eiter, Hecher, *et al.*, 2024; Kiesel and Eiter, 2023). The superior performance of SharpSAT-TD can be attributed to its advanced top-down compilation techniques, where C2D stands as a robust, but older alternative, and D4 shows competitive performance in certain scenarios but generally lags behind the other two compilers.

These results highlight the importance of selecting an appropriate knowledge compiler for neuro-symbolic applications, specially given an specific domain, such is the case of argumentation mining. The efficiency gains achieved by using SharpSAT-TD can significantly enhance the practicality of neuro-symbolic learning loops, enabling larger and more complex argumentation graphs to be reasoned about effectively. Future work will focus on exploring more classes of argumentations structures, specially those that take advantage of the non-monotonic reasoning capabilities of PASP, and integrating the compiled circuits into end-to-end neuro-symbolic learning frameworks, in order to develop robust argumentation mining systems capable of handling real-world data that current methods struggle with or cannot even model adequately.

References

[Akers 1978] Akers. “Binary decision diagrams”. *IEEE Transactions on computers* 100.6 (1978), pp. 509–516 (cit. on p. 3).

[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. 19).

[Arora and Barak 2009] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009 (cit. on pp. 3–5).

[Azzolini and Riguzzi 2023] Damiano Azzolini and Fabrizio Riguzzi. “Inference in probabilistic answer set programming under the credal semantics”. In: *International Conference of the Italian Association for Artificial Intelligence*. Springer. 2023, pp. 367–380 (cit. on p. 23).

[Bogaerts and Van den Broeck 2015] Bart Bogaerts and Guy Van den Broeck. “Knowledge compilation of logic programs using approximation fixpoint theory”. *Theory and Practice of Logic Programming* 15.4-5 (2015), pp. 464–480 (cit. on p. 16).

[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. 3).

[Cerveira do Amaral *et al.* 2023] Filipe Cerveira do Amaral, H Sofia Pinto, and Bruno Martins. “Argumentation mining from textual documents combining deep learning and reasoning”. In: *EPIA Conference on Artificial Intelligence*. Springer. 2023, pp. 389–401 (cit. on pp. 1, 25).

[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. 6).

[Chavira and Adnan 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. 7, 8).

[Chavira, Adnan 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. 7, 8).

[Clark 1978] Keith L Clark. "Negation as failure". In: *Logic and Databases*. Ed. by Hervé Gallaire and Jack Minker. Springer, 1978, pp. 293–322 (cit. on p. 27).

[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. 4, 6).

[Cozman and Mauá 2017] Fabio Gagliardi Cozman and Denis Deratani Mauá. "On the semantics and complexity of probabilistic logic programs". *Journal of Artificial Intelligence Research* 60 (2017), pp. 221–262 (cit. on pp. 20–23).

[Cozman and Mauá 2020] Fabio Gagliardi Cozman and Denis Deratani Mauá. "The joy of probabilistic answer set programming: semantics, complexity, expressivity, inference". *International Journal of Approximate Reasoning* 125 (2020), pp. 218–239 (cit. on pp. 20, 22).

[A. Darwiche and P. Marquis 2002] A. Darwiche and P. Marquis. "A knowledge compilation map". *Journal of Artificial Intelligence Research* 17 (Sept. 2002), pp. 229–264. issn: 1076-9757. doi: 10.1613/jair.989. url: http://dx.doi.org/10.1613/jair.989 (cit. on p. 28).

[Adnan Darwiche 2002] Adnan Darwiche. "A logical approach to factoring belief networks". *KR* 2 (2002), pp. 409–420 (cit. on pp. 7, 8).

[Adnan Darwiche 2011] Adnan Darwiche. "Sdd: a new canonical representation of propositional knowledge bases". In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011 (cit. on p. 28).

[Adnan Darwiche and Pierre Marquis 2002] Adnan Darwiche and Pierre Marquis. "A knowledge compilation map". *Journal of Artificial Intelligence Research* 17 (2002), pp. 229–264 (cit. on p. 28).

[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. 14).

[Eiter, Hecher, *et al.* 2021] Thomas Eiter, Markus Hecher, and Rafael Kiesel. "Treewidth-aware cycle breaking for algebraic answer set counting". In: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*. Vol. 18. 1. 2021, pp. 269–279 (cit. on p. 30).

[Eiter, Hecher, *et al.* 2024] Thomas Eiter, Markus Hecher, and Rafael Kiesel. "Aspmc: new frontiers of algebraic answer set counting". *Artificial Intelligence* 330 (2024), p. 104109. issn: 0004-3702. doi: https://doi.org/10.1016/j.artint.2024.104109. url: https://www.sciencedirect.com/science/Article/pii/S0004370224000456 (cit. on pp. 28–31, 33).

[Eiter, Ianni, *et al.* 2009] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. *Answer set programming: A primer*. Springer, 2009 (cit. on pp. 14–18, 21, 28).

[Fierens *et al.* 2015] Daan Fierens *et al.* "Inference and learning in probabilistic logic programs using weighted boolean formulas". *Theory and Practice of Logic Programming* 15.3 (2015), pp. 358–401 (cit. on pp. 1, 25).

[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).

[Gebser *et al.* 2014] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. "Clingo= asp+ control: preliminary report". *arXiv preprint arXiv:1405.3694* (2014) (cit. on p. 30).

[Geh *et al.* 2024a] Renato Lui Geh, Jonas Gonçalves, Igor C Silveira, Denis D Mauá, and Fabio G Cozman. "Dpasp: a probabilistic logic programming environment for neurosymbolic learning and reasoning". In: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*. Vol. 21. 1. 2024, pp. 731–742 (cit. on p. 21).

[Geh *et al.* 2024b] Renato Lui Geh, Jonas Gonçalves, Igor C Silveira, Denis D Mauá, and Fabio G Cozman. "Dpasp: a probabilistic logic programming environment for neurosymbolic learning and reasoning". In: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*. Vol. 21. 1. 2024, pp. 731–742 (cit. on p. 21).

[Kiesel and Eiter 2023] Rafael Kiesel and Thomas Eiter. "Knowledge compilation and more with sharpsat-td". In: *Proceedings of the 20th IInternational Conference on Principles of Knowledge Representation and Reasoning*. IJCAI Organization. 2023, pp. 406–416 (cit. on pp. 23, 33).

[Kiesel, Totis, *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. 7, 10, 26).

[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. 3, 7, 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. 5).

[Lee and Lifschitz 2003] Joohyung Lee and Vladimir Lifschitz. "Loop formulas for disjunctive logic programs". In: *Logic Programming: 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003. Proceedings 19*. Springer. 2003, pp. 451–465 (cit. on p. 30).

[Levesque 1986] Hector J Levesque. "Knowledge representation and reasoning". *Annual review of computer science* 1.1 (1986), pp. 255–287 (cit. on p. 13).

[Levin 1973] Leonid Anatolevich Levin. "Universal sequential search problems". *Problemy peredachi informatsii* 9.3 (1973), pp. 115–116 (cit. on pp. 4, 6).

[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 pp. 16, 27).

[Lukasiewicz 2005] Thomas Lukasiewicz. "Probabilistic description logic programs". In: *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*. Springer. 2005, pp. 737–749 (cit. on p. 22).

[Lukasiewicz 2007] Thomas Lukasiewicz. "Probabilistic description logic programs". *International Journal of Approximate Reasoning* 45.2 (2007), pp. 288–307 (cit. on p. 22).

[Maene *et al.* 2024] Jaron Maene, Vincent Derkinderen, and Pedro Zuidberg Dos Martires. "Klay: accelerating neurosymbolic ai". *arXiv preprint arXiv:2410.11415* (2024) (cit. on pp. 29, 30).

[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 pp. 1, 25).

[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 pp. 18, 21).

[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 p. 8).

[Sato 1995] Taisuke Sato. "A statistical learning method for logic programs with distribution semantics" (1995) (cit. on p. 20).

[Shannon 1938] Claude E Shannon. “A symbolic analysis of relay and switching circuits”. *Electrical Engineering* 57.12 (1938), pp. 713–723 (cit. on p. 3).

[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. 8).

[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 pp. 1, 25).

[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, Potyka, *et al.* 2023] Francesca Toni, Nico Potyka, Markus Ulbricht, and Pietro Totis. “Understanding problog as probabilistic argumentation”. *arXiv preprint arXiv:2308.15891* (2023) (cit. on p. 26).

[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 pp. 13, 26).

[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 pp. 1, 26).

[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 p. 6).

[Van Gelder *et al.* 1991] Allen Van Gelder, Kenneth A Ross, and John S Schlipf. “The well-founded semantics for general logic programs”. *Journal of the ACM (JACM)* 38.3 (1991), pp. 619–649 (cit. on p. 16).

[Wang *et al.* 2025] Benjie Wang, Denis Deratani Mauá, Guy Van den Broeck, and YooJung Choi. *A Compositional Atlas for Algebraic Circuits*. 2025. arXiv: 2412.05481 [cs.AI]. url: https://arxiv.org/abs/2412.05481 (cit. on pp. 28, 29).