KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

From Atoms to Possible Worlds

Probabilistic Inference in the Discrete-Continuous Domain

Pedro Miguel Zuidberg Dos Mártires

Supervisor:
Prof. dr. Luc De Raedt

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

November 2020

From Atoms to Possible Worlds

Probabilistic Inference in the Discrete-Continuous Domain

Pedro Miguel Zuidberg Dos Mártires

Examination committee:
Prof. dr. ir. Robert Puers, chair
Prof. dr. Luc De Raedt, supervisor
Prof. dr. Jesse Davis
Prof. dr. ir. Tinne De Laet
Prof. dr. ir. Guy Van den Broeck
(University of California, Los Angeles)
Prof. dr. Amy Loutfi
(Örebro Universitet)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

November 2020

I dedicate this thesis to Antonio and Johanna – my parents.

Acknowledgments

Roughly five years ago I finished my Master's in particle physics. Soon after, I took up the mad challenge of getting accepted into an artificial intelligence PhD program. Needless to say that my academic background did not fit the usual requirements to perform research in computer science. As a consequence, my chances of getting accepted into a PhD program were, unbeknowst to me at the time, rather slim. It rained rejections. My first thank you goes to Luc, my doctoral advisor. For some reason, Luc, after only two video calls between Belgium and Australia, went out on a limb and offered me a PhD position in his lab. I must confess, in retrospect, I would not have given myself a PhD position. Luc did. This thesis would not exist without his leap of faith almost five years ago and his role as PhD supervisor.

A this points I would also like to thank the other members of my examination jury. Amy, Guy, Jesse, Tinne, thank you for reading and commenting on the text of the thesis and thanks to Robert Puers for chairing the defense.

My third big thank you goes to all the colleagues, of which some have become friends. I was lucky to meet you and collaborate with you. Doing a PhD is challenging. I reckon the last fourish years have been the biggest intellectual challenge of my life but also the emotional toll of pursuing a PhD can be taxing on one's state of mind. It would not have been possible without you! For example, I remember fondly our coffee breaks, which sometimes turned into long scientific discussions, stupid shenanigans, or just ramblings about how doing a PhD sucks (sometimes). An important distraction were also our nights out. Often the most difficult thing was to not talk about work and research but just unwind. Strangely, those nights out were more frequent at the beginning of my PhD. I guess I am getting old and bold and it is time to wrap this thing up.

From a scientific point of view I would like to name a couple of people who helped me quite a lot during the PhD and were influential on forming my scientific persona. I would like to thank Anton, who helped me out a lot in the beginning of the PhD and who I could ask any question. And if Anton did not know the answer, Wannes

did. I would also like to thank all the people involved in the ReGROUND project and especially Andreas. Working on the ReGROUND project allowed me to get in touch with a lot of different ideas, which I am still working on (some more actively than others). Additionally, the ReGROUND project paid for some nice trips to Sweden.

I would like to stress that science is a team sport these days and doing research without great and dedicated collaborators is infinitely more difficult. I remember for example some very long nights working together with my collaborators (Andreas, Nitesh, Ozan, Samuel, Vincent) on papers and give up on hours of sleep just to polish a paper a little bit more. Although those were fun experiences, I hope they won't repeat ever again.

I could mention heaps of other people and at least one story that we share. Either on the hallway of the computer science department, on the squash court, at a conference on a remote island 🏝, buying a car, giving exercise sessions, redecorating the lab, sharing a house for three years, laughing about Italian prime ministers speaking English, ... Unfortunately, I procrastinated a little bit too heavily when I was writing this and I do not have a lot of time to write much more.

Finally, I would like to say that doing research is an absolute privilege. Even though it still sounds odd, I can tell people I am a scientist. I also tell them that I am grateful that I can spend my time asking questions about the world and wasting even more time on finding answers that might not exist. This is not self-evident. I am thankful for living in a society that values public education and promotes research.

Abstract

Life is uncertain, full of ambiguities. Paradoxically, only embracing stochasticity, not fighting randomness, lets us cut through the surrounding fog of noise and find meaning. Similarly, machines will only make sense of the world accepting its probabilistic nature.

This dissertation studies probabilistic artificial intelligence, a broad field of research. We will investigate probabilistic AI at three distinct conceptual levels, or three levels of abstraction. Throughout all three levels of abstraction, special focus is given to problems that incorporate discrete and continuous random variables alike – a challenge only embraced by very few. We start our study with logic **atoms** (Boolean variables) from which we build probabilistic logic formulas. A variable instantiation, which satisfies a probabilistic logic formula with probability greater than zero, is also called a **possible world**. In the last two chapters of the thesis these possible worlds will model the real-world observed through a 3D-camera.

1. **Microscopic Level:** The microscopic level studies the marriage of logic and probability theory. We formalize their combination by starting out from logic **atoms**, in the context of *weighted model integration*, from which we then construct entire probabilistic models. We introduce a range of state-of-the-art probabilistic inference algorithms. The presented algorithms are based on *knowledge compilation* and *arithmetic circuits*, and use either *symbolic integration* for exact inference or *Monte Carlo integration* for approximate inference. The algorithms show that probabilistic inference techniques from the purely discrete or the purely continuous domain can be adapted to perform probabilistic inference in the discrete-continuous domain.

2. **Macroscopic Level:** While the microscopic level constitutes a principled approach to expressing probabilistic models, the level of abstraction is rather ill-suited for human users. We introduce DC-ProbLog, a probabilistic logic programming language that allows users to operate at a high-level of abstraction. We show that we can perform inference by mapping back to weighted model integration (the microscopic level).

3. **Cognitive Level:** At the cognitive level we build a *perceptual anchoring* system. Perceptual anchoring solves the problem of creating and maintaining, in time and space, the correspondence between symbols and objects in the real-world. Our system constructs a probabilistic model of the surrounding **world** (perceived through a 3D-camera) and has the capability of probabilistically reasoning about objects that are present. The key contribution is the design of a framework that combines perceptual anchoring and probabilistic programming (macroscopic level). The probabilistic reasoning capacity is useful in situations where objects are not directly observed but occluded by other objects. Occluded objects can then be anchored through probabilistic reasoning.

Beknopte samenvatting

Het leven is onzeker, vol ambiguïteit. Stochasticiteit vermijden brengt ons echter niet dichter bij een oplossing. Enkel door het te omarmen kunnen we doorheen de mist van ruis zien, en de betekenis erin vinden. Gelijkaardig, kunnen machines enkel de wereld begrijpen door de probabilistische natuur ervan te erkennen.

Deze dissertatie bestudeert probabilistische artificiële intelligentie, een breed onderzoeksveld. We zullen probabilistische AI bestuderen op drie verschillende conceptuele niveaus, drie niveaus van abstractie. Doorheen alle drie die niveaus gaat er speciale aandacht naar problemen met zowel discrete als continue willekeurige variabelen – een uitdaging door zeer weinigen aangegaan. We starten onze studie met logische atomen (Boolean variabelen) van waaruit we probabilistische logische formules opbouwen. Een instantie van de variabelen, die voldoet aan een probabilistische logische formule met een kans groter dan nul, wordt ook wel een mogelijke wereld genoemd. In de laatste twee hoofdstukken van de thesis zullen deze mogelijke werelden de reële wereld geobserveerd door een 3D camera, modelleren.

1. **Microscopisch niveau:** Het microscopisch niveau bestudeert het huwelijk van logica enerzijds en probabilistische theorie anderzijds. We formaliseren deze combinatie door te starten vanuit logische atomen, in de context van *w*eighted model integration, waaruit we daarna volledige probabilistische modellen opbouwen. We introduceren een waaier van state of the art probabilistische inferentie algoritmen. De gepresenteerde algoritmen zijn gebaseerd op *k*nowledge compilation en aritmetische circuits, en gebruiken ofwel symbolische integratie voor exacte inferentie of Monte Carlo integratie voor benaderende inferentie. De algoritmen tonen aan dat probabilistische inferentie technieken van de puur discrete of de puur continue domeinen aangepast kunnen worden voor het uitvoeren van probabilistische inferentie in het discreet-continue domein.

2. **Macroscopisch niveau:** Terwijl het microscopisch niveau een principiële aanpak vormt om probabilistische modellen in te uiten, is het niveau van

abstractie echter ongeschikt voor menselijke gebruikers. We introduceren DC-ProbLog, een probabilistische logische programmeertaal die gebruikers toelaat om te opereren op een hoog niveau van abstractie. Inferentie wordt uitgevoerd door het terug afbeelden naar weighted model integration (het microscopisch niveau).

3. **Cognitief niveau:** Op het cognitief niveau bouwen we een perceptueel verankeringssysteem. Perceptuele verankering lost het probleem op van het creëren en onderhouden, in tijd en ruimte, van de overeenkomsten tussen symbolen en objecten in de reëel wereld. Ons systeem construeert een probabilistisch model van de omgevende wereld (waargenomen door een 3D-camera) en heeft de capaciteit om probabilistisch te redeneren over aanwezige objecten. De hoofdzakelijke bijdrage omvat het ontwerp van een framework dat perceptuele verankering combineert met probabilistisch programmeren (macroscopisch niveau). De capaciteit om probabilistisch te redeneren is nuttig in situaties waar objecten niet rechtstreeks observeerbaar zijn maar bedekt door andere objecten. Deze bedekte objecten kunnen worden verankerd via probabilistisch redeneren.

List of Abbreviations

AC Arithemtic Circuit.

AI Artificial Intelligence.

ALW Algebraic Likelihood Weighting.

AMC Algebraic Model Counting.

d-DNNF Deterministic Decomposable Negation Normal Form.

DAG Directed Acyclic Graph.

DD Decision Diagram.

DDC Dynamic Distributional Clause.

DS Distribution Semantics.

FPGA Field Programmable Gate Array.

GPU Graphics Processing Unit.

KC Knowledge Compilation.

MC Monte Carlo.

MCAD Monte Carlo Anti-Differentiation.

MCMC Markov Chain Monte Carlo.

NMC Nested Monte Carlo.

NNF Negation Normal Form.

PLP Probabilistic Logic Programming.

PWM Permanent World Model.

RRMSE Relative Root Mean Squared Error.

RSTD Relative Standard Deviation.

SAT Boolean Satisfiability.

sd-DNNF Smooth Deterministic Decomposable Negation Normal Form.

SDD Sentential Decision Diagram.

SLD Resolution Selective Linear Definite Resolution.

SMT Satisfiability Modulo Theory.

SRL Statistical Relational Learning.

SVM Support Vector Machine.

ToO Theory of Occlusion.

TWM Temporary World Model.

WMC Weighted Model Integration.

WMI Weighted Model Counting.

XADD Extended Algebraic Decision Diagram.

XSDD Extended Sentential Decision Diagram.

Contents

List of Figures

List of Tables

Introduction

Chapter 1

Introduction

1.1 Logic, Probability and Programming

The mathematical discipline of logic constitutes the bedrock of a large body of work in artificial intelligence (AI) research, including sub-disciplines such as expert systems [Jackson, 1998], inductive logic programming [Muggleton; De Raedt, 1994], or more broadly knowledge representation and reasoning [Levesque, 1986]. One of the main shortcomings of logic-based AI systems is that they fail to capture the inherent uncertainty present in the world – aleatoric and epistemic uncertainty alike [Hüllermeier; Waegeman, 2019]. To express such uncertainties, AI researchers have set out to enhance pure logic with probabilities.

As a matter of fact, the marriage of logic and probability is an enduring endeavor of AI research that predates AI itself [Russell, 2015]. Grandees of mathematics, philosophy, and economics such as Leibniz, Peirce, and Keynes, undertook early efforts to this end [Hailperin et al., 1984; Howson, 2003]. In the field of AI, Nilsson [1986] and Pearl [1988] presented seminal works investigating the combination of logic and probability, which heralded the era of *modern AI*: systems were now able to capture the inherent uncertainty present in the world as well as reason about it[1]

Poole [1993] improved upon these ideas and introduced *probabilistic Horn abduction*, which allowed him to later on construct the *Independent Choice Logic* language [Poole, 1997], an expressive universal probabilistic logic programming language (PLP) [De

[1] Interestingly Nilsson later revisited, under the influence of Pearl, his original formulation of probabilistic logic [Nilsson, 1994]. Nevertheless, to the best of our knowledge, Nilsson [1986] was the first one to coin the term. *probabilistic logic* itself, from which a rich tradition of combining logic and probability theory emerged in the field of AI.

RAEDT; KIMMIG, 2015; RIGUZZI, 2018] rooted in logic programming. Subsequently, SATO [1995] built on top of the ideas advanced by POOLE [1993] and introduced the so-called *distribution semantics* (DS) for probabilistic logic programs. After the introduction of DS, a variety of further PLP languages appeared, such as *PRISM* [SATO; KAMEYA, 1997], *Logic Programs with Annotated Disjunctions* [VENNEKENS et al., 2004], and *ProbLog* [DE RAEDT et al., 2007; FIERENS et al., 2015], to name a few.

This thesis is situated in the broader context of the probabilistic logic programming language ProbLog. ProbLog inherits its syntax from the (non-probabilistic) logic programming language Prolog [STERLING; SHAPIRO, 1994]. ProbLog allows one to express, in addition to logic rules and facts, so-called *probabilistic facts*, which are logic facts labeled with the probability of them being satisfied.

Example 1.1. *We model two machines (Line 1 in the program below). We want to know the probability of the first machine working (Line 11), given that the second machine works (Line 10) and given a model that describes under which conditions the machines work (Lines 7 and 8). Additionally the program models the outside temperature (Line 3) and whether the cooling of each machine works (Lines 4 and 5) as (Boolean) random variables (expressed as probabilistic facts).*

```
1  machine(1). machine(2).
2
3  0.8::temperature(low).
4  0.99::cooling(1).
5  0.95::cooling(2).
6
7  works(N):- machine(N), cooling(N).
8  works(N):- machine(N), temperature(low).
9
10 evidence(works(2)).
11 query(works(1)).
```

Running the program yields $p(\texttt{works(1)}|\texttt{works(2)}) \approx 0.998$.

1.2 The Symbolic and the Subsymbolic

The focus of probabilistic logic programming lies on reasoning and probabilistic *inference* and only to a lesser extend on *learning*. The question of how to perform learning in a probabilistic logic setting is tackled by practitioners of *statistical relational learning* (SRL) [GETOOR, 2013; DE RAEDT et al., 2016]. SRL integrates predicate logic with graphical models [KOLLER; FRIEDMAN, 2009] in order to extend the expressive power

of probabilistic graphical models towards relational logic – resulting in probabilistic logics than can handle uncertainty, with a focus on learning. After two decades of research, a plethora of SRL frameworks have emerged, e.g. [Sato; Kameya, 2001; Richardson; Domingos, 2006; Getoor; Taskar, 2007; Fierens et al., 2015].

One obstacle that still lies ahead in the field of SRL is combining symbolic reasoning and learning, on the one hand, with sub-symbolic data and perception, on the other hand (see Gardner et al. [2014] and Beltagy et al. [2016]). The question is how to create a symbolic representation of the world from sensor data in order to reason and ultimately plan in an environment riddled with uncertainty and noise.

In recent years it has become undeniable that the right way to handle high-dimensional raw sensor data, such as images, is by means of deep learning [Goodfellow et al., 2016]. Although exhibiting impressive results, deep models do suffer from certain drawbacks. As opposed to probabilistic rules, it is, for example, not straightforward to include prior (symbolic) knowledge in a neural system, which makes them notoriously data-hungry.

Given these two approaches to artificial intelligence and their respective advantages and disadvantages, researchers have started investigating how to combine symbolic and subsymbolic approaches to AI [Garcez et al., 2019; De Raedt et al., 2020]. This avenue of research was dubbed neuro-symbolic AI. A major recent accomplishment of a neuro-symbolic algorithm is the much publicized win of AlphaGo over Lee Sedol. AlphaGo combines Monte Carlo tree search (a symbolic method) with deep Q-learning (a neural method) [Silver et al., 2016]. While this thesis does not tackle neuro-symbolic AI directly, it can be placed in the broader neuro-symbolic context.

1.3 Thesis Contributions

The themes and questions studied in this thesis can be placed in the broader context of probabilistic logic programming (and to a lesser extend in the realm of neuro-symbolic AI). We study probabilistic programming at three levels of abstraction:

1. the **microscopic level**
2. the **macroscopic level**
3. the **cognitive level**

Microscopic: Weighted Model Integration

Weighted model integration (WMI) is a recently introduced formalism used to perform probabilistic inference in domains that exhibit discrete and continuous random variables [Belle et al., 2015a]. Probability distributions are expressed by means of so-called weighted SMT formulas, which are an extension of weighted Boolean formulas. SMT formulas consist of *atomic* SMT formulas and can be regarded as a microscopic assembly language for probabilistic programming.

Example 1.2. *Consider the SMT formula ϕ.*

$$\phi \leftrightarrow a \wedge (0 < x) \wedge (x < 3) \wedge (0 < y) \wedge (y < 2) \wedge (x < y) \bigvee \tag{1.1}$$

$$\neg a \wedge (0 < x) \wedge (x < 3) \wedge (0 < y) \wedge (y < 2) \wedge (x < y + 1) \tag{1.2}$$

Where a is a Boolean variable, and x and y are real valued variables. Additionally, consider also the following weight functions $w(a) = 0.2$, $w(\neg a) = 0.8$, $w(x) = 2x$ and $w(y) = y^2$. The goal of weighted model integration is then to determine the weight of an SMT formula. For ϕ and the two given weight functions we obtain:

$$\begin{aligned}
w(\phi) &= w(a) \int_0^2 \int_0^y 2xy^2 dxdy + w(\neg a) \int_0^2 \int_0^{y+1} 2xy^2 dxdy \\
&= 0.2 \int_0^2 2\frac{1}{2}(y-0)y^2 dy + 0.8 \int_0^2 2\frac{1}{2}(y+1-0)y^2 dy \\
&= 0.2 \int_0^2 y^3 dy + 0.8 \int_0^2 (y^3 + y^2) dy \\
&= 0.2\frac{1}{3}(8-0) + 0.8\left(\frac{1}{3}(8-0) + \frac{1}{2}(4-0)\right) = 0.2\frac{8}{3} + 0.8(\frac{8}{3} + 2) = \frac{64}{15}
\end{aligned}$$

Note how the inequalities in the SMT formula translate to the bounds of integration when computing the weight of ϕ.

Geometrically, SMT formulas, for which we compute the weight, represent regions in $\mathbb{R}^D$. Weighted model integration then computes the integral of the weight function where the regions give the bounds of integration.

Example 1.3. *Consider the geometric representation of a WMI problem in Figure 1.1. The problem has two continuous random variables (x and y) and three Boolean random variables, which produce the different feasible regions (the red region and the two blue regions). The regions themselves are given by constraints on the continuous variables.*

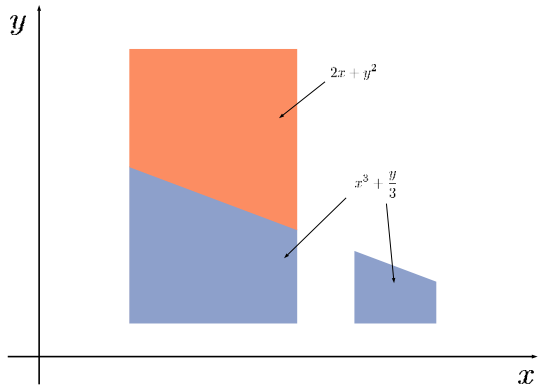


Figure 1.1: Geometric representation of a WMI problem.

Moreover, for each feasible region a weight function is given. Outside of the regions the weight is zero. WMI tackles the problem of computing the integral over the feasible regions.

The first research question of the thesis addresses solving weighted model integration problems.

RQ1: Can we adopt inference algorithms from the purely discrete domain or the purely continuous domain to develop novel WMI solvers?

The **first contribution** of this thesis is a set of novel algorithms that perform probabilistic inference for weighted SMT formulas. To achieve this, we follow the philosophy of extending existing techniques used in the purely discrete domain, such as *knowledge compilation*, and the purely continuous domain, such as *Monte Carlo integration*.

1. **Symbo** and **Sampo** are an exact and an approximate inference algorithm for WMI, respectively. They are the first ones to be based on standard knowledge compilation. Knowledge compilation is a well-established technique to perform probabilistic inference in the purely discrete domain. We show how to extend it to the discrete-continuous domain.

2. **F-XSDD** is a family of solvers that builds on Symbo but exploits additional structure present. A key component to efficient probabilistic inference in

the discrete-continuous domain is the aggressive exploitation of structural regularities in probabilistic inference problems.

3. In order to compute integrals, Symbo and F-XSDD use symbolic integration. For high dimensional integrals this can be prohibitively slow, or even infeasible all together. To remedy this, we developed **Monte Carlo Anti Differentiation** and contribute the F-XSDD(MCAD) inference algorithm. Monte Carlo integration, on which Monte Carlo Anti-Differentiation is based, is a widely used technique to approximate intractable integrals.

Apart from algorithms that resulted from our research conducted at the microscopic level, we also present theoretical contributions that led to these algorithms.

We would like to point out that most prior weighted model integration solvers focus on exact inference, with the notable exception of [Belle et al., 2015b]. In this thesis we additionally start tackling approximate probabilistic inference for weighted model integration – a problem that has so far not received a lot of attention. This is done by means of Monte Carlo estimation of integrals, an approximation technique complementary to the approximations done by Belle et al. [2015b].

Macroscopic: Probabilistic Logic Programming

While weighted SMT formulas present a sound way to express intricate probabilities distributions over discrete and continuous random variables, they are quite low level. As such, modeling and encoding problems might be a cumbersome task from a user's perspective. This leads to the second research question.

RQ2: Can we develop a high-level probabilistic logic programming language for which inference reduces to weighted model integration?

As a **second contribution** we present DC-ProbLog, a probabilistic logic programming language that allows a user to express probability distributions in the discrete-continuous domain while having access to the expressive power of a high-level programming language. A small teaser example program is shown in Example 1.4. Probabilistic inference in DC-ProbLog is reduced to inference over weighted SMT formulas, i.e. to weighted model integration. A similar reduction [Fierens et al., 2015] has already been performed in the purely discrete domain, where the probabilistic programming language ProbLog was reduced to weighted model counting (the discrete equivalent of weighted model integration).

Example 1.4. *Consider a simplified and modified version of the program in Example 1.1. We now model the temperature as continuous random variable that*

is distributed according to normal distributions with mean 20 *and standard deviation* 4. *We would like to know the probability of the machine working.*

```
temperature ~ normal(20,4).

works:- temperature>15.

query(works).
```

In contrast to a probabilistic programming language with discrete/Boolean random variables exclusively, a language with continuous variables as well gives more expressive power to the user of the language.

Cognitive: Probabilistic Perceptual Anchoring

At the cognitive level we study the possibility of applying probabilistic logic programming (macroscopic level) to cognitive robotics. The goal of cognitive robotics is the design of autonomous robotic agents with reasoning, learning and planning capacities. For an autonomous agent to be able to intelligently navigate the surrounding world, it has to construct an internal representation thereof. A notion that tackles this problem in a data-driven fashion is bottom-up perceptual anchoring [Coradeschi; Saffiotti, 2000; Coradeschi; Saffiotti, 2001; Loutfi et al., 2005]. Perceptual anchoring tackles the problem of creating and maintaining, in time and space, the correspondence between symbols and sensor data that refer to the same physical object in the external world. In other words, it targets bridging the symbolic/sub-symbolic gap in cognitive robotics.

RQ3: Can we equip a cognitive robotics system with probabilistic reasoning capacities?

As a **third contribution**, we present a cognitive robotics architecture that couples perceptual anchoring and probabilistic logic programming. The resulting system is capable of reasoning probabilistically about the observed world by constructing a symbolic representation thereof. Furthermore we deploy techniques from the field of statistical relational learning, which endows the developed cognitive robotics system with learning faculties.

Example 1.5. *Consider the scenario in Figure 1.2. The perceptual anchoring system produces symbolic information from raw image data, e.g.* `ball-1`. *However, once the stream of data is interrupted, for example by the hand in the right panel of Figure 1.2, the anchoring system does not produce any symbolic information anymore. At this*

point the probabilistic logic programming component kicks in: through the relation that the ball enters with the hand (the ball is hidden by the hand*), the probabilistic logic programming component knows that the ball is still present. This is indicated by the yellow dots drawn on top of the hand. This information is then fed to the anchoring system. The probabilistic reasoning system (in form of a probabilistic logic programming language), covers situations when there is no direct sensor data arriving but qualities, such as the position of an object, can be inferred (probabilistically). An in detail introduction to and treatise of anchoring, including its historical development, can be found in* Persson*'s PhD thesis [*Persson*, 2019].*

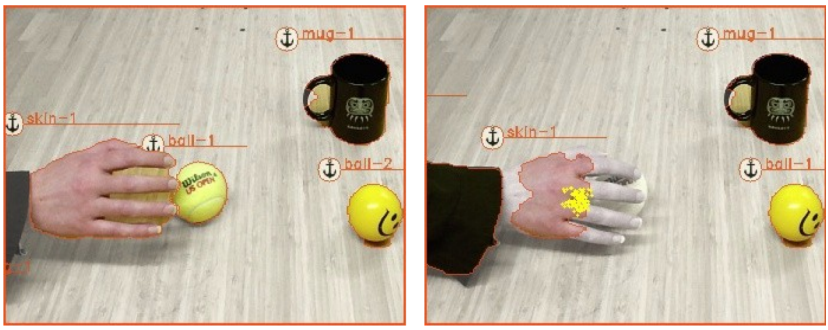


Figure 1.2: Example of the ability of a system that combines perceptual anchoring and probabilistic logic programming to perform reasoning on raw sensor data.

1.4 Structure of the Thesis

The thesis will take you, as a reader, on a journey through the land of probability theory, logic, and programming. First, we will start out at the indivisible atomic building blocks that constitute propositional logic from which we construct our first probabilistic gadgets: Boolean variables that might be true or false. From these atoms we construct then possible (logic) worlds consisting of more than one atom and we will try to figure out how many worlds are true (in expectation). Secondly, we continue by adding logic programming to the mix and develop a probabilistic logic programming language and link it back to the very first atomic building blocks that we started out with. Thirdly, we apply our experience gained so far to cognitive robotics where we introduce a cognitive robotics agent that constructs possible internal representations of the real world in order to reason about the world adequately under uncertainty. We split

up the journey into three legs. Each leg treats one of the three levels of abstractions and can be read rather independently of the other two legs.

Weighted Model Integration

In this first part we study weighted model integration. **Chapter 2** introduces the necessary background, which is then followed by **Chapters 3, 4 and 5** – each introducing novel probabilistic inference algorithms for solving weighted model integration problems. The following papers have been incorporated in this part.

Zuidberg Dos Martires, Pedro; Dries, Anton; De Raedt, Luc [2019b]. Exact and Approximate Weighted Model Integration with Probability Density Functions Using Knowledge Compilation. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Kolb, Samuel; Zuidberg Dos Martires, Pedro; De Raedt, Luc [2019b]. How to Exploit Structure while Solving Weighted Model Integration Problems. In: *Proceedings of the Uncertainty in Artificial Intelligence (UAI) Conference*.

Zuidberg Dos Martires, Pedro; Kolb, Samuel [2020]. Monte Carlo Anti-Differentiation for Approximate Weighted Model Integration. In: *Ninth International Workshop on Statistical Relational AI @ AAAI*.

Probabilistic Logic Programming

In the second part, we continue with the presentation of DC-ProbLog, a probabilistic logic programming language. In **Chapter 6** we introduce the syntax, sketch the semantics, and provide an inference algorithm for DC-ProbLog. Chapter 6 is largely based on a workshop paper and paper currently in preparation.

Zuidberg Dos Martires, Pedro; Dries, Anton; De Raedt, Luc [2018]. Knowledge Compilation with Continuous Random Variables and its Application in Hybrid Probabilistic Logic Programming. In: *Eigth International Workshop on Statistical Relational AI @ IJCAI*.

Zuidberg Dos Martires, Pedro; Kimmig, Angelika; De Raedt, Luc [2020a]. Extending ProbLog with Random Function Symbols. In: *(in preparation)*.

Probabilistic Perceptual Anchoring

The last part is concerned with probabilistic perceptual anchoring where we combine perceptual anchoring and probabilistic programming. **Chapter 7** first introduces the necessary preliminaries on perceptual anchoring and probabilistic programming. This is followed by **Chapters 8 and 9**, where we present and evaluate our architecture of a cognitive system combing probabilistic programming and perceptual anchoring. The presented material is based on two previously published papers.

Persson, Andreas; Zuidberg Dos Martires, Pedro; Loutfi, Amy; De Raedt, Luc [2020b]. Semantic Relational Object Tracking. In: *IEEE Transactions on Cognitive and Developmental Systems* 12.1, pp. 84–97.

Zuidberg Dos Martires, Pedro; Kumar, Nitesh; Persson, Andreas; Loutfi, Amy; De Raedt, Luc [2020b]. Symbolic Learning and Reasoning with Noisy Data for Probabilistic Anchoring. In: *Frontiers in Robotics and AI* 7, p. 100.

Conclusions

Besides providing a concluding chapter at the end of each of the three parts of the thesis, we additionally provide an overarching concluding chapter at the end of the thesis with a brief summary and an outlook on future work.

Weighted Model Integration

Introduction

In the first part of the thesis we discuss, the microscopic level, i.e. the **atomic** building blocks that are needed to perform probabilistic inference in the discrete-continuous domain. These atomic building blocks will serve us as a low level language to express uncertainty about the world. In a sense, this part of the thesis describes an assembly language to express both uncertainty and certainty.

We start with a chapter delivering the necessary background on weighted model counting, weighted model integration, and knowledge compilation. We then proceed with three chapters where we present probabilistic inference algorithms that tackle problems in the discrete continuous domain, while adopting probabilistic inference techniques deployed for to either the purely discrete domain, or the purely continuous domain, and thereby answering our first research question.

RQ1: Can we adopt inference algorithms from the purely discrete domain or the purely continuous domain to develop novel WMI solvers?

In Chapter 3 we study how standard knowledge compilation, an inference technique from the discrete domain, can be extended to the discrete-continuous domain. This contend was previously published as[2]:

Zuidberg Dos Martires, Pedro; Dries, Anton; De Raedt, Luc [2019b]. Exact and Approximate Weighted Model Integration with Probability Density Functions Using Knowledge Compilation. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

[2] I played a central role in developing the research idea, developing the theoretical contributions, implementing the algorithms, and writing the paper.

Chapter 4 further develops these ideas by exploiting additional structure, leading to more efficient algorithms. The content was previously published as[3]:

Kolb, Samuel; Zuidberg Dos Martires, Pedro; De Raedt, Luc [2019b]. How to Exploit Structure while Solving Weighted Model Integration Problems. In: *Proceedings of the Uncertainty in Artificial Intelligence (UAI) Conference*.

While Chapter 3 and Chapter 4 have a strong focus on exact probabilistic inference, Chapter 5 puts approximate probabilistic inference (in form of Monte Carlo approximations) front and center. Specifically, we study the combination of dynamic programming and Monte Carlo integration in high-dimensional linearly constrained spaces.[4]:

Zuidberg Dos Martires, Pedro; Kolb, Samuel [2020]. Monte Carlo Anti-Differentiation for Approximate Weighted Model Integration. In: *Ninth International Workshop on Statistical Relational AI @ AAAI*.

[3]First authorship is shared with Samuel Kolb. Together, we shared a central role in the theoretical and practical contributions of this paper, as well as writing the paper itself.

[4]I played a major role in developing the research idea, implementing the algorithms, and writing the paper (first and last author).

Chapter 2

Background

This chapter describes key concepts necessary to understand our research on weighted model integration. In other words, we explain the underpinnings of the probabilistic inference techniques that we developed and that we present in the first part of the thesis.

2.1 Logic and Satisfiability

2.1.1 Boolean Satisfiability

Logic is one of the pillars of mathematics and computer science alike, and plays a prominent role in artificial intelligence as well. Logic constitutes a formal language to describe the world, which in turn allows us to model or describe the world in a way that machines/computers are able to reason and draw conclusions.

Example 2.1. *We have a factory and would like to model describe under which conditions a machine in this factory breaks down. Consider the following logic formula.*

$$\texttt{broken} \leftrightarrow \texttt{no\_cool} \vee \texttt{high\_temperature} \tag{2.1}$$

In other words, the machine breaks down if and only if the cooling of the machine is not working or the ambient temperature is too high.

Formally we define logic formulas as follows:

Definition 2.1. *Let* **b** *be a set of M Boolean variables. We then define propositional logic formulas as Boolean combinations (by means of the standard Boolean operators* $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$*) of* **Boolean variables** $b_i \in \mathbf{b}$.

A fundamental question is whether there exists an assignment to the Boolean variables present in the logic formula that satisfies the formula. For the formula in Example 2.1 this is indeed the case. Picking, for instance `no_cool` $= \top$ (true) and `high_temperature` $= \bot$ (false) evaluates the formula to true. There is at least one satisfying assignment and the formula is *satisfiable*.

Definition 2.2 (Satisfying interpretation of propositional logic formula)**.** *Let* $\mathbf{j}$ *and* $\mathbf{k}$ *be two disjoint sets of variables and* $\phi(\mathbf{j}, \mathbf{k})$ *be a propositional formula over* $\mathbf{j}$ *and* $\mathbf{k}$*. The set of total interpretations (or total assignments) that satisfy* ϕ *is the set of assignments to the elements in* $\mathbf{j}$ *and* $\mathbf{k}$ *that satisfy* $\exists \mathbf{j},\mathbf{k}$*:* $\phi(\mathbf{j},\mathbf{k})$*. We denote the set of total satisfying interpretations (or models) by* $\mathcal{I}_{\mathbf{j},\mathbf{k}}(\phi)$*. The set of partial interpretations is denoted by* $\mathcal{I}_{\mathbf{j}}(\phi)$*, which is the set of assignments to* $\mathbf{j}$ *that satisfy* $\exists \mathbf{k} : \phi(\mathbf{j}, \mathbf{k})$*. The set of total assignments to a partially interpreted formula is denoted by* $\mathcal{I}_{\mathbf{j}}(\phi^{\mathbf{k}})$*, which denotes the set of assignments to the elements in* $\mathbf{j}$ *that satisfy* $\phi(\mathbf{j}, \mathbf{k}_{\mathcal{I}})$*, with* $\mathbf{k}_{\mathcal{I}} \in \mathcal{I}_{\mathbf{k}}(\phi)$*.*

Determining whether a propositional logic formula is satisfiable is in a computational hard problem and falls in the NP-complete completxity class[1]. Nevertheless, a plethora of practical solvers exists (e.g. MiniSAT [Sorensson; Een, 2005],CryptoSAT [Lafitte, 2018]) that tackle the Boolean satisfiability problem and perform astoundingly well in practice by exploiting structure present in all problems but the most intricate ones.

2.1.2 Satisfiability Modulo Theories

In Example 2.1 we modeled the temperature using a Boolean variable, which seems odd. After all, temperature is inherently continuous. This is where satisfiability modulo theory (SMT) formulas come to the rescue [Barrett; Tinelli, 2018].

Example 2.2. *Consider the SMT theory* `broken`*:*

$$\texttt{broken} \leftrightarrow (\texttt{no\_cool} \wedge (\texttt{t} > 20)) \vee (\texttt{t} > 30) \tag{2.2}$$

`no_cool` *is a Boolean variable and* `t` *a real-valued variable.*

SMT formulas generalize propositional logic formulas to additionally allow the use of expressions formulated in a background theory.

Definition 2.3 (SMT($\mathcal{RA}$) (real arithmetics))**.** *Let* $\mathbf{b}$ *be a set of* M *Boolean and* $\mathbf{x}$ *a set of* N *real variables. An* **atomic formula** *is an expression of the form* $g(X) \bowtie c$*,*

[1] We will repeatedly mention the complexity class of specific computational problems throughout the text. However, we will not provide any extra detail or background on these classes, or how they relate to each other. In the context of probabilistic inference, we invite the interested reader to consult the the following three papers on the topic of complexity theory: [Park; Darwiche, 2004; De Campos; Cozman, 2005; Ceylan et al., 2016].

where $c \in \mathbb{Q}$, $\bowtie \in \{=, \neq, \geq, \leq, >, <\}$, *and* $g : \mathbb{R}^N \to \mathbb{R}$. *The symbols* $\mathbb{Q}$ *and* $\mathbb{R}$ *denote the rational number and real numbers, respectively. We then define SMT($\mathcal{LRA}$) formulas as Boolean combinations (by means of the standard Boolean operators* $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$*) of* **Boolean variables** $b_i \in \mathbf{b}$ *and of* **atomic formulas** *over* $\mathbf{x}$.

We distinguish two special cases:

- *SMT($\mathcal{NRA}$) (non-linear real arithmetics): atomic formulas take the form* $\sum_i c_i \cdot x_i^{p_i} \bowtie c$, *where the* $x_i \in \mathbf{x}$ *and* $c_i, c, p_i \in \mathbb{Q}$.
- *SMT($\mathcal{LRA}$) (linear real arithmetics): atomic formulas take the form* $\sum_i c_i \cdot x_i \bowtie c$, *where the* $x_i \in \mathbf{x}$ *and* $c_i, c \in \mathbb{Q}$.

Note that we introduced three different quantifier free fragments of SMT formulas that all use a variation of the reals as a background theory. These will be the only types of SMT formulas used in the remainder of the text. Other common background theories used are *linear integer arithmetics* or *bit-vectors* [Barrett; Tinelli, 2018].

Definition 2.4 (Satisfying interpretation of SMT formula)**.** *Let* $\mathbf{j}$ *and* $\mathbf{k}$ *be two disjoint sets of variables and* $\phi(\mathbf{j}, \mathbf{k})$ *be an SMT formula over* $\mathbf{j}$ *and* $\mathbf{k}$. *The set of total interpretations (or total assignments) that satisfy* ϕ *is the set of assignments to the elements in* $\mathbf{j}$ *and* $\mathbf{k}$ *that satisfy* $\exists \mathbf{j},\mathbf{k}$: $\phi(\mathbf{j},\mathbf{k})$. *We denote the set of total satisfying interpretations (or models) by* $\mathcal{I}_{\mathbf{j},\mathbf{k}}(\phi)$. *The set of partial interpretations is denoted by* $\mathcal{I}_{\mathbf{j}}(\phi)$, *which is the set of assignments to* $\mathbf{j}$ *that satisfy* $\exists \mathbf{k} : \phi(\mathbf{j}, \mathbf{k})$. *The set of total assignments to a partially interpreted formula is denoted by* $\mathcal{I}_{\mathbf{j}}(\phi^{\mathbf{k}})$, *which denotes the set of assignments to the elements in* $\mathbf{j}$ *that satisfy* $\phi(\mathbf{j}, \mathbf{k}_{\mathcal{I}})$, *with* $\mathbf{k}_{\mathcal{I}} \in \mathcal{I}_{\mathbf{k}}(\phi)$.

Analogous to the SAT problem, the satisfiability problem can also be asked for SMT formulas, i.e. is there an assignment to Boolean and real variables present in an SMT formula that satisfy the formula. For our Example 2.2 SMT answers the question whether or not there is an assignment to the variables `no_cool` and `t` such that the formula is satisfied.

Similarly, to the SAT problem the SMT problem is computationally hard as well (NP-complete for SMT($\mathcal{LRA}$)), yet tools, such as Z3 [De Moura; Bjørner, 2008] and MathSAT [Cimatti et al., 2013], that solve the SMT problem reliably in practice have been developed over the last couple of years.

We also introduce the notion of formula abstraction.

Definition 2.5. *(Atomic formula abstraction) Let* $c(\mathbf{x})$ *be an atomic formula (cf. Definition 2.3),* $abs_{c(\mathbf{x})}$ *is then called the atomic formula abstraction of c, given that* $(abs_{c(\mathbf{x})} \leftrightarrow \exists \mathbf{x}.c(\mathbf{x}))$ *holds.*

Example 2.3. *Consider the following SMT formula from Example 2.2:*

$$\texttt{broken} \leftrightarrow (\texttt{no\_cool} \wedge (\texttt{t} > 20)) \vee (\texttt{t} > 30) \tag{2.3}$$

Abstracting all SMT($\mathcal{LRA}$) atomic formulas gives use the abstracted formula:

$$\texttt{broken}_{abs} \leftrightarrow (\texttt{no\_cool} \wedge b_{\texttt{t}>20}) \vee b_{\texttt{t}>30} \tag{2.4}$$

$b_{\texttt{t}>20}$ and $b_{\texttt{t}>30}$ are fresh Boolean variables equivalent to the respective SMT($\mathcal{LRA}$) atomic formulas.

Iverson Brackets A handy notational apparatus, of which we will make ample use, are *Iverson brackets* [Iverson, 1962; Knuth, 1992]. Iverson brackets map propositional logic formulas to 1 if their argument evaluates to true (is satisfied) and to 0 otherwise. We will denote Iverson brackets by double square brackets:

$$[\![p]\!] = \begin{cases} 1, \text{ if p is true} \\ 0, \text{ otherwise} \end{cases} \tag{2.5}$$

Iverson brackets allow us to seamlessly mix real numbers and SMT formulas:

$$f(x) = 0.2 \times [\![x > 4]\!] + 0.8 \times [\![x < 3]\!] \tag{2.6}$$

2.2 The Weight of a Model

2.2.1 The Count of a Model

In the previous section we were interested in the question whether logical formulas (or their SMT extensions) are satisfiable. We can also ask an other questions: how many distinct assignments satisfy the a formula. For propositional logic formulas this simply amounts to counting the number of satisfying assignment to the Boolean variables, a #P-complete problem [Valiant, 1979]. The number of distinct satisfying assignments is dubbed the *model count* and the task of computing the model count is also referred to as $\#SAT$.

Example 2.4. *Consider the Boolean formula for M*

$$M \leftrightarrow (x \vee y) \wedge (y \vee \neg z)$$

A naive and straightforward way to compute the model count is through a truth table, cf. Table 2.1.

Table 2.1: The model count of M is 5

x	y	z	M
0	0	0	
0	0	1	
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	
1	1	0	1
1	1	1	1
			5

While the #SAT problem has received quite some attention from the research community, its sister problem #SMT, where the satisfying models to an SMT formula are counted, has only been of limited interest to researchers so far, with some exceptions [Ma et al., 2009; Chistikov et al., 2015; Zhou et al., 2015; Phan; Malacaria, 2014]. We speculate that this is due to the complicating factor of including a (decidable) first-order logic fragment to propositional logic.

2.2.2 Weighted and Algebraic Model Counting

Weighted model counting generalizes #SAT tasks. Instead of simply counting the number of satisfying assignments, one performs a weighted sum over models.

Definition 2.6 (Weighted mode counting (WMC))**.** *Given a set* $\mathbf{b}$ *of* M *Boolean variables, a weight function* $w : \mathbb{B}^M \to \mathbb{R}_{\geq 0}$*, and a propositional formula* ϕ *(called support) over* $\mathbf{b}$*, the* **weighted model count** *is*

$$WMC(\phi, w|\mathbf{b}) = \sum_{\mathbf{b}_\mathcal{I} \in \mathcal{I}_\mathbf{b}(\phi)} w(\mathbf{b}_\mathcal{I}) \tag{2.7}$$

$\mathcal{I}_\mathbf{b}(\phi)$ *is the set of interpretations that satisfy* ϕ *(cf. Definition 2.4).*

Traditionally, WMC is used when the weight function w factorizes as product of weights of literals:

$$WMC(\phi, w|\mathbf{b}) = \sum_{\mathbf{b}_\mathcal{I} \in \mathcal{I}_\mathbf{b}(\phi)} \prod_{b_i \in \mathbf{b}_\mathcal{I}} w(b_i) \tag{2.8}$$

When performing probabilistic inference, we take

$$0 \leq w(b_i) \leq 1 \quad \text{and} \quad w(b_i) + w(\neg b_i) = 1 \tag{2.9}$$

The resulting sum over products is then a computation in the probability semiring [Kimmig et al., 2017]. Effectively, this computes the probability that propositional logic formula is satisfied.

In [Kimmig et al., 2017] probabilistic inference over propositional formulas, i.e. over satisfying models, has been extended to arbitrary commutative semirings. The general setting of model counting over commutative semirings is called algebraic model counting. More formally,

Definition 2.7. *A* **commutative semiring** *is an algebraic structure* $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ *equipping a set of elements* $\mathcal{A}$ *with addition and multiplication such that*

1. *addition* $\oplus$ *and multiplication* $\otimes$ *are binary operations* $\mathcal{A} \times \mathcal{A} \to \mathcal{A}$
2. *addition* $\oplus$ *and multiplication* $\otimes$ *are associative and commutative binary operations over the set* $\mathcal{A}$
3. $\otimes$ *distributes over* $\oplus$
4. $e^{\oplus} \in \mathcal{A}$ *is the neutral element of* $\oplus$
5. $e^{\otimes} \in \mathcal{A}$ *is the neutral element of* $\otimes$
6. $e^{\oplus}$ *is an annihilator for* $\otimes$

Definition 2.8. *(Algebraic model counting (AMC)) [Kimmig et al., 2017] Given:*

- *a propositional logic theory* ϕ *over a set of variables* $\mathbf{b}$
- *a commutative semiring* $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$
- *a labeling function* $\alpha : \mathcal{L} \to \mathcal{A}$*, mapping literals* $\mathcal{L}$ *from the variables in* B *to values from the semiring set* $\mathcal{A}$

The algebraic model count of a theory ϕ *is then defined as:*

$$AMC(\phi, \alpha|\mathbf{b}) = \bigoplus_{b \in \mathcal{I}_{\mathbf{b}}(\phi)} \bigotimes_{b_i \in b} \alpha(b_i)$$

We use α instead of w and the term label rather than weight to reflect that the elements of the semiring cannot always be interpreted as weights.

Algebraic model counting constitutes a general framework for many common inference tasks in artificial intelligence. Defining an appropriate semiring and labeling functions allows one, for instance, to perform sensitivity analysis, compute gradients or determine the provenance of queries in databases [Kimmig et al., 2017].

2.2.3 Weighted Model Integration

Standard weighted model counting only supports discrete probability distributions. To repair this omission, WMC has recently been extended towards weighted model integration (WMI) [Belle et al., 2015a], supporting additionally continuous variables.

Example 2.5. *Consider again the theory* `broken` *(cf. Equation 2.2). Assume that* `t` *is distributed according to:* $\texttt{t} \sim \mathcal{N}_{\texttt{t}}(20, 5)$ *and that the probability for* `no_cool` *being true is* 0.01. *Determining the probability of the formula being true extends the SMT problem to weighted model integration.*

Following the formulation of [Morettin et al., 2017] we give the defintion of WMI definition:

Definition 2.9 (Weighted model integration (WMI))**.** *Given a set* $\mathbf{b}$ *of* M *Boolean variables,* $\mathbf{x}$ *of* N *real variables, a weight function* $w : \mathbb{B}^M \times \mathbb{R}^N \to \mathbb{R}_{\geq 0}$*, and a support* ϕ*, in the form of an SMT formula, over* $\mathbf{b} \cup \mathbf{x}$*, the* **weighted model integral** *is*

$$WMI(\phi, w|\mathbf{x}, \mathbf{b}) = \sum_{\mathbf{b}_I \in \mathcal{I}_{\mathbf{b}}(\phi)} \int_{\mathcal{I}_{\mathbf{x}}(\phi^{\mathbf{b}_I})} w(\mathbf{x}, \mathbf{b}_I) d\mathbf{x} \tag{2.10}$$

2.3 Knowledge Compilation and Counting

As mentioned earlier, model counting is a computationally hard problem, #P-complete to be precise. Nevertheless practically useful model counters exist. State-of-the-art techniques for solving model counting problems, are based on exhaustive DPLL algorithms [Birnbaum; Lozinskii, 1999], which count the number of satisfying assignments to a formula. These solvers can be divided into two classes: the ones that build up a trace of the DPLL search, and the ones that do not. The latter return immediately the model count. The former builds up a diagrammatic representation of the propositional formula over which the model count can be obtained efficiently. By keeping a trace, such #SAT solvers [Huang; Darwiche, 2005; Oztok; Darwiche, 2018] constitute, in fact, top-down *knowledge compilation* schemes.

Knowledge compilation [Darwiche; Marquis, 2002] has emerged as the go-to technique for dealing with the computational intractability of propositional reasoning (#P-hard [Valiant, 1979]). The key idea is to split up inference on logical formulas into an *off-line* and an *on-line* step. In the off-line step, a propositional formula is compiled from its source representation into a target representation, in which repeated on-line poly-time inference is available.

#SAT can also be performed by compiling formulas bottom-up [Choi; Darwiche, 2013]. However, it has been shown [Huang; Darwiche, 2005; Oztok; Darwiche, 2018] that

top-down compilation, i.e. knowledge compilation through exhaustive DPLL search, outperforms bottom-up compilers.

Similar to model counting, weighted model counting has as well been performed via knowledge compilation, for instance for probabilistic inference in Bayesian networks [Chavira; Darwiche, 2008] probabilistic programming [Fierens et al., 2015].

A popular language to compile propositional formulas into are *Sentential Decisions Diagrams* (SDDs) [Choi et al., 2013], which we will use throughout this part of the thesis. SDDs are a subset of *deterministic decomposable negation normal form* (d-DNNF) formulas [Darwiche, 2001]. SDDs and d-DNNFs are well-known target langauges for knowledge compilation.

Boolean formulas in *negation normal form* [Darwiche, 1999] are nested conjunctions and disjunctions that allow for negation only applied directly to the atoms in the formula. In the next chapter we will need three more restrictions on the logic formulas, which are satisfied by definition by d-NNFs and SDDs[2].

Definition 2.10 (Determinism)**.** *An NNF formula is deterministic if and only if for every disjuntion the disjuncts are pairwise logical inconsistent.*

Definition 2.11 (Decomposability)**.** *An NNF formula is decomposable if and only if for every conjunction the conjuncts do not share any variables.*

Definition 2.12 (Smoothness)**.** *An NNF formula is smooth if and only if for every disjuncion all the disjuncts are Boolean functions over the same variables.*

Smooth d-DNNFs are denoted by sd-DNNFs.

Example 2.6. *Consider again the formula from Example 2.1:*

$$\texttt{broken} \leftrightarrow \texttt{no\_cool} \vee \texttt{high\_temperature}$$

A diagrammatic representation of the formula is depicted in Figure 2.1. Note the extra negation present in the diagram. This ensures that the or node is deterministic, i.e. the disjuncts are mutually inconsistent, which avoid double counting when computing the (weighted) model count.

Let us assume that the cooling of the machine fails with probability

$$p(\texttt{no\_cool})=0.01$$

and that the temperature is too high with probability

$$p(\texttt{high\_temparature})=0.02$$

[2] Further restrictions/properties exist but will not be used in the thesis.

The weighted model count is easily computed by turning the decision diagram (representing the Boolean formula) into a so-called arithmetic circuit *(AC) [*Darwiche, *2003]. An AC is obtained from a DD by replacing the conjunction nodes by multiplication nodes and the disjunction nodes by addition nodes, as well as replacing the Boolean atoms in the leaves with their corresponding weight (in our case these are the probabilities of the atoms being true). The probability for the formula is obtained by evaluated the AC:*

$$p(\texttt{broken}) = 0.98 \times 0.01 + 0.02 = 0.0298 \tag{2.11}$$

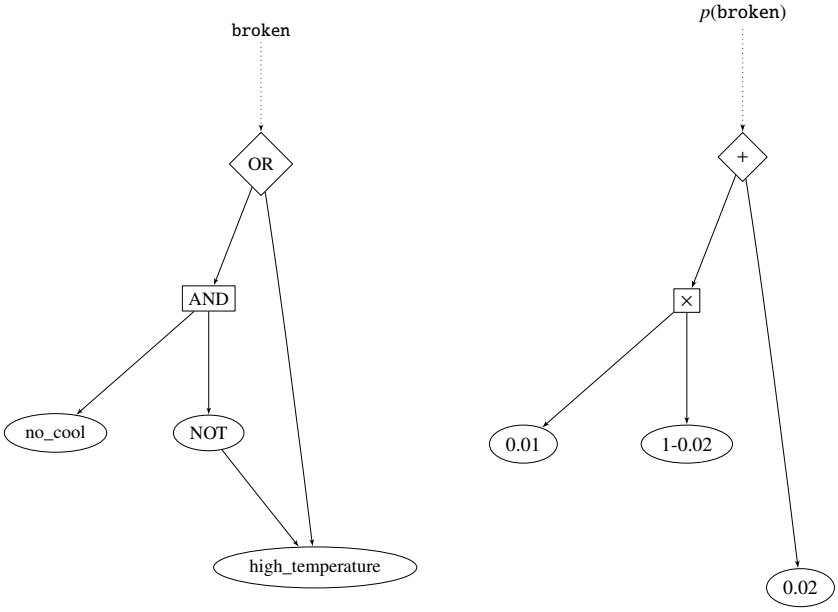


Figure 2.1: On the right we show a diagrammatic d-DNNF representation of the Boolean formula in Equation 2.1 and on the right the corresponding arithmetic circuit.

To compute the algebraic model count on an NNF, Kimmig et al. [2017] give algorithm 2.1.

In order to compute algebraic/weighted model counts on a d-DNNF, we need the neutral-sum property.

Definition 2.13 (Neutral-sum property [Kimmig et al., 2017])**.** *A semiring addition and labeling function pair* $(\oplus, \alpha)$ *is neutral* if and only if $\forall b \in \mathbf{b} : \alpha(b) \oplus \alpha(\neg b) = e^{\otimes}$.

Theorem 2.1 (AMC on d-DNNF [Kimmig et al., 2017])**.** *Evaluating a d-DNNF representation of the propositional theory* ϕ*, using Algorithm 2.1, for a semiring*

Algorithm 2.1 Evaluating an NNF circuit N for a commutative semiring $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ and labeling function α[Kimmig et al., 2017].

1: **function** Eval($(N, \oplus, \otimes, e^{\oplus}, e^{\otimes}, \alpha)$)
2: **if** N is a true node $\top$ **then return** $e^{\otimes}$
3: **if** N is a false node $\bot$ **then return** $e^{\oplus}$
4: **if** N is a literal node l **then return** $\alpha(l)$
5: **if** N is a disjunction $\bigvee_{i=1}^{m} N_i$ **then**
6: **return** $\bigoplus_{i=1}^{m}$ Eval$(N_i, \oplus, \otimes, e^{\oplus}, e^{\otimes}, \alpha)$
7: **if** N is a conjunction $\bigwedge_{i=1}^{m} N_i$ **then**
8: **return** $\bigotimes_{i=1}^{m}$ Eval$(N_i, \oplus, \otimes, e^{\oplus}, e^{\otimes}, \alpha)$

and labeling function with neutral tuple $(\oplus, \alpha)$ is a correct computation (cf. [Kimmig et al., 2017, Definition 10]) of the algebraic model count.

Decision diagrams (DD), such as SDDs, are data structures that compactly represent Boolean formulas. DDs are obtained by compiling Boolean formulas into directed acyclic graphs. Although traditionally developed for propositional logical formulas, the idea of compiling logical formulas into a more succinct representation has also found its way into the hybrid domain, where a prominent example are XADDs [Sanner et al., 2011; Kolb et al., 2018], a language based on binary decision diagrams [Bryant, 1986]. In the next three chapter we will further investigate this line of work: using diagrammatic representations of SMT formulas to perform probabilistic inference.

Chapter 3

WMI Using KC*

Weighted model counting has recently been extended to weighted model integration, which can be used to solve hybrid probabilistic reasoning problems. Such problems involve both discrete and continuous probability distributions. We show how standard knowledge compilation techniques (to SDDs and d-DNNFs) apply to weighted model integration, and use it in two novel solvers, one exact and one approximate solver.

3.1 Introduction

As portrayed in Chapter 2, the state-of-the-art method for probabilistic inference with discrete random variables reduces inference to weighted model counting [Chavira; Darwiche, 2008], while utilizing knowledge compilation [Darwiche; Marquis, 2002].

Standard weighted model counting only supports discrete probability distributions. To repair this omission, WMC has recently been extended towards weighted model integration (WMI) [Belle et al., 2015a], supporting additionally continuous variables. However, the weight functions supported within most formulations of WMI [Belle et al., 2015a; Belle et al., 2016; Morettin et al., 2017; Kolb et al., 2018] allow only for piecewise polynomial functions. Moreover, none of these works has studied the applicability of standard knowledge compilation techniques to WMI.

The key contribution displayed in this chapter is that we show how to handle actual probability density functions instead of piecewise polynomials in the context of WMI by applying standard knowledge compilation techniques. To this end, we cast weighted

*This chapter has previously been published as [Zuidberg Dos Martires et al., 2019b].

model integration within the framework of algebraic model counting [Kimmig et al., 2017].

More specifically, we elucidate the following contributions:

1. We introduce the probability density semiring.
2. We show how this allows us to cast WMI within AMC and thereby to use the general body of literature on knowledge compilation.
3. We introduce *Symbo*, a solver for WMI that realizes knowledge compilation and **exact symbolic inference**.
4. We introduce *Sampo*, a solver for WMI that realizes knowledge compilation and **approximate inference** via sampling.

Symbo exploits the PSI-Solver by [Gehr et al., 2016] to simplify algebraic expressions, while Sampo is based on mapping the arithmetic circuit that results from the KC step onto Edward [Tran et al., 2016], a probabilistic programming language wrapped around TensorFlow [Abadi et al., 2016]. The latter transforms approximate inference into an embarrassingly parallelizable task.

For the remainder of the chapter, we will assume that the weight function factorizes as:

$$w(\mathbf{x}, \mathbf{b}) = w_x(\mathbf{x}) w_b(\mathbf{b}) = w_x(\mathbf{x}) \prod_{b \in \mathbf{b}} w_b(b) \tag{3.1}$$

We can assume this without loss of generality as any weight function that does not follow this factorization can be rewritten as a sum of weight functions over mutually exclusive partial assignments to the Boolean variables, where each individual term of the sum factorizes according to Equation 3.1. The weighted model integral is then expressed as a sum over weighted model integrals. This point is exposed in more detail in Section 4.2.

3.2 The Probability Density Semiring

We are now going to define the probability density semiring and the labeling function, cf. Definition 2.8. This will allow us to cast WMI as AMC.

Definition 3.1 (Labeling function α)**.** *Let l be a literal. Then the label of the literal l is given by:*

$$\alpha(l) := \begin{cases} p(l) & \text{if } l \text{ is a Boolean variable} \\ [\![c(\mathbf{x})]\!] & \text{if } l \text{ is an atomic formula abstraction} \end{cases}$$

In the former case, $p(l)$ denotes the probability for l being true and in the latter case, $c(\mathbf{x})$ denotes the condition of which l is the abstraction.

The label of a negated literal $\neg l$ is given by:

$$\alpha(\neg l) := \begin{cases} 1 - p(l) & \text{if } l \text{ is a Boolean variable} \\ [\![\neg c(\mathbf{x})]\!] & \text{if } l \text{ is an atomic formula abstraction} \end{cases}$$

Example 3.1. *Applying the labeling function α to the literals in our introductory Example 2.5 yields, for instance: $\alpha(\texttt{no\_cool})=0.01$ and $\alpha(abs_{\texttt{t>20}})=[\![\texttt{t}>20]\!]$.*

Definition 3.2 (Probability density semiring $\mathcal{S}$)**.** *The elements of the semiring $\mathcal{S}$ are given by the set*

$$\mathcal{A} := \{a\} \tag{3.2}$$

where a denotes any algebraic expression over $\mathcal{RA}$. The neutral elements $e^{\oplus}$ and $e^{\otimes}$ are defined as:

$$e^{\oplus} := 0 \qquad\qquad e^{\otimes} := 1 \tag{3.3}$$

For the addition and multiplication we define:

$$a_1 \oplus a_2 := a_1 + a_2 \tag{3.4}$$

$$a_1 \otimes a_2 := a_1 \times a_2 \tag{3.5}$$

Example 3.2. *An example of an algebraic expression over $\mathcal{RA}$ would be $0.01\times[\![s+20 < t]\!] \times [\![t \leq 30]\!] + [\![t > 30]\!]$, $s \in \mathbb{R}$, $t \in \mathbb{R}$.*

Lemma 3.1. *The structure $\mathcal{S} = (\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ is a commutative semiring.*

Proof (Sketch). We need to show that the properties in Definition 2.7 hold. The proof relies on the commutativity and associativity of the Iverson brackets under standard addition and multiplication. Similarly for the distributivity of the multiplication over the addition (cf. Property 3). Lastly, properties 4 to 6 are trivially satisfied. We conclude that the structure $\mathcal{S}$ is indeed a commutative semiring. □

Lemma 3.2. *The pair $(\oplus, \alpha)$ is neutral, i.e. $\alpha(l) \oplus \alpha(\neg l) = e^{\otimes}$, where l is a literal.*

Proof. We have two cases:

1. l is a Boolean variable, $\alpha(l) \oplus \alpha(\neg l) = P(l) \oplus 1 - P(l) = 1$.
2. l is an atomic formula: $\alpha(l) \oplus \alpha(\neg l) = [\![l]\!] \oplus [\![\neg l]\!] = [\![l]\!] + [\![\neg l]\!] = [\![\top]\!] = 1$ □

Lemma 3.3 (AMC on d-DNNF with $\mathcal{S}$)**.** *The algebraic model count is a correct calculation on a d-DNNF representation of a logic formula given the density semiring $\mathcal{S}$.*

Proof. This follows immediately from Lemma 3.1 and 3.2, together with Theorem 2.1. □

3.3 WMI via AMC

A key difference between WMI and AMC is that in an AMC task there is no integral. This intuitively implies that we need to perform an integration on the algebraic model count if we want to cast WMI using AMC: "WMI = $\int$ AMC". Additionally, WMI is defined on SMT formulas and AMC on propositional logic formulas. We address these differences in Theorem 3.1, which also allows us to show that WMI can be cast as AMC.

Theorem 3.1. *Let ϕ be an SMT($\mathcal{RA}$) formula over the Boolean variables in the set $\mathbf{b}$ and continuous variables in the set $\mathbf{x}$. Let ϕ_a be the propositional logic formula over the set of Boolean variables $\mathbf{b}$ and $\mathbf{b_x}$, where $\mathbf{b_x}$ is the set of abstractions of atomic formulas (cf. Definition 2.5) in ϕ. Let w be a weight function over the Boolean variables in $\mathbf{b}$ and the continuous variables in $\mathbf{x}$. Furthermore, let $AMC(\phi_a, \alpha|\mathbf{b_x} \cup \mathbf{b})$ evaluate to Ψ in the semiring $\mathcal{S}$, with $\Psi = \sum_{v \in \mathcal{I}_{\mathbf{b},\mathbf{b_x}}(\phi_a)} \prod_{v_i \in v} a_{v_i}$. Then*

$$WMI(\phi, w|\mathbf{x}, \mathbf{b}) = \int \Psi w_x(\mathbf{x}) d\mathbf{x} \tag{3.6}$$

Proof. In the first step we rewrite Ψ (an example is given in Example 3.2) as the sum-product over the algebraic expressions a_v. The a_v are the probability or Iverson labels of the literals from Definition 3.1. In the second step (Equation 3.8 to 3.9) we split up the sum and the product over the variables b into sums over the abstractions of atomic formulas a_{x_i} and atomic propositions a_{b_i} - likewise for the product. b_i and x_j denote the assignment to a specific variable for a variables in the set of variables $\mathbf{b}$ and $\mathbf{x_a}$, respectively. The superscript $\mathbf{b}$ in $\phi_a^{\mathbf{b}}$ indicates a specific assignment to the Boolean variables corresponding to the atomic propositions. Next (Equation 3.9 to 3.10), we push the product over the atomic propositions through and note that this product corresponds to the weight of the Boolean variables $w_b(\mathbf{b})$.

$$\int \Psi w_x(\mathbf{x}) d\mathbf{x} \tag{3.7}$$

$$= \int \left(\sum_{\mathbf{v} \in \mathcal{I}_{B,B_X}(\phi_a)} \prod_{v_i \in \mathbf{v}} a_{v_i} \right) w_x(\mathbf{x}) d\mathbf{x} \tag{3.8}$$

$$= \int \sum_{\mathbf{b} \in \mathcal{I}_B(\phi_a)} \sum_{\mathbf{x_a} \in \mathcal{I}_{B_X}(\phi_a^{\mathbf{b}})} \left(\prod_{b_i \in \mathbf{b}} a_{b_i} \right) \left(\prod_{x_j \in \mathbf{x_a}} a_{x_j} \right) w_x(\mathbf{x}) d\mathbf{x} \tag{3.9}$$

$$= \int \sum_{\mathbf{b} \in \mathcal{I}_B(\phi)} \sum_{\mathbf{x_a} \in \mathcal{I}_{B_X}(\phi_a^{\mathbf{b}})} \prod_{x_j \in \mathbf{x_a}} a_{x_j} w_b(\mathbf{b}) w_x(\mathbf{x}) d\mathbf{x} \tag{3.10}$$

$$= \sum_{\mathbf{b} \in \mathcal{I}_B(\phi)} \int \sum_{\mathbf{x_a} \in \mathcal{I}_{B_X}(\phi_a^{\mathbf{b}})} \prod_{x_j \in \mathbf{x_a}} a_{x_j} w(\mathbf{x}, \mathbf{b}) d\mathbf{x} \tag{3.11}$$

$$= \sum_{\mathbf{b} \in \mathcal{I}_B(\phi)} \int_{\mathbf{x} \in \mathcal{I}_X(\phi^{\mathbf{b}})} w(\mathbf{x}, \mathbf{b}) d\mathbf{x} \tag{3.12}$$

In Equation 3.11 we exchanged the summation and the integration (assuming that Fubini's theorem [Fubini, 1907] holds). We also rewrote the product of the weight functions for the Booleans and for the continuous variables as a single weight function, assuming that the weight function factorizes accordingly. The integral over the so-obtained sum-product is the integral over Iverson brackets. In Equation 3.12 we rewrite the indefinite integral over the Iverson brackets as the definite integral with boundary conditions corresponding to the conditions present in the Iverson brackets. This corresponds to the definition of the weighted model integral. □

We have shown that we can solve a WMI problem by formulating it as an AMC problem, given that the weight function is factorizable. The weighted model integral for a non-factorizable weight function is then obtained by adding up the weighted model integrals for the factorizeable weight functions into which the problem decomposes.

3.4 Computing the Probability of SMT Formulas

We describe now **Symbo** and **Sampo**, algorithms that, respectively, produce the exact and the approximate weighted model integral of an SMT formula ϕ and a factorizable

weight function w utilizing knowledge compilation.[1]

3.4.1 Symbo

In Lemma 3.3 we saw that the probability semiring S can be used to calculate the algebraic model count on a d-DNNF representation of a logical formula. Recalling Theorem 3.1, we are hence also capable of obtaining the weighted model integral for an SMT formula, given the probability distributions of the random variables.

Algorithm 3.1 (Symbo)**.** *Symbo computes the weighted model integral of an SMT($\mathcal{NRA}$) formula ϕ for a factorizable weight function w by executing the following steps:*

1. *Abstract all atomic formulas in ϕ according to Definition 2.5 and obtain ϕ_a.*
2. *Compile ϕ_a into a d-DNNF representation $\phi_{compiled}$.*
3. *Transform $\phi_{compiled}$ into an arithmetic circuit AC_ϕ by replacing logical and/or operations with symbolic multiplications/additions.*
4. *Label literals in AC_ϕ according to the labeling function given in Definition 3.1 with corresponding symbolic values.*
5. *Symbolically evaluate AC_ϕ and obtain Ψ.*
6. *Multiply Ψ by the weight of the continuous variables on which Ψ depends.*
7. *Symbolically integrate over the continuous variables by calling a symbolic inference engine.*

We implemented Algorithm 3.1 using the SDD package[2] for the KC step and the inference engine of the PSI-Solver for symbolic manipulations[3]. Note that in our initial work [Zuidberg Dos Martires et al., 2019b] we did not explicitly give a name to the class of compiled SMT formulas. Only in [Kolb et al., 2019b], the paper on which Chapter 4 is based, did we introduce the name XSDD: the abstracted SMT formula is represented as an SDD.

Example 3.3. *Consider our introductory Example 2.5. Executing the first two steps of Symbo yield the compiled logic formula that is shown on the left in Figure 3.1. Steps*

[1] Implementations of both algorithms are available under `https://bitbucket.org/pedrozudo/hal_problog`.

[2] `http://reasoning.cs.ucla.edu/sdd/`

[3] For a detailed discussion of allowed symbolic manipulations see [Gehr et al., 2016]

number three and four of Symbo produce the arithmetic circuit on the right in Figure 3.1.

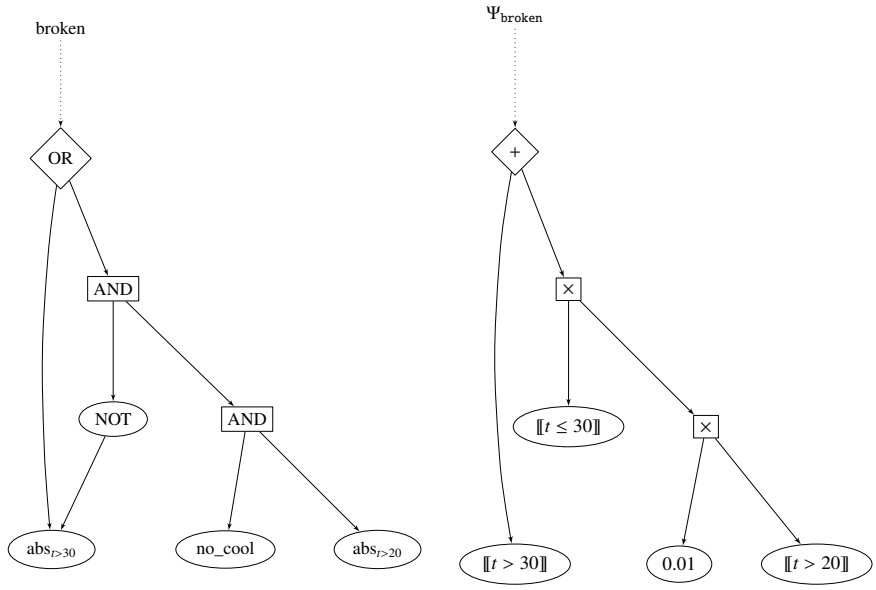


Figure 3.1: Shown on the left is a graphical representation of the compiled logic formula given in Equation 2.2 (in the SDD target language), where the atomic formulas have been abstracted away. On the right we see the corresponding arithmetic circuit where the literals have been replaced by corresponding labels according to the labeling function in 3.1 and where the logic and/or operation have been replaced by ×/+ respectively.

The probability for the theory `broken`*, which coincides with the weighted model integral, is obtained by evaluating the arithmetic circuit (step five), multiplying this expression by the probability density function for* `t` *(step six) and carrying out the integral (step 7).*

$$
\begin{aligned}
& p(\texttt{broken}) \\
&= \int (0.01[\texttt{t}{>}20][\texttt{t}{\leq}30] + [\texttt{t}{>}30])\, \mathcal{N}_{\texttt{t}}(20,5) d\texttt{t} \\
&= 0.01 \int_{20<\texttt{t}\leq 30} \mathcal{N}_{\texttt{t}}(20,5) d\texttt{t} + \int_{\texttt{t}>30} \mathcal{N}_{\texttt{t}}(20,5) d\texttt{t} \\
&= 1 - 0.01 \int_{-\infty}^{-\frac{5\sqrt{8}}{2}+\frac{20}{\sqrt{8}}} e^{-x^2} dx - 0.99 \int_{-\infty}^{-\frac{5\sqrt{8}}{2}+\frac{30}{\sqrt{8}}} e^{-x^2} dx
\end{aligned}
$$

In Example 3.3, the weight function on the continuous variables depended only on a single variable. It is, however, easy to see that our formalism does also allow for multivariate distributions that are then used with more intricated integration bounds, such as in Example 3.2.

3.4.2 Sampo

In the general case, symbolic inference methods are not able to produce numerical results to a given problem. This is because the resulting integrals are not tractable utilizing symbolic integration. For such cases Monte Carlo (MC) methods are used to compute intractable integrals by approximating the integration by a summation.

Theorem 3.2 (MC approximation of WMI)**.** *Let ϕ be an SMT($\mathcal{RA}$) theory, w a factorizable weight function over the Boolean variables $\mathbf{b}$ and continuous variables $\mathbf{x}$. Furthermore, let $AMC(\phi, w|\mathbf{x} \cup \mathbf{b})$ evaluate to Ψ. Then the Monte Carlo approximation of $WMI(\phi, w|\mathbf{x}, \mathbf{b})$ is given by:*

$$WMI_{MC}(\phi, w|\mathbf{x}, \mathbf{b}) \coloneqq \frac{1}{N} \sum_{i=1}^{N} \Psi(\mathbf{x}_i) \tag{3.13}$$

where the $\mathbf{x}_i$'s are N independent and identically distributed random variables drawn from the density w.

Proof.

$$WMI(\phi, w|\mathbf{x}, \mathbf{b}) = \int \Psi(\mathbf{x}) w_x(\mathbf{x}) d\mathbf{x} \tag{3.14}$$

$$= E_{w_x(\mathbf{x})}[\Psi(\mathbf{x})] \tag{3.15}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \Psi(\mathbf{x}_i) \tag{3.16}$$

The expression in Eqaution 3.15 denotes the expectation of $\Psi(\mathbf{x})$ with respect to $w(\mathbf{x})$. The approximation in 3.16 is the mean value of Ψ obtained through MC assignments to the continuous random variables present in Ψ. □

The MC approximation of the weighted model integral of an SMT formula necessitates that we evaluate a compiled SMT problem at N different points, i.e. we need to evaluate a compiled theory N times with different weights. This is exactly where the strength of knowledge compilation lies: expensively compile once and cheaply evaluate often.

Numerical computation libraries such as TensorFlow rely heavily on the concept of computation graphs. Realizing that we can translate a d-DNNF formula to a computational graph and express the labels of literals in an SMT formula as tensors allows us to compute the N evaluations necessary for the MC approximation of the weighted model integral not only cheaply but also in parallel.

Algorithm 3.2 (Sampo)**.** *Sampo computes the weighted model integral of an SMT($\mathcal{RA}$) formula ϕ for a factorizable weight function w by executing the following steps:*

1. *Abstract all atomic formulas in ϕ according to Definition 2.5 and obtain ϕ_a.*
2. *Compile ϕ_a into a d-DNNF representation $\phi_{compiled}$.*
3. *Transform $\phi_{compiled}$ into an arithmetic circuit AC_ϕ, i.e. replacing logical and/or operations with elementwise tensor multiplications/additions.*
4. *Label the literals in AC_ϕ according to the labeling function given in Definition 3.1 with corresponding tensors.*
5. *Symbolically evaluate AC_ϕ and obtain Ψ represented by a computation graph CG.*

6. *Run the CG representing Ψ N times, where N is the number of samples approximating the probability densities.*

7. *Take the mean of the values of the N runs of the CG.*

We implemented Algorithm 3.2 using again the SDD package for the KC step and using TensorFlow as the underlying numerical computation library. Random variables are sampled using the Edward library [Tran et al., 2016].

Example 3.4. *Let us illustrate Sampo on our running example in Equation 2.2. Assume therefore that we already have at hand $AC_{\phi}^{Evaluated}$. We then need to sample N values for the random variable* `t`*. Lets suppose we sample 5 values.*

$$\mathtt{t}_{MC} \in \{12.8, 35.1, 17.6, 22.2, 21.4\} \tag{3.17}$$

and plug these samples into Ψ. We map the Boolean random variable `no_cool` *to a $1D$ tensor whose entries are* 0.01*. Consulting the arithmetic circuit in Figure 3.1, we easily see that we obtain for the MC estimate:*

$$\begin{aligned}\Psi_{MC} &= \begin{bmatrix}0.01\\0.01\\0.01\\0.01\\0.01\end{bmatrix} \circ \begin{bmatrix}[12.8{>}20]\\[35.1{>}20]\\[17.6{>}20]\\[22.2{>}20]\\[21.4{>}20]\end{bmatrix} \circ \begin{bmatrix}[12.8{\leq}30]\\[35.1{\leq}30]\\[17.6{\leq}30]\\[22.2{\leq}30]\\[21.4{\leq}30]\end{bmatrix} + \begin{bmatrix}[12.8{>}30]\\[35.1{>}30]\\[17.6{>}30]\\[22.2{>}30]\\[21.4{>}30]\end{bmatrix} \\ &= \begin{bmatrix}0.01\\0.01\\0.01\\0.01\\0.01\end{bmatrix} \circ \begin{bmatrix}0\\1\\0\\1\\1\end{bmatrix} \circ \begin{bmatrix}1\\0\\1\\1\\1\end{bmatrix} + \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0.01\\0.01\end{bmatrix}\end{aligned} \tag{3.18}$$

where $\circ$ denotes the elementwise multiplication of tensors. With the Monte Carlo estimate of Ψ we obtain the MC estimate for the weighted model integral by simply averaging:

$$WMI_{MC} = \frac{1}{5}\sum_{i=1}^{5} \Psi_{MC,i} = 1.02/5 = 0.204$$

Compiling an SMT formula and transforming the resulting arithmetic circuit into a computation graph has the advantage that sampling becomes embarrassingly parallelizable. To the best of our knowledge, Sampo is the first probabilistic inference algorithm for the hybrid domain that is able to harness parallelization on a GPU.

3.4.3 Discussion on Complexity

The complexity of Symbo and Sampo is mainly determined by the complexity of their subcomponents. The knowledge compilation step is #P-hard. The evaluation of the resulting arithmetic circuit is done in polytime. Symbo, however, suffers from the problem that the search for simplifications in symbolic expressions is a hard problem. One such simplification is the symbolic integration step itself. For example, integrating convex polytopes is #P-hard [Dyer; Frieze, 1988; Koutis, 2003; Koutis, 2003]. These complexity concerns do not hold for Sampo, as we are dealing with mere additions and multiplications on the GPU. In the next section the computational complexity of symbolic simplifications becomes experimentally apparent in the ClickGraph benchmark for Symbo (cf. Table 3.1).

In the case of weighted model counting the time to evaluate the arithmetic circuit is polynomial in time polynomial in size of the circuit. As discussed in the previous paragraph this does not hold anymore for WMI. While WMC consists of an expensive compilation step and a cheap evaluation step. WMI consists of two expensive steps!

3.5 Experimental Evaluation

In the previous section we have developed two algorithms that perform weighted model integration for weight functions in the form of probability density functions. Because general probability density functions are common in probabilistic programs, but have only been approximated in existing weighted model integration algorithms (using piecewise polynomial weight functions), we compare Symbo and Sampo with state-of-the-art inference algorithms in probabilistic programming.

To this end, we extended the syntax of the probabilistic programming system ProbLog2 [Dries et al., 2015], so that it allows for the use of abstractions of atomic formulas and for the declaring how continuous random variables are distributed. ProbLog2 implements inference for the probabilistic programming language aProbLog [Kimmig et al., 2011], where inference is done through algebraic model counting.

We are interested in three main questions during the experimental evaluation of Symbo and Sampo.

Q1 How does Symbo, a logico-symbolic solver, compare to a pure, state-of-the-art, symbolic solver for the hybrid domain?

Q2 How does Sampo compare to related state-of-the-art probabilistic inference algorithms?

Q3 In the interest of completeness we also adopted Symbo to solve traditional weighted model integration problems, where the weight function is expressed as a polynomial function.

We answer **Q1** by comparing Symbo, which uses the PSI-Solver and combines it with KC, to pure symbolic inference with the PSI-Solver.

For **Q2**, we compare Sampo to the inference algorithms of Distributional Clauses (DC) [Nitti et al., 2016a][4], BLOG [Milch et al., 2005][5] and to Hybrid Probabilistic Model Counting (IHPMC) [Michels et al., 2016][6]. These are state-of-the-art probabilistic programming systems that all support first order logic as well as hybrid representations.

In **Q3** we compare Symbo to the existing WMI solver of [Morettin et al., 2017], which uses predicate abstraction, SMT solving and numerical integration, and to the solver of [Kolb et al., 2018], which uses XADDs [Sanner; Abbasnejad, 2012] and hence symbolic integration.

Experiments were performed on a laptop Intel(R) i7 CPU 2.60GHz with 16 Gb memory, running Linux OS. Sampo took additionally advantage of an NVIDIA Quadro M1000M.

Q1 (Symbo): We compared Symbo and the PSI-Solver on the set of benchmark experiments given in Gehr et al., 2016, section F of Appendix[7].

In Table 3.1, we observe that Symbo outperforms the PSI-Solver for 9/10 benchmarks, for 7/10 even when including the time spent on the knowledge compilation step. Only for the ClickGraph benchmark does PSI perform better than Symbo, which timed-out after 15s during circuit evaluation. This is because PSI integrates out variables after loop iterations. This is not yet supported in the ProbLog implementation and Symbo ends up with a large symbolic expression that is hard to integrate over. This could be solved, for example, by using sub-queries, as can be done in ProbLog2.

We note that the symbolic inference engine underlying the PSI-Solver has until now only been used for imperative programing. The implementation of Symbo shows that the powerful symbolic inference engine can also be adopted for logic programming when making use of KC.

To conclude, it is generally beneficial to perform logical inference on top of symbolic inference in the hybrid domain.

[4] https://bitbucket.org/problog/dc_problog
[5] https://bayesianlogic.github.io
[6] https://github.com/SteffenMichels/IHPMC
[7] cf.: Fun [Minka et al., 2014] and R2 [Nori et al., 2014]

Benchmark	KC	Evaluation	PSI	Domain
BurglarAlarm	31.4	0.8	190.1	D
CoinBias	41.9	7.9	12.9	H
Grass	31.2	1.2	228.0	D
NoisyOR	35.8	11.2	12.7	D
TwoCoins	27.0	2.1	57.8	D
ClickGraph	4300	–	10500	H
ClinicalTrial	54.6	25.7	3400	H
AddFun/max	25.2	4.4	53.1	H
AddFun/sum	27.1	2.1	84.9	H
MurderMystery	27.6	0.3	65.4	D

Table 3.1: Knowledge compilation and arithmetic circuit evaluation times for Symbo, and problem solving time for PSI. Times are given in ms. Run times were averaged over 50 runs. We omitted the standard deviations on the run time as they are negligible. The domain column indicates whether the problem is **D**iscrete or **H**ybrid.

Q2 (Sampo): In order to evaluate Sampo, we chose benchmarks from [Nitti et al., 2016a] and [Michels et al., 2016], which were stated to be hardest in terms of query complexity. We show our results in Figures 3.2 and 3.3. In Figure 3.2, we compare Sampo to DC and BLOG. A comparison with IHPMC for this first problem is not possible as IHPMC does not allow for expressing hierarchical models. DC an BLOG are, just like Sampo, sampling based methods, which use both importance sampling and likelihood weighting.[8] This is why we plot the evaluation time and the standard deviation in function of the number of samples. IHPMC is not a sampling based method but iteratively splits up the space into mutually exclusive pieces and calculates bounds for each piece, which translates to iteratively tighter and tighter error bounds. For this reason we investigate in the plots in Figure 3.3 the standard deviation of the four methods scrutinized in function of the run time.

All four plots clearly indicate that once Sampo has transformed a probabilistic program into an arithmetic circuit, the run time is not only lower but also that Sampo is more accurate than the competing algorithms. This is especially true for two distinct cases. Firstly, when there are binary random variables present. Contrary to DC and BLOG, Sampo does not sample these random variables but includes their probability as weight in the circuit evaluation. This can be seen Figure 3.2a and 3.3a. The reason why the STD is not zero for Sampo in 3.2a is due to floating point rounding errors. The second case where Sampo clearly outperforms the other methods is when we condition on low probability events, cf. Figure 3.3b. Here we condition on an event that has probability 0.0001 to occur. The logic structure of the problem implies that the query

[8] BLOG also provides rejection sampling and MCMC.

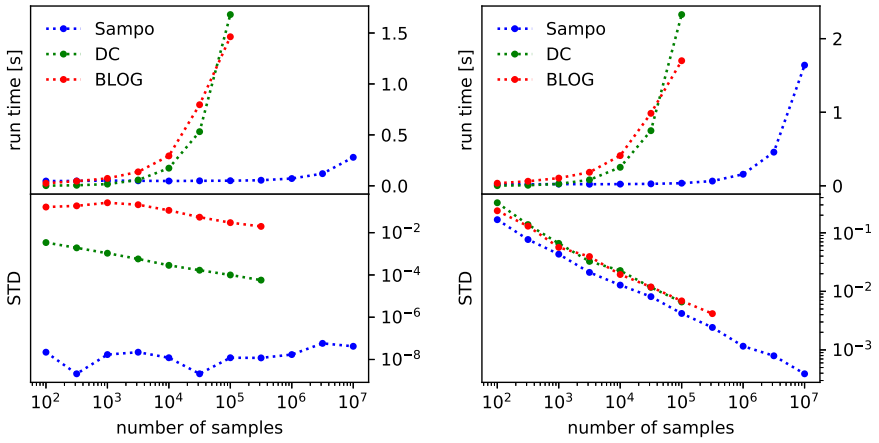
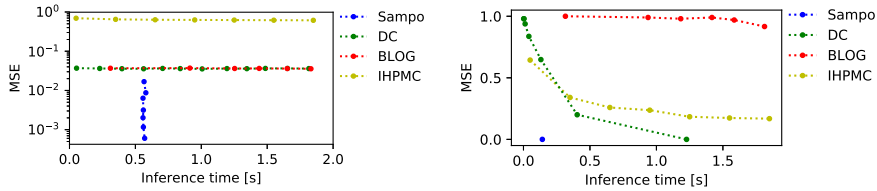


Figure 3.2: The example used is *drawing balls (denoted by b_i and having different size, color and material) with replacement from an urn* [Nitti et al., 2016a]. The queries used are (a) $p(b_1{=}b_2 \wedge col(b_1){=}black)$ and (b) $p(b_1{=}1|0.39{<}size(b_1){<}0.41)$. The upper panel shows the run time (circuit evaluation for Sampo). Evaluation runs are averaged over 50 runs and the knowledge compilation step is averaged over 50 compilations. Time-out was set at 2.5s. The linear behavior of Sampo towards higher sample numbers is due to the GPU starting to run out of memory. Sampo spent 1.62s for (a) and 0.11s for (b) on the knowledge compilation step, averaged over 50 runs.

given the observation must be satisfied. In Figure 3.3b we see that Sampo is the only algorithm that picks up this structure. As the inference reduces to inference on exclusviely Boolean random variables, Sampo immediately finds the correct solution without drawing any samples for continuous random variables, in contrast to the other algorithms.

We also observe that there is practically no time penalty for the number of samples for Sampo, contrary to DC and BLOG. This behavior manifests itself most prominently in the upper panel of Figure 3.2a and in Figure 3.3a. For the latter, we see that higher sample numbers, which correspond to lower STDs, take up just as much time as lower sample numbers. This produces the quasi-vertical line Figure 3.3a. This behavior is due to delegating the N evaluations of the arithmetic circuits, which correspond to N times sampling the continuous random variables, to the GPU and executing the evaluation in parallel. Only in Figure 3.2a we observe a linear dependency of the run time in function of the number of samples towards high sample numbers. This is caused by the GPU running out of memory.



(a) query: $p(f_{99})$

(b) query: $p(f_{99}|f_0)$

Figure 3.3: We show the dependencies of the mean squared error on time for two queries of the theory: $f_i \leftrightarrow d_i \vee c > l_i \vee f_{i-1}$, cf. [Michels et al., 2016]. f_i and d_i are Bools. The probability of d_i being true is 0.0001. c and l_i are normally distributed variables with mean 20 and 30 respectively and standard deviation 5. Note the two different scales for the plots on the y-axis. The mean squared errors are averaged over 50 runs. The average KC time (over 50 iterations) is 4.34s for (a) and 2.66s for (b). In the left plot the mean squared error was calculated with respect to the mean of 50 runs using Sampo with 10^5 samples. In the right plot, we stopped when all the runs for a given number of samples for an algorithm reached the correct solution (which is 1.0) or the algorithm timed-out after 2s.

Q3 (WMI): By allowing Symbo to handle also bounded polynomial weights, instead of probability density distributions, we can compare Symbo to the existing exact WMI solvers of WMI-PA [Morettin et al., 2017] and WMI-XADD [Kolb et al., 2018]. This extension of Symbo is necessary as these solvers are limited to polynomial weights and cannot handle proper probability densities.

We made the experimental comparison of the three methods on a set of synthetic problems given in [Morettin et al., 2017][9]. The benchmarks consist of WMI problems that have from five to seven Boolean variables and where the weight functions are multivariate polynomials of dimensions two to three.

We compared the three methods on the benchmarks with two dimensional polynomial weights. We observe in Figure 3.4 that Symbo solves the majority of the problems with bivariate polynomials faster than the other two methods. We omit the comparison plot for the benchmarks with three dimensional polynomials, as here the other methods, which are specialized algorithms for polynomial weight functions, beat Symbo at large. Symbo spends most of the time on the final integration step (cf. point 7 in Algorithm 3.1). In fact, Symbo spent at most 0.32s on the KC step, at most 0.34s on the circuit evaluation and any remaining time on the symbolic integration - for any of the presented benchmarks. Using a dedicated integrator for bounded polynomials instead of the generic PSI integrator could mitigate this problem. In the next chapter we will

[9] https://github.com/unitn-sml/wmi-pa

investigate integrating out variables during the evaluation of the arithmetic circuit, as this leads to smaller symbolic expressions.

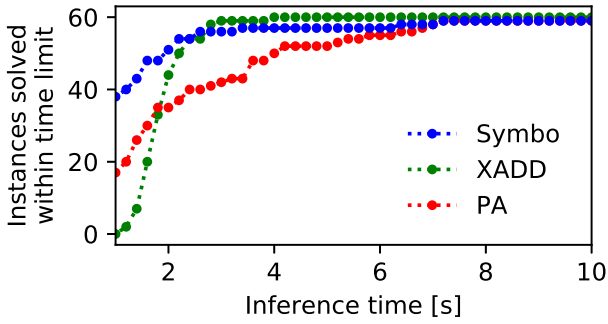


Figure 3.4: We show the number of problem instances solved below the time limit for problems with bivariate polynomial weights.

3.6 Related Work

In the initial work on weighted model integration [Belle et al., 2015a] the authors perform weighted model integration on piecewise polynomials by iteratively generating models by adding the negation of the model from the previous iteration to the formula. In subsequent work by [Morettin et al., 2017] the number of generated models is substantially reduced by deploying SMT-based predicate abstraction [Graf; Saïdi, 1997]. In this line of work [Belle et al., 2016] also investigated component caching while performing a DPLL search when calculating a weighted model integral. Their approach is indeed related to knowledge compilation. However, it is not applicable in cases when algebraic constraints exist between variables and couple these. The methods proposed on WMI are strictly limited to piecewise polynomials. We, completely lift this restrictions and are able to perform WMI via knowledge compilation on SMT($\mathcal{RA}$) and SMT($\mathcal{NRA}$) formulas using probability density functions instead of piecewise polynomials on SMT($\mathcal{LRA}$).

As seen in Section 3.5, WMI has also been studied in the context of XADDs [Kolb et al., 2018] and the approach is closely related to Symbo. Here again, the weight functions considered are only of polynomial form. Another drawback of this approach is that, unlike for the SDDs and d-DNNFs used in our approach, there are not yet any efficient compilers available for converting WMI to XADDs. For SDDs and d-DNNFs, one can employ standard state-of-the-art KC technology. SDDs are more succinct than BDDs, of which XADDs are an extension. This entails, in turn, that SDDs are more

succinct than XADDs. Standard knowledge compilation techniques are not readily available for XADDs as Kolb et al.'s compilation algorithm interleaves symbolic and logic inference.

Somewhat related to WMI with piecewise polynomials is the work of [Gutmann et al., 2010], who restricted distributions to Gaussians, which are chopped up into easily integrable and axis-aligned pieces.

Contrary to these works, we generalize WMI by providing a much larger class of weight functions and constraints.

With respect to inference for probabilistic programming in the hybrid domain, two classes of algorithms exist: approximate and exact. Firstly, for what concerns one of the few exact inference systems, there is the already mentioned work for imperative probabilistic programming [Gehr et al., 2016], which has contributed the PSI-Solver that we use in Symbo. The PSI-Solver beats other recent approaches in exact probabilistic inference [Narayanan et al., 2016]. We show that knowledge compilation speeds up pure symbolic inference and that Symbo outperforms the PSI-Solver.

Another approach, related to exact inference in probabilistic logic programming, is that of [Islam et al., 2012]. Similarly to Symbo, they symbolically evaluate a theory in order to obtain an expression for a probability density. However, their approach is restricted to Gaussians (although gamma distributions are in theory also implementable), and more importantly it is built on top of Prism [Sato, 1995], which assumes that proofs are mutually exclusive, and which avoids the disjoint sum problem. As a consequence they do not support WMI in its full generality. Supporting WMI requires the KC step, which they do not address.

Secondly, for what concerns approximate inference, we have the sampling approaches in Distributional Clauses by [Gutmann et al., 2011; Nitti et al., 2016a] and BLOG by [Milch et al., 2005], which we have already discussed in Section 3.5 and which both deploy importance sampling in order to sample from probability distributions and densities alike, combined with likelihood weighting.

Approximate inference is also performed in [Michels et al., 2016]. In their work, a hybrid probabilistic problem is represented by so called *hybrid probability trees* (discussed in Section 3.5). Our experiments show that Sampo outperforms DC, BLOG and IHPMC. Moreover, Sampo has the advantage that when conditioning on rare events in the discrete domain, we still obtain reliable estimates of the weighted model integral. Using pure sampling based methods such as in Distributional Clauses and BLOG leads to poor results. This is known to be problematic when using importance sampling based methods.

Note that when conditioning on rare events in continuous domains, Sampo performs as poorly as other sampling techniques as it performs essentially rejection sampling with

almost all samples being rejected.

3.7 Conclusions

We have shown how standard knowledge compilation can be applied to the task of weighting model integration by leveraging algebraic model counting and thereby presenting a unified formalism for weighted model integration and knowledge compilation. We have also introduced an exact and an approximate solver based on this idea and demonstrated their effectiveness. Sampo is to the best of our knowledge the first sampling based algorithm deployable in the WMI setting.

Chapter 4

Exploiting Factorizability*

Solving WMI problems in the SMT($\mathcal{LRA}$) domain consists of two sub-problems 1) finding convex polytopes, and 2) integrating over them efficiently.

We formalize the first step as λ-SMT and discuss what strategies solvers apply to solve both the λ-SMT and the integration problem. This formalization allows us to compare state-of-the-art solvers and their behavior across different types of WMI problems. Moreover, we identify factorizability of WMI problems as a key property that emerges in the context of probabilistic programming. Problems that can be factorized can be solved more efficiently. However, current solvers exploiting this property restrict themselves to WMI problems with univariate conditions and fully factorizable weight functions. We introduce a new algorithm, F-XSDD, that lifts these restrictions and can exploit factorizability in WMI problems with multivariate conditions and partially factorizable weight functions. Through an empirical evaluation, we show the effectiveness of our approach.

4.1 Introduction

Contrary to solvers for WMC, the relative advantages and drawbacks of the different WMI solvers are not yet well understood. Understanding these solvers and their differences requires a clear separation of the model counting and the integration component. The existing attempts [Belle et al., 2015a; Morettin et al., 2017] to separate these two components are all tied to specific solving paradigms. To decouple the formulation in a general way, we, as a **first contribution**, introduce λ-SMT, the

*This chapter has previously been published as [Kolb et al., 2019b].

problem of rewriting a generic WMI problem into a sum of integrals over convex polytopes. This allows us to formally disentangle the *model counting* (solving λ-SMT) and *integration* steps (computing integrals). As a **second contribution**, we discuss the main paradigms used to solve the λ-SMT step – DPLL search and knowledge compilation – and the integration step – numeric and symbolic integration.[1] This allows us to compare different state-of-the-art solvers and understand how their design choices affect the kind of WMI problems they are able to solve efficiently. Finally, we observe that *fully factorizable* WMI problems have given rise to efficient solvers for subsets of WMI [Belle et al., 2016; Molina et al., 2018]. While *factorizability* naturally emerges in probabilistic programming, the strong conditions imposed by full factorizability – no multivariate conditions and fully factorizable weight functions – fail to cover most applications. Therefore, as a **third contribution**, we present a novel algorithm to solve generic WMI problems that can automatically exploit *factorizability* in the problem structure.

4.2 λ-SMT

In the weighted model integration literature the weight function w of a WMI problem $\text{WMI}(\phi, w|\mathbf{x}, \mathbf{b})$ over $\text{SMT}(\mathcal{LRA})$ formulas and polynomial weight functions is expressed as an AST with $\mathcal{LRA}$ atoms and polynomials. The class of functions expressible by these ASTs is equivalent to the class of piecewise-polynomial case functions, as shown in [Kolb et al., 2018]. A piecewise-polynomial case function $f = \{\phi_1 : \omega_1, \cdots, \phi_n : \omega_n\}$ consists of tuples $\langle \phi_i, \omega_i \rangle$, where ϕ_i is a conjunction of $\text{SMT}(\mathcal{LRA})$ literals and ω_i is a polynomial over $\mathbf{x}$. The *world-supports* ϕ_i form a partition of the space spanned by the Cartesian product $\mathbf{x} \times \mathbf{b}$, i.e., they are mutually exclusive (disjoint) and exhaustive (covering the whole space). Note that, as we consider only $\mathcal{LRA}$ atomic formulas, every ϕ_i corresponds to a convex polytope in the real space (contrary to the given support ϕ). All assignments to the variables in $\mathbf{x}$, $\mathbf{b}$ that satisfy ϕ_i are weighted with the polynomial *world-weight* ω_i, which no longer relies on $\mathbf{b}$.

Example 4.1. *Consider the weight function*

$$w(\{x, y\}, \{a\}) = \text{ite}(a, 2x + y, x^2 y) \times \text{ite}((y < 5), 3, 0)$$

where x and y are real variables, and a is a Boolean variable. Moreover, consider the SMT(LRA) formula

$$\phi = ((a \wedge (x < 5)) \vee (x > y)) \wedge \mathit{bounds}$$

[1] We understand a numeric integration method as a method that outputs the value of a definite integral (not necessarily obtained through numeric approximations) and symbolic integration as the problem of finding the anti-derivative

with

$$bounds = (x{<}10) \wedge (x{>}2) \wedge (y{<}10) \wedge (y{>}2)$$

We can then partition the space with the following case functions:

$$\begin{cases} (y < 5) \wedge (x < 5) \wedge bounds \wedge a & : 6x + 3y \\ (y < 5) \wedge (x \geq 5) \wedge (x > y) \wedge bounds \wedge a & : 6x + 3y \\ (y < 5) \wedge (x > y) \wedge bounds \wedge \neg a & : 3x^2 y \end{cases} \tag{4.1}$$

We omitted the cases where the polynomial weight is 0*.*

By using the notion of case functions, the weighted model integral (Equation 2.10) can be rewritten as a sum of integrations of polynomials over convex polytopes. In order to show this, we first write the definite sum and integral as an indefinite sum and integral of indicator functions multiplied by the weight function w.

$$\begin{aligned} \mathrm{WMI}(\phi, w|\mathbf{x}, \mathbf{b}) &= \sum_{\mathbf{b}_I \in \mathcal{I}_{\mathbf{b}}(\phi)} \int_{\mathcal{I}_{\mathbf{x}}(\phi^{\mathbf{b}_I})} w(\mathbf{x}, \mathbf{b}_I) d\mathbf{x} \\ &= \sum_{\mathbf{b}} \int [\![\phi(\mathbf{x}, \mathbf{b})]\!] w(\mathbf{x}, \mathbf{b}) d\mathbf{x} \end{aligned} \tag{4.2}$$

We continue manipulating this expression by 1) rewriting $[\![\phi(\mathbf{x}, \mathbf{b})]\!] w(\mathbf{x}, \mathbf{b})$ as a case-function $f(\mathbf{x}, \mathbf{b})$ with tuples $\mathcal{W}_f = \{\langle \phi_i, \omega_i \rangle\}$, 2) exploiting the mutual exclusivity of the world-supports to rewrite $f(\mathbf{x}, \mathbf{b}) = \sum_{\langle \phi_i, \omega_i \rangle \in \mathcal{W}_f} [\![\phi_i(\mathbf{x}, \mathbf{b})]\!] \omega_i(\mathbf{x})$, and 3) reordering the summations:

$$\sum_{\mathbf{b}} \int f(\mathbf{x}, \mathbf{b}) d\mathbf{x}$$

$$= \sum_{\mathbf{b}} \int \sum_{\langle \phi_i, \omega_i \rangle \in \mathcal{W}_f} [\![\phi_i(\mathbf{x}, \mathbf{b})]\!] \omega_i(\mathbf{x}) d\mathbf{x} \tag{4.3}$$

$$= \sum_{\langle \phi_i, \omega_i \rangle \in \mathcal{W}_f} \sum_{\mathbf{b}} \int [\![\phi_i(\mathbf{x}, \mathbf{b})]\!] \omega_i(\mathbf{x}) d\mathbf{x} \tag{4.4}$$

Finally, we push the Iverson brackets back into the index and the bound of the summation and integral, respectively.

$$\sum_{\langle \phi_i, \omega_i \rangle \in \mathcal{W}_f} \sum_{\mathbf{b}_I \in \mathcal{I}_{\mathbf{b}}(\phi_i)} \int_{\mathcal{I}_X(\phi_i^{\mathbf{b}_I})} \omega_i(\mathbf{x}) d\mathbf{x} \tag{4.5}$$

In accordance to the terms world-support and world-weight, we define the *world-volume* vol w.r.t. to a tuple $\langle \phi_i, \omega_i \rangle$ as:

$$\text{vol}(\phi_i, \omega_i | \mathbf{x}, \mathbf{b}) = \sum_{\mathbf{b}_I \in \mathcal{I}_{\mathbf{b}}(\phi_i)} \int_{\mathcal{I}_{\mathbf{x}}(\phi_i^{\mathbf{b}_I})} \omega_i(\mathbf{x}) d\mathbf{x} \tag{4.6}$$

Solving any WMI problem can thus be reduced to a two step procedure: 1) rewriting the problem into a sum over tuples $\langle \phi_i, \omega_i \rangle$ of disjoint world-supports and world-weights (i.e., convex polytopes and polynomials), and 2) integrating every world-weight ω_i over the corresponding world-support ϕ_i. We formally define the first step of this procedure as λ-*SMT*:

Definition 4.1. *(λ-SMT)* ***Given*** *a WMI problem* $\text{WMI}(\phi, w | \mathbf{x}, \mathbf{b})$*,* ***find*** *a set $\mathcal{W}$ of pairwise logically inconsistent world-supports ϕ_i (i.e., their conjunction is unsatisfiable) and world-weights ω_i such that the sum over their world-volumes is equal to the weighted model integral:*

$$\sum_{\langle \phi_i, \omega_i \rangle \in \mathcal{W}} \text{vol}(\phi_i, \omega_i | \mathbf{x}, \mathbf{b}) = \text{WMI}(\phi, w | \mathbf{x}, \mathbf{b}) \tag{4.7}$$

We call a tuple $\langle \phi_i, \omega_i \rangle \in \mathcal{W}$ redundant *if ϕ_i is not satisfiable (i.e., logically inconsistent) or $\omega_i = 0$, since integrating over them yields 0. Further, we call a λ-SMT solution $\mathcal{W}^*$* reduced*, if no element in $\mathcal{W}^*$ is redundant.*

The λ-SMT problem lies at the heart of all WMI solvers, and it is easy to see that such sets $\mathcal{W}$ always exist, as any WMI problem can be written as a case-function (see above and [Kolb et al., 2018]) and enumerating all cases yields a solution to the λ-SMT problem. In order to obtain a reduced solution, we can discard all redundant cases – doing this efficiently is a key part of WMI-solving.

Conceptually similar 2-step decompositions of WMI solving can be found in prior works. In [Morettin et al., 2017, Definition 5], the authors separate the steps of finding truth assignments to the Boolean variables and solving non-Boolean WMI problems (WMI_{nb}). In [Kolb et al., 2018, Section 3], a case function representation is built by compiling the WMI problem to an XADD. We continue this discussion in Section 4.4.

4.3 Anatomy of a Solver

With the formal definition of λ-SMT we can now study how different solvers handle this problem and how the representation of the solution to the λ-SMT problem influences strategies to solve the subsequent integration step. Even though the λ-SMT and the

integration steps can be intertwined, there are broadly two ways to tackle the λ-SMT step, and two ways to tackle integration.

4.3.1 λ-SMT: Search vs Compilation

λ-SMT concerns the analysis of the structure of the WMI problem, finding a way to rewrite the problem into disjoint, convex polytopes with purely polynomial weight functions. The techniques used to solve this part of the problem relate closely to techniques used for WMC and #SAT. On the one hand, solvers such as PA [Morettin et al., 2017] and PRAiSE [De Salvo Braz et al., 2016] use variants of DPLL search to enumerate the conditions of the polytopes over which to integrate. On the other hand, XADD-based solvers [Sanner et al., 2011; Kolb et al., 2018] and SDD-based Symbo [Zuidberg Dos Martires et al., 2019b] use knowledge compilation to compile the problem structure into a compilation language in which the solutions to the λ-SMT problem are efficiently represented.

A key aspect to solving λ-SMT efficiently is detecting redundant tuples $\langle \phi, \omega \rangle$ early in the solving process. For many solvers, the support ϕ of a WMI problem plays a crucial role in avoiding the enumeration of redundant tuples by restricting their search to the feasible space described by the support.

Solvers relying on compilation aim to find small representations for λ-SMT solutions. By checking for redundant tuples during or after the compilation, they try to find reduced λ-SMT solutions. Additionally, they often *compress* the representation of an λ-SMT solution by grouping tuples $\langle \phi_1, \omega \rangle, ..., \langle \phi_n, \omega \rangle$ that have the same world-weight ω as $\langle \bigvee_i \phi_i, \omega \rangle$ (e.g., by using DAGs [Kolb et al., 2018]).

The set $\mathcal{W}$ can be further compressed when a solver supports the concept of a *labeling function*. The labeling function originates from the WMC literature, where a labeling function α maps literals to real-valued weights. This allows them to factorize the weight function over the Boolean variables: $w = \prod_{b \in \mathbf{b}} \alpha_b(b)$. The Symbo solver [Zuidberg Dos Martires et al., 2019b] reuses the notion of a labeling functions to *partially* factorize the weight function w as $w(\mathbf{x}, \mathbf{b}) = w'(\mathbf{x}, \mathbf{b}) \cdot \prod_{b \in \mathbf{b}} \alpha_b(b)$ or $w'(\mathbf{x}, \mathbf{b}) \cdot \prod_{b \in \mathbf{b}} \alpha_b(\mathbf{x}, b)$. Consider two tuples $\langle \phi_1, \omega_1 \rangle$ and $\langle \phi_2, \omega_2 \rangle$ of a λ-SMT solution, where $\phi_1 = \phi_s \wedge b$ and $\phi_2 = \phi_s \wedge \neg b$ and $b \in \mathbf{b}$. If, then, $\omega_1 = \omega_s \cdot \alpha_b(\texttt{true})$ and $\omega_2 = \omega_s \cdot \alpha_b(\texttt{false})$, we can group these cases as $\langle \phi_s, \omega_s \cdot \alpha_b(b) \rangle$. Labeling functions over multiple literals can thus allow a number of tuples exponential in the number of literals to be grouped together (e.g., within a single SDD [Zuidberg Dos Martires et al., 2019b]). Note, that we now have to relax the fact that the world weight does not depend on $\mathbf{b}$. We can reintroduce this dependency on $\mathbf{b}$ as long as for any Boolean instantiation, the world-weight will be polynomial.

4.3.2 Numeric vs Symbolic Integration

With respect to integration, WMI solvers fall into two categories: those using *numeric* integration, and those using *symbolic* integration. Given the function w and the set of free variables $\mathbf{x}$ to integrate, numeric integration approaches directly compute the result as a real number. Symbolic integration approaches will instead integrate out the variables one-by-one, obtaining symbolic intermediate results (i.e., repeated variable elimination). Integrating out a variable that has multiple symbolic lower- or upper-bounds causes these expressions to grow quickly. Therefore, numeric integration procedures are usually more efficient at performing individual integrations.

There are, however, two key advantages of symbolic integration, as demonstrated in [Kolb et al., 2018]. First, symbolic integration can be used to solve structured problems by reusing intermediate results across different $\text{vol}(\phi_i, \omega_i)$ computations. In doing so, it aims to avoid having to compute a potentially exponential number of individual integrations. This reuse resembles caching in traditional DPLL search and exploits the compression of the λ-SMT models discussed in the previous section. Secondly, when computing the probabilities of multiple queries Q over a small subset of variables $\mathbf{x}_Q$, the result r of integrating out all variables except those in $\mathbf{x}_Q$ can be computed symbolically and then reused to quickly compute query probabilities, integrating out $\mathbf{x}_Q$ from $r \cdot q, \forall q \in Q$ [Kolb et al., 2018]. Note that this inference scheme is similar to the *compile once, query multiple times* knowledge compilation approach.

4.4 Categorizing Existing Solvers

In this section we categorize current solvers by analyzing how they handle both the λ-SMT problem and integration.

Predicate Abstraction [Morettin et al., 2017] (PA) The PA solver uses the MathSAT [Cimatti et al., 2013] SMT solver to solve WMI problems. First, it solves the λ-SMT problem by introducing fresh Boolean variables for the SMT conditions in the weight function and performing a two-step DPLL search. In the first step, it finds truth assignments to the original and fresh Boolean variables. Such an assignment defines a set $\mathcal{W}_i$ of world-supports (defined by the original variables) with a common world-weight (defined by the fresh variables). If, after using the SMT solver to remove redundant tuples from $\mathcal{W}_i$, $\mathcal{W}_i$ contains more than one world-support, the second step enumerates them by finding truth assignments to the $\mathcal{LRA}$ atoms in the support. The PA solver avoids enumerating sets of equivalent world-supports by using *partial assignments*. Second, for every world-support ϕ_i, it uses numeric integration to integrate

the polynomial over the convex polytope specified by ϕ_i. For the integration it relies on the LattE Integrale integration software [De Loera et al., 2013b].

Bound Resolution [Kolb et al., 2018] (BR) The BR solver tackles a WMI problem by first compiling it to an equivalent XADD, which represents a solution to the λ-SMT problem. Compilation is performed through a bottom-up compilation scheme[2], using recursive *apply* operations. Pruning unsatisfiable paths within XADDs can be done using an LP or SMT oracle. While it is an expensive operation, it is crucial to keep the XADDs small and efficient. Paths in the XADD correspond to tuples in $\mathcal{W}$ and the DAG structure of the XADD compresses paths with the same world-weight. Instead of performing integration separately for every path and summing the results (which could be done using symbolic or total integration), the BR algorithm exploits overlapping paths using symbolic integration. The algorithm recursively integrates every variable, tracking only one pair of upper- and lower-bound at a time, and building the result of the integration dynamically as a new XADD (integration is a closed operation for polynomial case functions with linear conditions).

Symbo [Zuidberg Dos Martires et al., 2019b] The Symbo solver first computes a compressed representation of an λ-SMT solution $\mathcal{W}$ in the form of pairs of XSDDs (representing $\bigvee_i \phi_i$)[3] and distinct world-weights ω. XSDD compilation is performed bottom-up and, as current SDD software is strictly discrete, fresh Boolean variables are introduced as abstractions for atomic SMT formulas. Currently, pruning redundant tuples within the circuit is not supported, instead, unfeasible world-supports are detected only later, at the integration stage. Symbo supports labeling functions (see Section 4.3.1), which enables a more compressed representation of $\mathcal{W}$. Solving and integrating requires a bottom-up pass over the circuit that reassembles the tuples $\mathcal{W}$, checking for redundancy during the assembly and, finally, integrating the different (weighted) tuples using a symbolic integration engine. Both inconsistency checking and symbolic integrations are performed using the computer algebra system PSI [Gehr et al., 2016].

PRAiSE [De Salvo Braz et al., 2016] Another DPLL based solver is PRAiSE, which, contrary to PA, performs symbolic integration, representing intermediate results as symbolic expression trees. These expression trees can be seen as tree-based alternatives to XADDs. They represent a compiled version of the problem *after*

[2] The authors refer to their compilation as top-down, however, within the knowledge compilation literature, this type of recursive compilation is commonly referred to as bottom-up.

[3] Zuidberg Dos Martires et al. did not use the term *XSDD*. This nomenclature was introduced in [Kolb et al., 2019b] to refer to SDD with atomic SMT formulas in the leaves. In practice, XSDDs are implemented using traditional SDD software and additional book-keeping for the non-Boolean literals (cf. Chapter 3).

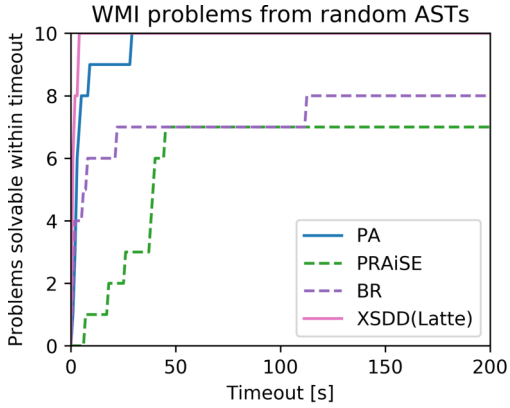


Figure 4.1: We test the performance of several solvers on a set of random WMI problems generated using the PA benchmark generator. For problems that have little structure and dense inequalities, solvers using numeric integration (full lines) perform better than solvers using symbolic integration (dashed lines). Since the problems are relatively shallow, the differences in solving the λ-SMT problem are secondary (runtimes include compilation).

(symbolic) integration. However, as PRAiSE does not keep a trace of its DPLL search, it is foremost a search based approach.

Other solvers As mentioned in the section on PA, other DPLL based methods exist, however we consider them superseded by PA. A functionally different solver is **CC** solver [Belle et al., 2016], which also performs DPLL search over abstractions of SMT formulas but introduces component caching to compute solutions more efficiently. However, the solver is restricted to a subset of WMI, requiring axis-aligned (univariate) $\mathcal{LRA}$ atoms in the real space and weight functions that factorize over all Boolean and real variables [Belle et al., 2016, Proof Theorem 6].

We also introduce a solver denoted as *XSDD(*LattE*)*. XSDD(LattE) follows the same strategy as Symbo to solve the λ-SMT problem using SDD compilation, but it then builds a list of all convex polytopes by doing a bottom-up evaluation of the circuit and evaluates them using numeric integration with Latte.

Discussion The solver design choices (see Table 4.1 for an overview), search vs compilation, and numeric vs symbolic integration, as well as how to implement them have a big influence on what problems the solver will be able to solve efficiently. On the one hand, solving the λ-SMT problem through DPLL avoids a potentially expensive

Table 4.1: Overview of the solvers discussed and their properties.

	PA	BR	Symbo	PRAiSE
λ-SMT				
DPLL	✓			✓
Compilation		XADD	XSDD	
Integration				
Numeric	Latte			
Symbolic		XADD	PSI (Tree)	Exp. Tree

compilation step and is less sensitive to the problem formulation. This allows the PA solver to efficiently tackle problems like their road-network problem [Morettin et al., 2017]. On the other hand, compiling a circuit representation of λ-SMT can avoid recomputing λ-SMT solutions as the circuit can be efficiently combined with other circuits (representing, e.g., queries for conditional inference). Moreover, solving structured problems in which many world-supports share a common (base) world-weight, for example, problems using mutual exclusivity constraints over terms, and computing multiple query probabilities benefit from knowledge compilation and symbolic integration [Kolb et al., 2018]. Caching in DPLL-based approaches is used in a similar spirit, however, no current DPLL-based WMI solver exploits caching for WMI problems with non-axis aligned SMT($\mathcal{LRA}$) atoms. When integration steps are not reusable, numeric integration approaches will fare better than symbolic integration for single WMI queries (see Fig. 4.1).

4.5 Exploiting Factorizability of WMI Problems

There is an additional type of structure that has been exploited in the WMI literature to speed up inference: *factorizable* WMI problems, which is a subset of WMI [Belle et al., 2016; Molina et al., 2018]. However, it is usually subjected to the strong constraints of using only axis-aligned $\mathcal{LRA}$ atoms and *fully* factorizable weight functions. In this subset of WMI problems, the computation of $\text{vol}(\phi, \omega)$ for any tuple $\langle \phi_i, \omega_i \rangle = \langle \bigwedge_{x \in \mathbf{x}} \phi_x(x), \prod_{x \in \mathbf{x}} \omega_x(x) \rangle$ can be rewritten as $\text{vol}(\phi, \omega) = \int \prod_{x \in \mathbf{x}} [\![\phi_x(x)]\!] \omega_x(x) d\mathbf{x}$. This formulation allows the integrations to be *pushed inwards* for any partition of the variables $\mathbf{x}$ into disjoint sets $\mathbf{x}_1$ and $\mathbf{x}_2$: $\int \prod_{x \in \mathbf{x}_1} [\![\phi_x(x)]\!] \omega_x(x) \left[\int \prod_{y \in \mathbf{x}_2} [\![\phi_y(y)]\!] \omega_y(y) d\mathbf{x}_2 \right] d\mathbf{x}_1$.

Factorization is less straight-forward in the case of general WMI (which includes non-axis aligned $\mathcal{LRA}$ atoms). However, many problems, especially those coming from a probabilistic programming context, are partially factorizable [Gehr et al., 2016].

Therefore, we introduce *F-XSDD*, a new solver that exploits factorized solving for WMI problems with multivariate inequalities and weight functions that might not factorize completely. Our method F-XSDD compiles a WMI problem into a set of XSDDs, performs a static circuit analysis and structures WMI as factorized, symbolic integration over the XSDDs.

4.5.1 Factorized Solving

After compiling a λ-SMT solution $\mathcal{W}$ for a WMI problem $\text{WMI}(\phi, w|\mathbf{x}, \mathbf{b})$, every set of tuples $\langle \phi_i, \omega \rangle$ sharing the same world-weight ω is grouped as $\langle \bigvee_i \phi_i, \omega \rangle$. Each disjunction of world-supports $(\bigvee_i \phi_i)$ is represented as an XSDD D[4]. Computing $\text{WMI}(\phi, w|\mathbf{x}, \mathbf{b})$ now consists of summing over all $\langle D, \omega \rangle$ pairs and computing $\sum_{\phi_i} \text{vol}(\phi_i, \omega)$ for every pair. Instead of integrating every $\langle \phi_i, \omega \rangle$ tuple separately, however, we first reconsider jointly integrating ω over $D = \bigvee_i \phi_i$. Using indefinite sums and integrals, and Iverson brackets (like in Equation 4.2), we can rewrite:

$$\sum_{\phi_i} \text{vol}(\phi_i, \omega) = \sum_{\phi_i} \sum_{\mathbf{b}} \int [\![\phi_i(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x} \tag{4.8}$$

$$= \sum_{\mathbf{b}} \int [\![\bigvee_i \phi_i(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x} \quad (\phi_i \text{ are disjoint}) \tag{4.9}$$

$$= \sum_{\mathbf{b}} \int [\![D(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x} \quad (\bigvee_i \phi_i \text{ as XSDD } D) \tag{4.10}$$

By representing the disjunction as an XSDD, we can use its nested representation to push integrations *inside*, i.e., towards the leaves of the SDD. The intuition is that if the expression D contains disjoint sub-expressions over independent sets of variables, those sub-expressions can be integrated separately and the results can be reused if those sub-expressions occur multiple times (we give an example in the supplementary material).

Consider, first, the case of a fully factorizable world-weight $\omega(\mathbf{x}) = \prod_{x \in \mathbf{x}} \omega_x(x)$, an assumption we will revisit later. Using the corresponding XSDD D, we can decompose the integral recursively into smaller sub-problems.

[4] XSDD compilation follows the XADD compilation scheme described in [Kolb et al., 2018] using tuples $\langle D, \omega \rangle$ of SDDs and polynomials to represent case-functions.

Algorithm 4.1 Factorized Integration

world-weight ω
procedure vol(XSDD D, vars $\mathbf{x}$)
  if $\mathbf{x} = \emptyset$ **then**
    return $[\![D]\!]$
  else if D is terminal **then**
    return $\int [\![D]\!] \prod_{x \in \mathbf{x}} \omega_x(x) d\mathbf{x}$
  else if $D = \bigvee_c D_c$ **then**
    return $\sum_c \text{vol}(D_c, \mathbf{x})$
  else if $D = D_1 \wedge D_2$ **then**
    $\mathbf{x}_s = \mathbf{x} \cap \text{vars}(D_1) \cap \text{vars}(D_2)$
    $\mathbf{x}_1^*, \mathbf{x}_2^* = \text{vars}(D_1) \setminus \mathbf{x}_s, \text{vars}(D_2) \setminus \mathbf{x}_s$
    $r_1 = \text{vol}(D_1, \mathbf{x}_1^* \cap \mathbf{x})$
    $r_2 = \text{vol}(D_2, \mathbf{x}_2^* \cap \mathbf{x})$
    return $\int r_1 \cdot r_2 \cdot \prod_{x \in \mathbf{x}_s} \omega_x(x) d\mathbf{x}_s$

Proposition 4.1 (OR Node)**.** *If D corresponds to an OR node, i.e., $D = \bigvee_{D_c} D_c$ where all D_c (again SDDs) are mutually exclusive, we obtain:*

$$\sum_{\mathbf{b}} \int [\![\bigvee_{D_c} D_c(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x} \tag{4.11}$$

$$= \sum_{\mathbf{b}} \int (\sum_{D_c} [\![D_c(\mathbf{x}, \mathbf{b})]\!]) \omega(\mathbf{x}) d\mathbf{x} \tag{4.12}$$

$$= \sum_{D_c} \sum_{\mathbf{b}} \int [\![D_c(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x} \tag{4.13}$$

Proposition 4.2 (AND node)**.** *If $\phi(\mathbf{x})$ corresponds to an AND node, i.e., $D(\mathbf{x}, \mathbf{b}) = D_1(\mathbf{x}_1, \mathbf{b}) \wedge D_2(\mathbf{x}_2, \mathbf{b})$, and we denote $\mathbf{x}_s = \mathbf{x}_1 \cap \mathbf{x}_2$, $\mathbf{x}_1^* = \mathbf{x}_1 \setminus \mathbf{x}_s$, $\mathbf{x}_2^* = \mathbf{x}_2 \setminus \mathbf{x}_s$,*

$\omega_{\mathbf{x}} = \prod_{x \in \mathbf{x}} \omega_x(x)$, $D_1 = D_1(\mathbf{x}_1, \mathbf{b})$, $D_2 = D_2(\mathbf{x}_2, \mathbf{b})$, *we obtain:*

$$\sum_{\mathbf{b}} \int [\![D_1 \wedge D_2]\!] \omega_{\mathbf{x}} d\mathbf{x} = \tag{4.14}$$

$$\sum_{\mathbf{b}} \int [\![D_1]\!][\![D_2]\!] \omega_{\mathbf{x}_1^*} \omega_{\mathbf{x}_2^*} \omega_{\mathbf{x}_s} d\mathbf{x} = \tag{4.15}$$

$$\sum_{\mathbf{b}} \int \Big[\int [\![D_1]\!] \omega_{\mathbf{x}_1^*} d\mathbf{x}_1^* \Big] \Big[\int [\![D_2]\!] \omega_{\mathbf{x}_2^*} d\mathbf{x}_2^* \Big] \omega_{\mathbf{x}_s} d\mathbf{x}_s = \tag{4.16}$$

$$\int \Big[\sum_{\mathbf{b}_1} \int [\![D_1]\!] \omega_{\mathbf{x}_1^*} d\mathbf{x}_1^* \Big] \Big[\sum_{\mathbf{b}_2} \int [\![D_2]\!] \omega_{\mathbf{x}_2^*} d\mathbf{x}_2^* \Big] \omega_{\mathbf{x}_s} d\mathbf{x}_s \tag{4.17}$$

This decomposition allows us to compute weighted model integrals using a recursive symbolic integration algorithm (Algorithm 4.1). Given the world-weight ω, XSDD D and variables to integrate $\mathbf{x}$, the algorithm computes $\sum_{\mathbf{b}} \int [\![D(\mathbf{x}, \mathbf{b})]\!] \omega(\mathbf{x}) d\mathbf{x}$. If the set of variables to integrate over is empty, the algorithm returns $[\![D]\!]$ (the integrations will occur higher in the circuit). If D is a literal (leaf of the XSDD), the integral over $[\![D]\!]$ is computed for the variables $\mathbf{x}$. If D is an OR node, the variables $\mathbf{x}$ are recursively integrated from the child nodes and the results are summed (line 8). Finally, if D is an AND node, the subsets of $\mathbf{x}$ that only occur in one of the child nodes are recursively integrated out and multiplied ($r_p \cdot r_s$), and the remaining subset $\mathbf{x}_s \subseteq \mathbf{x}$ that occur in both children are integrated out from the resulting expression (line 14). The vars values (lines 10 and 11) are precomputed using static circuit analysis.

Let us briefly explain why the outer sum over the Boolean variables $\mathbf{b}$ does not occur in the algorithm. Conjuncts of conjunctions in SDDs do not share Boolean variables and disjuncts of disjunctions in SDDs are pairwise logically inconsistent. We further assume that the algorithm is applied to a *smoothed* circuit, i.e., logically irrelevant Boolean variables are not dropped from the circuit. The sum over the truth values of a Boolean can be pushed into the integration (cf. Equations 4.17) until it reaches a disjunction (cf. Equations 4.13) for which every disjunct is logically consistent only for one of the truth values, which eliminates the sum over that Boolean. For non-smoothed circuits such a disjunction does not necessarily exist.

In order to avoid traversing the SDD multiple times unnecessarily, intermediate results are cached using the tuple ⟨SDD D, variables $\mathbf{x}$⟩ as key. Nodes will only be revisited if different variables need to be integrated out from them. If a node is visited multiple times with different sets of integration variables, the common subset of variables could be detected using circuit analysis and integrated out first. The result can then be cached and reused.

Example 4.2. *Consider the weight function $w = 2xy$ and the following SMT formula:*

$$x{>}0 \wedge x{<}1 \wedge [y{<}1 \vee x{>}y] \wedge y{>}1/2 \tag{4.18}$$

Abstracting the atomic SMT formulas we can compile this formula into the XSDD seen in Figure 4.2.

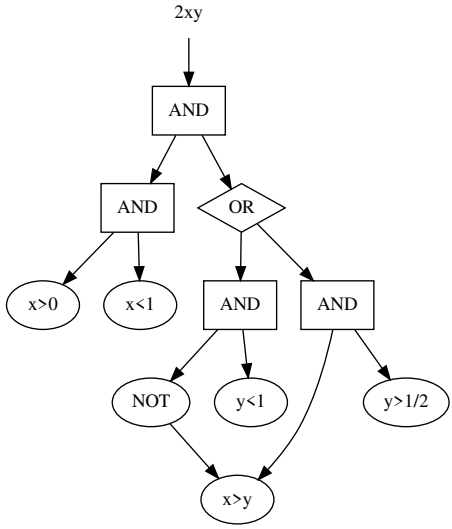


Figure 4.2: The SMT formula in Equation 4.18 for weight $2xy$ from Example 4.2 compiled into an equivalent XSDD.

We are now able to compute the weighted model integral of the problem by evaluating the XSDD and integrating out the resulting symbolic expression:

$$\int \Big([\![x{>}0]\!][\![x{<}1]\!][\![y{<}1]\!][\![x \geq y]\!] + [\![x{>}0]\!][\![x{<}1]\!][\![y{>}1/2]\!][\![x{>}y]\!] \Big) 2xydxdy$$

To solve this integral efficiently we would like to push the integration inside the evaluation of the XSDD and reuse intermediate integration steps. This process of pushing-in the integration over variables is visualized in Figure 4.3.

Evaluating the XSDD depicted in Figure 4.3 results in computing the following integral:

$$2 \int_{(x{>}0)\wedge(x{<}1)} \left(\int_{(y{<}1)\wedge(x \leq y)} ydy + \int_{(y{>}1/2)\wedge(x{>}y)} ydy \right) xdx$$

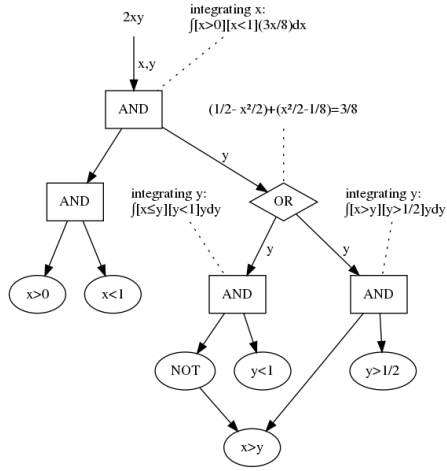


Figure 4.3: We show how integration variables can be pushed inside the XSDD, to integrate subexpressions separately.

We see that the atomic formulas in the SMT($\mathcal{LRA}$) formula in Equation 4.18 become the integration bounds over which to integrate the weight function.

Generic weight functions We describe three mechanisms to relax the condition that world-weight functions have to factorize as expressions over single variables. First, any polynomial can be rewritten into sums of products over powers of single variables. Integration can then be performed separately – each product fully factorizes into expressions over single variables – and the results can be summed to obtain the final result. Second, if the world-weight function factorizes into several products of expressions over non-overlapping sets of variables, we can group these sets of variables together, treating them as inseparable units (and substituting each inseparable unit by a single "set-variable" in the algorithm). If this factorization is not readily apparent, i.e., it is not a product of expressions that do not share real variables, the world-weight could be analyzed upfront to find a suitable decomposition. Third, F-XSDDs, like Symbo, support labeling functions (see Section 4.3.1) which occur frequently in the context of probabilistic programming [Kimmig et al., 2011]. To use a labeling function α, the factorized integration algorithm requires two changes: 1) for any Boolean literal l over a Boolean variable b, the evaluation of $[\![l]\!]$ has to be replaced by the label assigned to that literal $\alpha_b(l)$; and 2) if labels are polynomials, vars(D) has to include the real variables occurring in the labels of Boolean literals in D (to prevent integrating out those real variables too early).

Circuit ordering and redundancy To what extent a WMI problem can be exploited by factorized solving depends on the problem itself, the structure of the XSDD and the ordering of the literals in the XSDD. Currently, compilation procedures for XSDDs do not take into account that some of the literals are abstractions of inequalities, and, therefore, they do not use information about the real variables and their occurrences in inequalities and labels.

4.5.2 Experimental Evaluation

In this section we want to answer three research questions.

Q1 Can the F-XSDD solver exploit factorizability in WMI problems?

Q2 What is the influence of the symbolic integration back-end on F-XSDD solver (F-XSDD(PSI) vs F-XSDD(BR)

Q3 How does F-XSDD perform compared to state-of-the-art solvers?

In our experimental evaluation we compare the state-of-the-art solvers described in Section 4.4 to a variety of XSDD based solvers. F-XSDD(PSI) and F-XSDD(BR) are both implementations of our factorized integration algorithm. F-XSDD(PSI) uses PSI as its symbolic integration back-end, while F-XSDD(BR) relies on XADDs to represent symbolic intermediate results and uses the BR (bound-resolution) algorithm to perform symbolic integration (on the XADDs). Using the BR algorithm within the F-XSDD solving scheme allows for having different variable integration orderings in different subdiagrams and to use labeling functions in combinataion with XADDs. When a problem does not factorize, F-XSDD(BR) reduces to BR with the overhead of performing a static circuit analysis. XSDD(PSI) and XSDD(BR) are implementations of F-XSDD(PSI) and F-XSDD(BR) where we turned off the factorization. XSDD(PSI) is a reimplementation of Symbo [Zuidberg Dos Martires et al., 2019b]. XSDD(Sampling) is functionally equivalent to XSDD(Latte) (cf. Section 4.4, Other Solvers), using a simple rejection-sampling based backend for Monte Carlo integration (10^5 samples per integration).[5]

In the table below we give an overview of WMI solvers used in our empirical analysis. For the PA [Morettin et al., 2017] and the PRAiSE [De Salvo Braz et al., 2016] solvers we used the original implementation, and we reimplemented the BR [Kolb et al., 2018] and the Symbo (listed as XSDD(PSI)) [Zuidberg Dos Martires et al., 2019b] algorithms. We indicate whether the solvers are based on knowledge compilation and whether the integration is performed through symbolic integration. All listed

[5] All experimentally evaluated solvers are part of the pywmi library [Kolb et al., 2019a] (`http://pywmi.org`).

solvers perform exact WMI inference with the exception of XSDD(Sampling), which approximates the integration step with Monte Carlo integration through rejection sampling.

Name	KC	Symbolic	Implementation
PA			Original
PRAiSE		✓	Original
BR	✓	✓	Reimplemented
XSDD(Latte)	✓		New
XSDD(Sampling)	✓		New
XSDD(PSI)	✓	✓	Reimplemented
XSDD(BR)	✓	✓	New
F-XSDD(PSI)	✓	✓	New
F-XSDD(BR)	✓	✓	New

We test the solvers on four WMI problem-templates, whose size can be controlled by a parameter (N). The **click-graph** problem is a probabilistic program and part of the benchmark used to compare Symbo and PSI [Zuidberg Dos Martires et al., 2019b], and is the problem both solvers struggled with most. We encoded the problem as a WMI problem template. We introduce a synthetic problem, **dual-mutex**, that is both structured and factorizable with $\phi = (\bigvee_i (x_{i0} \leq x_{i1})) \wedge \bigwedge_{i,j \neq i} \neg(x_{i0} \leq x_{i1}) \vee \neg(x_{j0} \leq x_{j1})$ and $w = 1$. Additionally, we encoded the highly-structured **mutually-exclusive** and **xor** problems from [Kolb et al., 2018]. Our experimental results are shown in Figure 4.4.

We can answer **Q1** affirmatively, as the performance of both F-XSDD solvers on click-graph and dual-mutex demonstrates that it can succesfully exploit the high degree of factorizability of those problems.

For **Q2**, we can observe that the BR solver and the corresponding XADD structure is required to solve the highly structured mutually-exclusive and xor problems. While both back-ends perform similarly on dual, only F-XSDD(BR) can achieve an exponential-to-linear in reduction time for the click-graph problem by combining factorizable solving with XADDs with BR. We can see the effect of using DAGs to compactly represent intermediate symbolic results and exploiting the overlapping paths using bound resolution. However, the results also clearly show that the BR solver is unable to efficiently solve these problems *without* factorized solving.

With respect to **Q3** we can clearly see that F-XSDD(BR) consistently delivers best-in-class results across these benchmark problems. The F-XSDD solvers outperform the numeric solvers PA and XSDD(Latte) on these structured problems. On the click-graph, mutually-exclusive and xor problems we see that F-XSDD(PSI), like PRAiSE, suffers from its tree-based representation for intermediate symbolic results. The performance of XSDD(Sampling) indicates that the number of tuples in the solutions of the λ-SMT problem of dual-mutex and mutually-exclusive grows polynomially, while for the

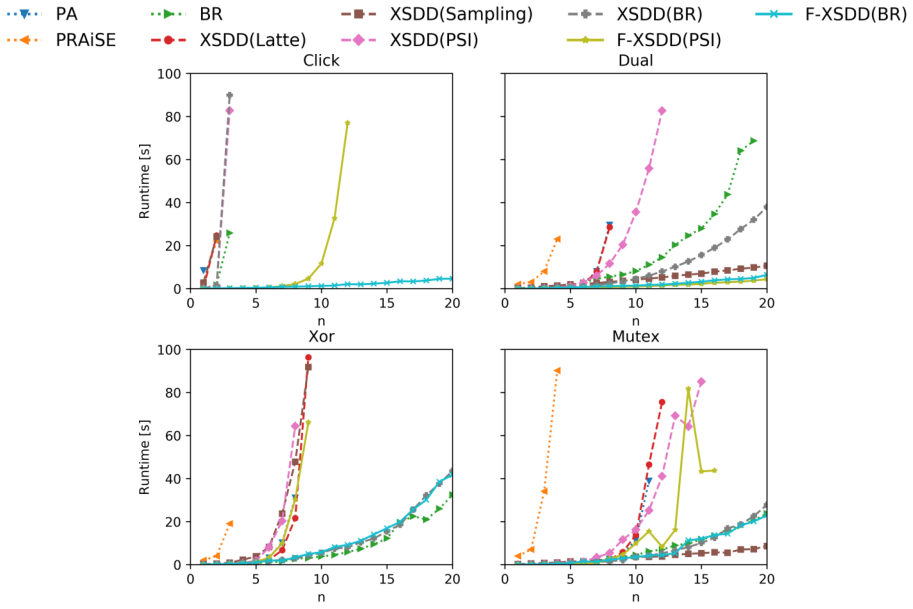


Figure 4.4: On the *click-graph* (top-left) and pairwise-factorizable *dual-mutex* (top-right) problems, F-XSDD outperforms all exact state-of-the-art solvers, while on the highly structured *xor* (bottom-left) and *mutually-exclusive* (bottom-right) problems, it achieves performance on par with BR when using XADDs and bound-resolution as integration back-end (runtimes include compilation).

click-graph and xor problems, they exhibit exponential growth. This demonstrates the need for symbolic integration and the ability to reuse symbolic integration steps in these cases.

4.5.3 Beyond Piecewise-Polynomial WMI

Our implementation of factorized solving focuses on WMI problems with SMT($\mathcal{LRA}$) atoms and piecewise polynomial densities. This puts us on even footing with most of the WMI literature [Belle et al., 2015a; Morettin et al., 2017; Kolb et al., 2018]. However, factorized solving does not require piecewise *polynomial* densities, the expressiveness depends on the back-end that is used for symbolic computations. As a matter of fact, by using PSI as a back-end for symbolic expression manipulation, our factorized solving implementation can already deal with common probability distributions such as Gaussians out of the box.

4.6 Conclusions

In this chapter, we introduced the problem of λ-SMT, which allowed us to dissect in detail different state-of-the-art solvers. Moreover, we introduced F-XSDD, a novel solver that exploits factorizable weight functions through static circuit analysis and that outperforms or is on par with the state-of-the-art.

A promising road for future research would be to realise an XSDD implementation that treats $\mathcal{LRA}$ literals as first-class citizens, either through a top-down knowledge compiler for SMT formulas, combining the strengths of DPLL search and knowledge compilation, or through a bottom-up compiler in the spirit of XADD compilation [Kolb et al., 2018]. Compared to the current XSDD compilation, an *SMT-aware* compiler can look inside the atomic SMT atomic formulas and prune out redundancies, potentially leading to significantly smaller circuits.

Chapter 5

WMI Using Monte Carlo Anti-Differentiation*

Probabilistic inference is a computationally hard task. Two prominent techniques to mitigate this hardness are Monte Carlo estimation and dynamic programming. The former is first and foremost used to cope with intractable probability distributions of continuous random variables, while the latter has been applied to probabilistic inference in domains with discrete random variables. Surprisingly, a serious effort of combining both techniques to perform probabilistic inference in discrete-continuous domains has not been made, yet. We introduce Monte Carlo anti-differentiation (MCAD), an integration method that recursively nests Monte Carlo estimators within a dynamic programming algorithm. MCAD can be thought of as approximate symbolic integration. In this chapter, we develop and experimentally evaluate MCAD in the context of weighted model integration.

5.1 Introduction

Comparing weighted model integration to weighted model counting, we identify one major complication: integrating out continuous variables. Computing integrals is a computationally hard problem and suffers from the curse of dimensionality[1]. Integrating

*This chapter is based on [Zuidberg Dos Martires; Kolb, 2020].

[1]The term *curse of dimensionality* was originally coined in [Bellman, 1957] in the context of dynamic programming but has since also been applied to the problem of integration.

a polynomial over a convex polytope, for example, is #P-hard [Dyer; Frieze, 1988; Koutis, 2003; Koutis, 2003].

Since the inception of WMI, a number of algorithms have emerged that exploit structure present in WMI problems, e.g. [Morettin et al., 2017; Kolb et al., 2018; Kolb et al., 2019b; Zeng; Van den Broeck, 2019]. Exploiting structure, such as determinism and context-specific independence [Chavira; Darwiche, 2008], allows these algorithms to avoid the combinatorial explosion induced by the discrete random variables, when possible. More concretely, the state-of-the-art F-XSDD solver [Kolb et al., 2019b] combines dynamic programming techniques and symbolic integration to exploit the structure present in common probabilistic inference problems.

Symbolic integration is the problem of performing anti-differentiation (finding the indefinite integral, also called the primitive). The F-XSDD solver performs inference by recursively *nesting* symbolic integration steps and caching symbolic sub-results that can be reused. This is conceptually elegant but computationally problematic as these symbolic integration steps are hard to perform (#P-hard in general).

A popular technique to (approximately) circumvent the computational hardness of integration is Monte Carlo estimation. An interesting twist to Monte Carlo estimation is *nested Monte Carlo* (NMC) [Rainforth et al., 2018]. NMC recursively computes approximations of expectations of expectations, i.e. it approximates nested integrations. However, Monte Carlo integration, including NMC, can only perform definite integration and not indefinite integration which is necessary for symbolic integration.

We introduce the concept of Monte Carlo anti-differentiation (MCAD), an extension of NMC that computes the integrals by performing approximate symbolic integration. MCAD allows us deploy Monte Carlo integration within a dynamic programming algorithm.

In this chapter we will use the following definition of the weighted model integral (cf. Equation 4.2 in the previous chapter):

Definition 5.1 (Weighted modeling integration [Kolb et al., 2019b])**.** *Given a set* $\mathbf{b}$ *of* M *Boolean variables,* $\mathbf{x}$ *of* N *real variables, a weight function* $w : \mathbb{B}^M \times \mathbb{R}^N \to \mathbb{R}_{\geq 0}$*, and a support* ϕ*, in the form of an SMT formula, over* $\mathbf{b}$ *and* $\mathbf{x}$*, the weighted model integral is given by:*

$$\mathrm{WMI}(\phi, w|\mathbf{x}, \mathbf{b}) = \sum_{\mathbf{b}} \int [\![\phi(\mathbf{x}, \mathbf{b})]\!] w(\mathbf{x}, \mathbf{b}) d\mathbf{x} \tag{5.1}$$

Furthermore, as we want to study integration over the real domain, we will, for the sake of clarity, ignore Boolean variables in the remainder of this chapter[2]:

$$\text{WMI}(\phi, w|\mathbf{x}) = \int [\![\phi(\mathbf{x})]\!] w(\mathbf{x}) d\mathbf{x} \tag{5.2}$$

In order to apply dynamic programming to solve WMI problems, the weight function has to be separable in some way. We will assume a fully separable weight function:

$$w(\mathbf{x}) = \prod_{k=1}^{T} w_k(x^k)$$

T being the number of real variables.

Following the notation developed in Chapter 4, we denote the compiled representation of $\phi(\mathbf{x})$ by $D(\mathbf{x})$. We write the weighted model integral for a fully factorized weight function now as:

$$\text{WMI}(\phi, w|\mathbf{x}) = \int [\![D(\mathbf{x})]\!] \prod_{k=1}^{T} w_k(x^k) d\mathbf{x} \tag{5.3}$$

This is equivalent to Equation 4.10, apart from the remove dependency on the Boolean variables.

5.2 Problem formulation

The main concept underlying symbolic integration is anti-differentiation of univariate functions.

Definition 5.2. *Given an interval* $[a, b] \subseteq \mathbb{R}$ *and a function* $f : [a, b] \to \mathbb{R}$*, a differentiable function* $F : [a, b] \to \mathbb{R}$ *is called an anti-derivative of* f *if* $\frac{\partial F(x)}{\partial x} = f(x)$*, for all* $x \in [a, b]$*.* $\int f(x)dx$ *denotes the set of anti-derivatives of* f*, which differ by a constant* k*:* $\int f(x)dx + k$*.*

Once an anti-derivative is obtained in closed form, which is trivial for polynomials, we compute the integral $\int_{lb}^{ub} f(x)dx = F(ub) - F(lb)$. Where $lb, ub \in [a, b]$ are the lower and upper bounds respectively. In order to use anti-differentiation for the integration of multivariate functions, we have to repeatedly compute the anti-derivative. The lower and upper bounds depend in this case on the variables that have not yet been

[2] Zeng; Van den Broeck [2019] have shown that summation over Boolean variables can be transformed into integration over real variables. Ignoring Boolean variables comes, hence, with no loss of generality.

integrated out. Due to the dependencies between variables in the bounds of integration, the number of lower and upper bound combinations can explode as variables are successively integrated out. This makes symbolic integration of multivariate functions a computationally hard problem.

To show how symbolic integration is used for WMI, let us rewrite the compiled representation of the support $D(\mathbf{x})$ as a disjunction of conjunctions $E(\mathbf{x}){=}\vee_i E^i(\mathbf{x}){=}\vee_i \wedge_j e^{ij}(\mathbf{x}^{ij})$, where the disjuncts are pairwise logically inconsistent. This is a crucial property as it avoids double counting. Note that rewriting $\phi(\mathbf{x})$ as $E(\mathbf{x})$ can lead to an exponential blow-up of the SMT formula. The $e^{ij}(\mathbf{x}^{ij})$ are SMT($\mathcal{LRA}$) atoms depending on the variables $\mathbf{x}^{ij}{\subseteq}\mathbf{x}$. The $E^i(\mathbf{x})$'s form convex polytopes as they are conjunctions of linear constraints on a convex domain of definition. Let us now calculate the weighted model integral of a single disjunct of $E(\mathbf{x})$:

$$\begin{aligned}\mathrm{WMI}(E^i, w|\mathbf{x}) &= \int [\![E^i(\mathbf{x})]\!] \prod_{k=1}^{T} w_k(x^k) d\mathbf{x} \\ &= \int [\![\wedge_j e^{ij}(\mathbf{x})]\!] \prod_{k=1}^{T} w_k(x^k) d\mathbf{x} \end{aligned} \tag{5.4}$$

Symbolic integration is performed by successively integrating out the T variables in $\mathbf{x}$. The order of integration is determined by the WMI algorithm[3], which uses symbolic integration as a subroutine.

Now we would like to integrate out the variable x^T. Therefore, we need to separate the conditions in E^i that depend on x^T and the ones that do not:

$$[\![E^i(\mathbf{x})]\!] = [\![E^{i,T-1}(x^{1:T-1})]\!][\![E^{i,T}(x^{1:T})]\!] \tag{5.5}$$

$[\![E^{i,T-1}(x^{1:T-1})]\!]$ is the Iverson bracket of the conjunction of SMT($\mathcal{LRA}$) atoms that do not contain the variable x^T and $[\![E^{i,T}(x^{1:T})]\!]$ is the Iverson bracket of the conjunction of atoms that do dependent on x^T. The latter become the bounds of integration for the integration over x^T.

[3] We have studied the problem of ordering in [Derkinderen et al., 2020].

Dropping, for notational clarity, the index on E^i's we write the weighted model integral for a specific E as:

$$\begin{aligned}\mathrm{WMI}(E, w|\mathbf{x}) = \int \Big[[\![E^{T-1}(x^{1:T-1})]\!] \prod_{k=1}^{T-1} w_k(x^k) \Big] \\ \times \underbrace{\Big[\int [\![E^T(x^{1:T})]\!] w_T(x^T) dx^T \Big]}_{=:\gamma_T(x^{1:T-1})} dx^{1:T-1} \qquad (5.6)\end{aligned}$$

$$= \int \Big[[\![E^{T-1}(x^{1:T-1})]\!] \prod_{k=1}^{T-1} w_k(x^k) \Big] \times \gamma_T(x^{1:T-1})\, dx^{1:T-1} \qquad (5.7)$$

We call $\gamma_T(x^{1:T-1})$ the anti-derivative of $[\![E^T(x^{1:T}) w_T(x^T)]\!]$ with respect to dx^T. In a dynamic programming scheme the function γ_T is cached and does not need to be recomputed. Computing the exact form of γ_T is computationally hard.

This brings us to the crux of this chapter. On the one hand, we would like to **exploit the structure present in WMI problems**, for which we need to perform symbolic integration. On the other hand, we want to **circumvent the curse of dimensionality of integration** by using MC techniques. We propose Monte Carlo anti-differentiation, which allows to cache subresults (the γ's) in a dynamic programming scheme, while deploying Monte Carlo methods (approximating the γ's). This leads to a recursive nesting of Monte Carlo estimators, akin to the concept behind NMC [Rainforth et al., 2018].

5.3 Monte Carlo anti-differentiation

In the previous section we stated that the γ_T in Equation 5.7 is the function that we would like to compute, cache, and reuse. Unfortunately, computing γ_T exactly is expensive. In this section we will describe how to approximate, cache, and reuse γ_T.

First, however, let us introduce some new notation to denote groups of variables. Until now we have had $\mathbf{x}$, which denotes an (ordered) set of variables of size T. x^i denotes a variable in that set. We will additionally use $\mathfrak{x}^i$ to denote $\mathfrak{D}$ subsets of $\mathbf{x}$. These subsets, which contain variables from the set $\mathbf{x}$ are exhaustive and pairwise mutually exclusive:

$$\mathbf{x} = \bigcup_{1 \le i \le \mathfrak{D}} \mathfrak{x}_i \qquad (5.8)$$

$$\mathfrak{x}_i \cap \mathfrak{x}_j = \emptyset \qquad i \neq j \qquad (5.9)$$

In other words, we group variables together in $\mathfrak{D}$ different disjoint sets of variables. The notation $\mathfrak{x}^{j:k}$, with $(j \leq k)$, means the following:

$$\mathfrak{x}^{j:k} := \bigcup_{j \leq i \leq k} \mathfrak{x}^i. \tag{5.10}$$

If all $\mathfrak{x}^i$ are singletons we have that $T = \mathfrak{D}$.

We now write Equation 5.7 in terms of sets of variables instead of single variables:

$$\mathrm{WMI}(E, w|\mathbf{x}) = \int \left[[\![E^{\mathfrak{D}-1}(\mathfrak{x}^{1:\mathfrak{D}-1})]\!] \prod_{k=1}^{\mathfrak{D}-1} w_k(\mathfrak{x}^k) \right] \times \gamma_{\mathfrak{D}}(\mathfrak{x}^{1:\mathfrak{D}-1})\, d\mathfrak{x}^{1:\mathfrak{D}-1} \tag{5.11}$$

5.3.1 One Level of Nesting

Let us revisit Equation 5.11. The function $\gamma_{\mathfrak{D}}$ depends here on all the variables in $\mathfrak{x}^{1:\mathfrak{D}-1}$. This dependency came about through $[\![E^{\mathfrak{D}}(\mathfrak{x}^{1:\mathfrak{D}})]\!]$, which is the Iverson bracket of the conjunction of SMT($\mathcal{LRA}$) atoms containing the variables in $\mathfrak{x}^{\mathfrak{D}}$. This means that $[\![E^{\mathfrak{D}}(\mathfrak{x}^{1:\mathfrak{D}})]\!]$ does not necessarily depend on all variables in $\mathfrak{x}^{1:\mathfrak{D}}$ but only on a subset thereof. We denote the subset of variables that co-occur with $\mathfrak{x}^{\mathfrak{D}}$ in SMT($\mathcal{LRA}$) atoms by $\mathfrak{x}_C^{1:\mathfrak{D}}$. This subset also includes all the variables in $\mathfrak{x}^{\mathfrak{D}}$.

As $\gamma_{\mathfrak{D}}$ does only depend on the variables in $\mathfrak{x}_C^{1:\mathfrak{D}-1}$, it is only defined on the domain of definition of the variables $\mathfrak{x}_C^{1:\mathfrak{D}-1} \backslash \mathfrak{x}^{\mathfrak{D}}$ given by the WMI problem.

$$\gamma_{\mathfrak{D}}(\mathfrak{x}^{1:\mathfrak{D}-1}) = [\![bounds(\mathfrak{x}_C^{1:\mathfrak{D}-1})]\!] \gamma_{\mathfrak{D}}(\mathfrak{x}_C^{1:\mathfrak{D}-1}) \tag{5.12}$$

We continue by writing $\gamma_{\mathfrak{D}}$ again in its integral form:

$$[\![bounds(\mathfrak{x}_C^{1:\mathfrak{D}-1})]\!] \gamma_{\mathfrak{D}}(\mathfrak{x}_C^{1:\mathfrak{D}-1}) = [\![bounds(\mathfrak{x}_C^{1:\mathfrak{D}-1})]\!] \int [\![E^{\mathfrak{D}}(\mathfrak{x}_C^{1:\mathfrak{D}})]\!] w_{\mathfrak{D}}(\mathfrak{x}^{\mathfrak{D}}) d\mathfrak{x}^{\mathfrak{D}} \tag{5.13}$$

$$= \int [\![bounds(\mathfrak{x}_C^{1:\mathfrak{D}-1})]\!] [\![E^{\mathfrak{D}}(\mathfrak{x}_C^{1:\mathfrak{D}})]\!] w_{\mathfrak{D}}(\mathfrak{x}^{\mathfrak{D}}) d\mathfrak{x}^{\mathfrak{D}} \tag{5.14}$$

The Iverson brackets in Equation 5.14 form a convex polytope (conjunction of linear inequalities). We call this polytope $K_{\mathfrak{D}}$ and denote its volume by $\mathrm{vol}(K_{\mathfrak{D}})$.

Definition 5.3. *The Monte Carlo anti-derivative $\hat{\gamma}_{\mathfrak{D}}$ of $\gamma_{\mathfrak{D}}$ is given by:*

$$\hat{\gamma}_{\mathfrak{D}}(\mathfrak{x}_C^{1:\mathfrak{D}-1}) = \hat{\rho}(\mathfrak{x}_C^{1:\mathfrak{D}-1} | K_{\mathfrak{D}}, w_{\mathfrak{D}}) \tag{5.15}$$

$\hat{\rho}$ is a function estimator that gives a Monte Carlo estimate, given the polytope $K_{\mathfrak{D}}$ and the weight function $w_{\mathfrak{D}}$. The symbol $\mathfrak{x}_C^{1:\mathfrak{D}}$ denotes the set of variables that are coupled to the variables in $\mathfrak{x}^{\mathfrak{D}}$ through SMT($\mathcal{LRA}$) atoms (this includes the variables in $\mathfrak{x}^{\mathfrak{D}}$).

Example 5.1. *Consider Figure 5.1 on Page 69, particularly the convex polytope given on the left and the weight function $w_2(x_2)$ (depth T=2) depicted next to it. We would like to compute the anti-derivative of the weight function with respect to the variable x_2, which means that we need to integrate out x_2 taking into consideration the bounds imposed on x_2 (blue):*

$$[\![E^2(x_1, x_2)]\!] = [\![1 \leq x_2]\!][\![x_2 \leq 4]\!][\![x_1 \geq x_2]\!]$$

The problem is that these bounds do not induce a convex polytope but only a region in $\mathbb{R}^2$ in which $w_2(x_2)$ is not Lebesgue integrable. The integral is not finite on this $\mathbb{R}^2$ region. To fix this, we take also into consideration the domain of the WMI problem on the x_1 variable $(1 \leq x_1)$ and $(x_1 \leq 4)$ (in dashed red). These five inequalities combined do now induce a convex polytope K_2(shaded in blue). With the convex polytope at hand, we can sample uniformly points from it. We then weight each sample with the weight produced by the weight function $w(x_2) = (x_2 - 2)^2$. The Monte Carlo approximation of the anti-derivative $\hat{\gamma}_2(x_1)$ is then obtained by estimating the density of the weighted samples. We performed the density estimation using a histogram (bottom right).

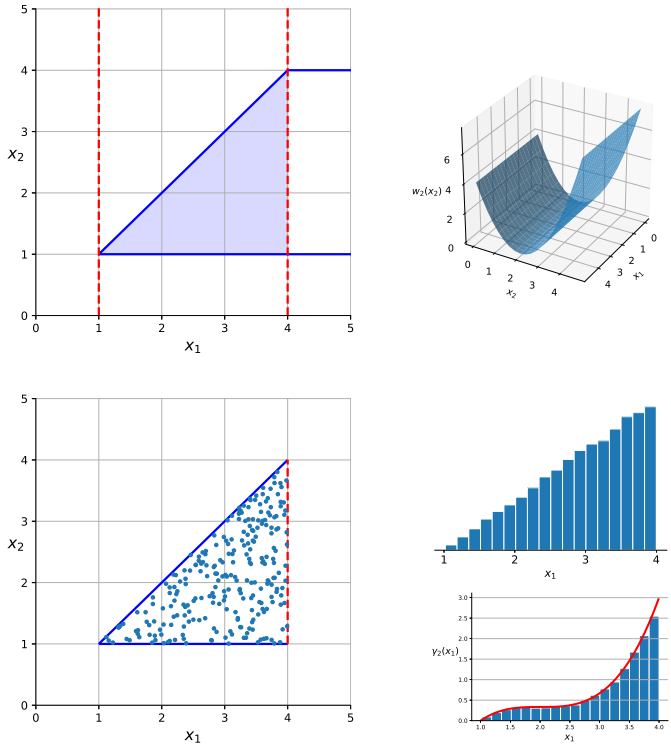


Figure 5.1: On the top left is given a convex polytope (shaded in blue) constrained by the inequalities $(1 \leq x_2)$, $(x_2 \leq 4)$, $(x_1 \geq x_2)$ (blue), and $(1 \leq x_1)$, $(x_1 \leq 4)$ (dotted red). To its right the weight function $w_2(x_2) = (x_2 - 2)^2$ is shown. The plot on the bottom left shows samples drawn uniformly from within the convex polytope K_2 delimited by the constraints $(1 \leq x_2)$, $(x_1 \geq x_2)$ and $(x_1 \leq 4)$. The first histogram on the bottom right shows the projection of the samples onto the x_1-axis and binned into 18 bins of equal width. In the lower right is shown the piecewise constant function that approximates the anti-derivative of x_2 over the weight function $w_2(x_2)$ constrained by the convex polytope. The anti-derivative is denoted by $\gamma_2(x_1)$. Its approximation is obtained by weighting the samples with the weight function $w_2(x_2) = (x_2 - 2)^2$ and estimating the density through a histogram. In other words, the second histogram is the weighted equivalent of the first histogram; using the weight function $w_2(x_2)$. The plot does also show in red the (exact) symbolic integral: $\int [\![1 \leq x_2]\!][\![x_2 \leq x_1]\!](x_2 - 2)^2 dx_2 = \int_1^{x_1} (x_2 - 2)^2 dx_2 = {}^1/_3 x_1^3 - 2x_1^2 + 4x_1 - {}^7/_3$.

Proposition 5.1. *On level of MCAD produces biased estimates of weighted model integrals if the estimate $\hat{\gamma}$ is biased.*

Proof. Let us assume we want to calculate the following weighted model integral:

$$\begin{aligned}
&\mathrm{WMI}(\phi, w_1 \times w_2 | \mathfrak{x}^1, \mathfrak{x}^2) \\
&= \int [\![E^1(\mathfrak{x}^1)]\!][\![E^2(\mathfrak{x}_C^{1:2})]\!] w_1(\mathfrak{x}^1) w_2(\mathfrak{x}^2) d\mathfrak{x}^2 d\mathfrak{x}^1 \\
&= \int [\![E^1(\mathfrak{x}^1)]\!] w_1(\mathfrak{x}^1) \Big(\int [\![E^2(\mathfrak{x}_C^{1:2})]\!] w_2(\mathfrak{x}^2) d\mathfrak{x}^2 \Big) d\mathfrak{x}^1 \\
&= \int [\![E^1(\mathfrak{x}^1)]\!] w_1(\mathfrak{x}^1) \gamma(\mathfrak{x}^1) d\mathfrak{x}^1 \qquad (5.16)
\end{aligned}$$

As $\hat{\gamma}$ is a random variable. we are interested in the expected bias:

$$\begin{aligned}
&\mathbb{E}\Big[\int [\![E^1(\mathfrak{x}^1)]\!] w_1(\mathfrak{x}^1) \hat{\gamma}(\mathfrak{x}^1) d\mathfrak{x}^1 - \int [\![E^1(\mathfrak{x}^1)]\!] w_1(\mathfrak{x}^1) \gamma(\mathfrak{x}^1) d\mathfrak{x}^1 \Big] \\
&= \int [\![E^1(\mathfrak{x}^1)]\!] w_1(\mathfrak{x}^1) \, \mathbb{E}\Big[\hat{\gamma}(\mathfrak{x}^2) \Big] d\mathfrak{x}^1 - \int [\![E^1(\mathfrak{x}^1)]\!] w_1(\mathfrak{x}^1) \gamma(\mathfrak{x}^1) d\mathfrak{x}^1 \\
&= \int [\![E^1(\mathfrak{x}^1)]\!] w_1(\mathfrak{x}^1) \Big(\gamma(\mathfrak{x}^1) + \beta(\mathfrak{x}^1) \Big) d\mathfrak{x}^1 - \int [\![E^1(\mathfrak{x}^1)]\!] w_1(\mathfrak{x}^1) \gamma(\mathfrak{x}^1) d\mathfrak{x}^1 \\
&= \int [\![E^1(\mathfrak{x}^1)]\!] w_1(\mathfrak{x}^1) \beta(\mathfrak{x}^1) d\mathfrak{x}^1 \qquad (5.17)
\end{aligned}$$

We see that already one level of MCAD nesting for the estimation of the weighted model integral is biased if the expectation of the bias β on the estimate is non-zero. □

As shown in the proof of Proposition 5.1, MCAD produces inevitably bias estimates. In future work we will have to provide bounds on this bias.

5.3.2 Repeated Nesting of MCAD

Until now we considered only one level of nesting, i.e. we only had the $\gamma_{\mathfrak{D}}$ that we estimated. We are now going too formalize repeated nesting of Monte Carlo estimators. Consider again Equation 5.7, where we have one level of nesting. Performing this

nesting repeatedly can be described via the following two equations:

$$\gamma_{\mathfrak{D}}(\mathfrak{x}^{1:\mathfrak{D}-1}) = \int [\![E_{\mathfrak{D}}(\mathfrak{x}_C^{1:\mathfrak{D}})]\!] w_{\mathfrak{D}}(\mathfrak{x}^{\mathfrak{D}}) d\mathfrak{x}^{\mathfrak{D}} \tag{5.18}$$

$$\gamma_k(\mathfrak{x}^{1:k-1}) = \int [\![E_k(\mathfrak{x}_C^{1:k})]\!] w_k(\mathfrak{x}^k) \gamma_{k+1}(\mathfrak{x}^{1:k}) d\mathfrak{x}^k \qquad 1 \leq k < \mathfrak{D} \tag{5.19}$$

Where $\mathfrak{x}_C^{1:k}$ denotes the set of variables that are coupled through SMT($\mathcal{LRA}$) atoms with $\mathfrak{x}^k$.

Definition 5.4. *We call a repeated nesting scheme* **layered** *if the following equations hold by definition:*

$$\gamma_{\mathfrak{D}}(\mathfrak{x}^{1:\mathfrak{D}-1}) \overset{!}{=} \gamma_{\mathfrak{D}}(\mathfrak{x}^{\mathfrak{D}-1}) = \int [\![E_{\mathfrak{D}}(\mathfrak{x}_C^{\mathfrak{D}-1:\mathfrak{D}})]\!] w_{\mathfrak{D}}(\mathfrak{x}^{\mathfrak{D}}) d\mathfrak{x}^{\mathfrak{D}} \tag{5.20}$$

$$\gamma_k(\mathfrak{x}^{1:k-1}) \overset{!}{=} \gamma_k(\mathfrak{x}^{k-1}) = \int [\![E_k(\mathfrak{x}_C^{k-1:k})]\!] w_k(\mathfrak{x}^k) \gamma_{k+1}(\mathfrak{x}^k) d\mathfrak{x}^k \qquad 1 \leq k < \mathfrak{D} \tag{5.21}$$

This means that γ_k at layer k depends only on the variables in $\mathfrak{x}^{k-1}$ from the previous layer.

Definition 5.5. *We call a repeated nesting scheme* **factorized** *if the following equations hold by definition:*

$$\gamma_{\mathfrak{D}}(\mathfrak{x}^{1:\mathfrak{D}-1}) \overset{!}{=} \gamma_{\mathfrak{D}} = \int [\![E_{\mathfrak{D}}(\mathfrak{x}^{\mathfrak{D}})]\!] w_{\mathfrak{D}}(\mathfrak{x}^{\mathfrak{D}}) d\mathfrak{x}^{\mathfrak{D}} \tag{5.22}$$

$$\gamma_k(\mathfrak{x}^{1:k-1}) \overset{!}{=} \gamma_k = \int [\![E_k(\mathfrak{x}^k]\!] w_k(\mathfrak{x}^k) \gamma_{k+1}(\mathfrak{x}^k) d\mathfrak{x}^k \qquad 1 \leq k < \mathfrak{D} \tag{5.23}$$

This means that γ_k at layer k does not depend on any variables and is constant.

Note that γ_1=WMI($E, w|x$) and recall that when solving WMI problems via dynamic programming it is exactly these γ's that are being cached. Since computing them exactly is computationally expensive, we replace them by function estimates.

$$\hat{\gamma}_{\mathfrak{D}}(\mathfrak{x}^{1:\mathfrak{D}-1}) \coloneqq \hat{\rho}(\mathfrak{x}^{1:\mathfrak{D}-1} | w_{\mathfrak{D}}, K_{\mathfrak{D}}) \tag{5.24}$$

$$\hat{\gamma}_k(\mathfrak{x}^{1:k-1}) \coloneqq \hat{\rho}(\mathfrak{x}^{1:k-1} | w_k \times \hat{\gamma}_{k+1}, K_k) \qquad 1 \leq k < \mathfrak{D} \tag{5.25}$$

5.3.3 Histograms as density estimators

The perhaps simplest approach to estimate a function is by means of (multidimensional) histograms, i.e. piecewise constant functions. If we assume the domain of γ_k to be the

hyperrectangle HR_k, we can partition the domain into a predefined number of bins M_k of volume v_k (v_k depends on M_k and the volume of HR_k). Let us also assume that we have N_k samples drawn uniformly from the polytope K_k. The estimate of γ_k in the i-th bin denoted by b_k^i ($1 \leq i \leq M_k$) is obtained with:

$$\begin{aligned}\hat{\gamma}_k(\mathfrak{x}^{1:k-1} \in b_k^i) &= \hat{\rho}(\mathfrak{x}^{1:k-1} \in b_k^i | w_k \times \hat{\gamma}_{k+1}, K_k) \\ &= \frac{\mathrm{vol}(K_k)}{v_k N_k} \sum_j^{N_k} \left([\![\mathfrak{x}_j^{1:k-1} \in b_k^i]\!] w_k(\mathfrak{x}_j^k) \hat{\gamma}_{k+1}(\mathfrak{x}_j^{1:k}) \right) \end{aligned} \tag{5.26}$$

$\mathfrak{x}_j^k$ denotes the j-th sampled values for the variables in $\mathfrak{x}^k$. Similarly for $\mathfrak{x}^{1:k-1}$. Note that the samples for $\mathfrak{x}_j^k$ and $\mathfrak{x}^{1:k-1}$ are drawn jointly as the polytope K_k is dependent on those sets of variables. Equation 5.26 simple tells us that we esimate γ_k in bin b_k^i by taking a weighted average over the samples that fall into bin b_k^i. Computing this weighted average for every bin in HR_k gives the estimate of $\hat{\gamma}_k$.

Even though using histograms as density estimators is straightforward, they are not well suited for estimating high dimensional data. If we want to preserve the bin resolution, the number of bins we need to partition our space into grows exponentially with the number variables present in the set $\mathfrak{x}_C$. A possible solution would be to represent the estimate by a lower dimensional representation instead of histograms, such as kernel density methods [Rosenblatt, 1956; Parzen, 1962], density estimation trees [Ram; Gray, 2011], or mixed sum-product-networks [Molina et al., 2018]. We leave this issue as a point for future research.

5.3.4 MCAD and Weighted Model Integration

In Subsection 5.3.2 we introduced nested MCAD under the assumption that the integrand decomposed into a product. However, when performing WMI (and for that matter WMC) the integrand decomposes into a sum-product. We incorporate this decomposition into MCAD by adapting the density estimate of the γ's at each level of integration in the following manner:

$$\hat{\gamma}_k(\mathfrak{x}^{1:k-1}) = \hat{\rho}(\mathfrak{x}^{1:k-1} | w_k \textstyle\sum_j \hat{\gamma}_{j,k+1}, [\![E_k]\!]) \qquad 1 \leq k < \mathfrak{D} \tag{5.27}$$

At each level we have now a sum over multiple γ's from the previous layer.

5.3.5 Implementation

As mentioned in the introduction, MCAD can be viewed as a Monte Carlo version of symbolic integration – variables are integrated out sequentially. Probabilistic inference algorithms using symbolic integration as underlying routine, such as the F-XSDD algorithm[4], naturally lend themselves to be equipped with MCAD. We denote the resulting algorithm by F-XSDD(MCAD), a probabilistic inference algorithm that combines dynamic programming and Monte Carlo integration.

An important issue when implementing MCAD is to avoid high rejection rates when sampling from convex polytopes. This is especially true when sampling from high-dimensional polytopes. We tackled this problem by leveraging the VOLESTI library[5]. VOLESTI is a highly optimized C++ library that implements different sampling strategies [EMIRIS; FISIKOPOULOS, 2014; COUSINS; VEMPALA, 2016; EMIRIS; FISIKOPOULOS, 2018] to efficiently and directly sample points uniformly from high dimensional convex polytopes while avoiding any sample rejection.

A further complication of MCAD, which we have not yet addressed, is the computational hardness of computing the volume of rational convex polytopes, which is strongly #P-hard [DYER; FRIEZE, 1988; KOUTIS, 2003; KOUTIS, 2003]. For low dimensional polytopes (<3 dimensions) the volume can be computed using symbolic integration (with PSI [GEHR et al., 2016]). For intermediate dimensional polytopes ($\lessapprox$10 dimensions) simplex decomposition can be used (e.g. LATTE INTEGRALE [DE LOERA et al., 2013a]). For even higher dimensional polytopes, only approximate algorithms, such as the randomized polynomial volume estimation algorithm available in the VOLESTI library constitute a practical choice. The volume approxation algorithm of VOLESTI approximates the volume up to a user-defined threshold in polynomial time by recursively constructing co-centric balls of diminishing radii [DYER et al., 1991]. We implemented these three options and denote the specific algorithms by: F-XSDD(MCAD(PSI)), F-XSDD(MCAD(LATTE)) and F-XSDD(MCAD(VOLESTI)), respectively.

5.4 Experimental Evaluation

In the experimental evaluation we study the performance and accuracy of MCAD based solvers on WMI problems. Firstly, we study MCAD on a set of standard benchmark problems already present in the WMI literature. Secondly, we modify two existing benchmarks problems to further demonstrate the strengths of MCAD based WMI

[4] Available in the PyWMI toolbox: https://github.com/weighted-model-integration/pywmi.
[5] https://github.com/GeomScale/volume_approximation

solvers. The experiments presented in this section were performed on a Intel(R) i7 CPU 2.60GHz machine with 16GB memory, running Linux OS.

5.4.1 Highly Structured Problems

How does an MCAD based solver fare against a symbolic integration based solver in terms of run time and accuracy of the MC approximation?

In order to answer this question we study F-XSDD(MCAD(PSI)) on four benchmarks problems (Click(N), Dual(N), Xor(N), Mutex(N)), where N denotes the variable problem size. The former two were introduced in [KOLB et al., 2019b], and the latter two in [KOLB et al., 2018].

We compare F-XSDD(MCAD(PSI)) to F-XSDD(BR) [KOLB et al., 2019b], the PA solver [MORETTIN et al., 2017], and the simple rejection sampling algorithm included in the PyWMI toolbox. F-XSDD(BR) is the state-of-the-art algorithm when it comes to exploiting structure. It outperforms or is on par with other solvers on these four benchmarks. PA is another state-of-the art solver, which does, however, perform poorly on highly structured problems – we include it for completeness. Both of these algorithms perform exact WMI. The rejection sampler simply samples uniformly from the domain of definition of the problem, which is a hypercube, and tests whether a sample satisfies all the constraints imposed by the problem. If the sample is accepted it is weighted with the value obtained from evaluating the integrand at the sampled point.

Figure 5.2 shows the experimental comparison of these four solvers. Plotted are the run times in dependency of the problem size and for the approximate solver we did also plot the relative root mean squared error (RRMSE). For F-XSDD(MCAD(PSI)) we used 5×10^4 samples per MCAD step. To determine the number of samples for the rejection algorithm we determined the number of MCAD steps performed by F-XSDD(MCAD(PSI)) and multiplied 5×10^4 with this number. RRMSEs were obtained by using five runs and comparing to the exact result obtained with the F-XSDD(BR) solver. The standard deviation on the run times are not shown as they are negligible.

The plots in Figure 5.2 show that F-XSDD(MCAD(PSI)) beats the exact WMI solvers when comparing run times and with only little damage to the accuracy (quantified by the RRMSE). This is in contrast to the rejection sampler, which suffers from increasing RRMSEs when going to larger problems. The PA solver, prematurely times out on all benchmarks (only solving the first instance for the Click(N) benchmark), which was to be expected as the PA solver does not target highly structured problems.

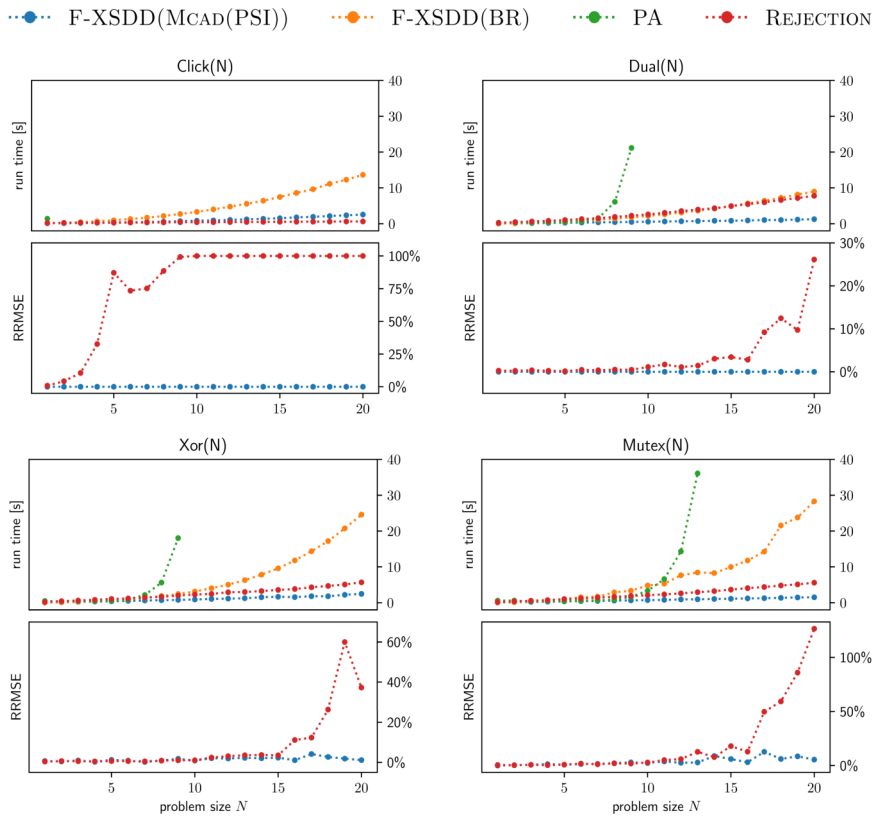


Figure 5.2: Comparison of the run time of F-XSDD(MCAD(PSI)) to the exact state-of-the-art WMI solver F-XSDD(BR) and PA on four benchmark problems in the WMI literature. Additionally we compare the accuracy of F-XSDD(MCAD(PSI)) to the one of a rejection sampling algorithm.

5.4.2 Highly Structured Problems with More Challenging Integration

The benchmarks used so far to assess the efficacy of WMI solvers that exploit structure are rather simple from an integration point of view: the weight over which the integration is performed is usually 1 and the dimension of the integration domain is at most 2. This leads us to our second experimental question.

How does MCAD fare against symbolic integration in problems that exhibit both a rich structure and more challenging integration steps?

To answer this questions we first introduce a variation of the Xor(N) benchmark. Instead of having a constant weight function of 1, we use now a weight function $w = \prod_i f(x_i)$ where the $f(x_i)$ are of polynomial form and the x_i are the continuous variables present in the benchmark. We denote this variation of the Xor(N) benchmark by Xor($f(x)$,N). Secondly we introduce a new benchmark that we dub M-ual($f(x)$, N, M), a variation of the Dual(N) benchmark, which is specified as follows:

$$\phi = \bigvee_{i=1}^{N} \left[\sum_{k=1}^{M} (x_{ik} \leq 0) \right] \wedge \bigwedge_{i=1}^{N} \bigwedge_{j=i+1}^{N} \left[\neg \left(\sum_{k=1}^{M} x_{ik} \leq 0 \right) \vee \neg \left(\sum_{k=1}^{M} x_{jk} \leq 0 \right) \right]$$

with domain $\bigwedge_{k=1}^{M} \bigwedge_{i=1}^{N} (-1 \leq x_{ik} \leq 1)$ and $w = \prod_{i=1}^{N} \prod_{k=1,k\neq i}^{M} f(x_{ik})$. $f(x_{ij})$ being a polynomial in x_{ij} (e.g. x_{ij}^2). It is easy to see that by increasing N and M simultaneously a high-dimensional problem is created. For instance, the problem M-ual($f(x)$, 10, 10) is 100-dimensional. Note that similarly to the Dual(N) problem the M-ual problem naturally decomposes into M-dimensional independent integrals with no shared constraints on the variables, i.e. the problem allows for a *factorized* nesting scheme (cf. Definition 5.5). This means that for the M-ual problems the error on the MC approximation of the weighted model integral stems entirely from the MC integration and not from the density estimation.

In Figure 5.3 we compare again F-XSDD(MCAD(PSI)) to F-XSDD(BR), PA and a simple rejection sampling algorithm (under similar condition as in Figure 5.2). Additionally we included the F-XSDD(MCAD(LATTE)) and the F-XSDD(MCAD(VOLESTI)) solvers in the experimental comparison.

We observe that in the first two experiments in Figure 5.3, F-XSDD(MCAD(PSI)) is again the fastest algorithm without significant hits on the accuracy and the rejection algorithm does again perform poorly when going to higher dimensions. Furthermore, we observe that the approximation of the volume computation with F-XSDD(MCAD(VOLESTI)) introduces extra noise in the weighted model integral. However, when going to high dimensional spaces (the third experiment) we see that F-XSDD(MCAD(VOLESTI)) is the only algorithm that can cope with the situation. Remember that computing the volume of a convex polytope is itself #P-hard. On this third comparison, the PA solver did not solve any of the problems within the time-out (set at 200s) and the rejection algorithm started producing negative weighted model integrals, which is impossible. This lead us to leave out the rejection algorithm from the comparison. Note that for the third experiment we plotted the relative standard deviation (RSTD) in the lower panel, as the exact result is not available for a comparison.

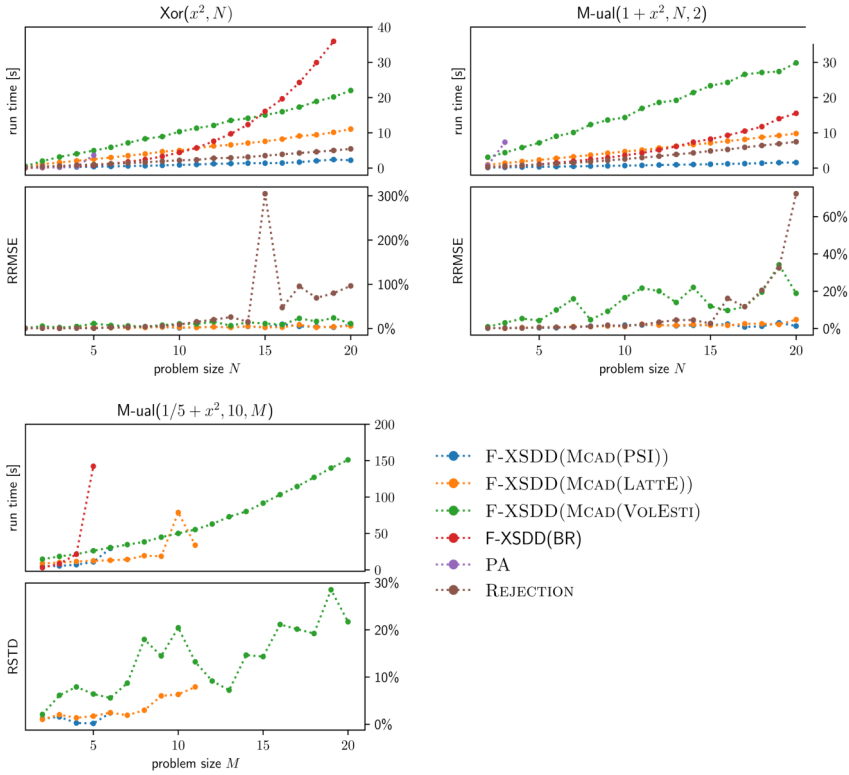


Figure 5.3: Run time comparison of MCAD based solvers to the state-of-the-art WMI solvers on modified benchmarks problems already present in the literature. Additionally, we study the accuracy of the MCAD based algorithms compares to a simple rejections sampling algorithm.

5.5 Conclusions

We developed Monte Carlo anti-differentiation, a method that combines two powerful methods in computer science: Monte Carlo integration and dynamic programming. We established an interesting link between weighted model integration and nested Monte Carlo estimation [RAINFORTH et al., 2018]. Furthermore, we provide an implementation of MCAD, dubbed F-XSDD(MCAD), based on state-of-the-art sampling techniques in linearly constrained spaces, which enriches the F-XSDD family of algorithms with an approximate solver. F-XSDD(MCAD) is the first inference algorithm for WMI that performs Monte Carlo integration while exploiting structure present in WMI problems,

and avoiding prohibitively high sample rejection rates. Our empirical evaluation shows that F-XSDD(MCAD) is able to deliver fast yet reliable approximate inference on a set of representative WMI benchmark problems. In order to increase the practicality of MCAD it might be of interest to investigate density estimators other than histograms, such as density estimation trees, which have already been used in the context of WMI[MORETTIN et al., 2020].

A major shortcoming of MCAD is its unavoidable biasedness and especially the lack of bounds on the bias. Two research directions ahead would be intersting. On the one hand the theoretical derivation of bounds on the bias and on the other hand, the study of unbiased Monte Carlo approximation schemes. For the former we stipulate that bounds can be derived in the spirit of the derivation of bounds given in [RAINFORTH et al., 2018]. For the latter it might be interesting to investigate Gibbs sampling techniques that have already been investigated for discrete-continuous probabilistic graphical models [AFSHAR et al., 2015; AFSHAR et al., 2016]. The proposed Gibbs samplers would have to be adapted to the WMI context: in AFSHAR et al. [2015] and AFSHAR et al. [2016] the conditioning set consists of (conjunctions of) equality constraints only, whereas for WMI and-or structures of inequality constraints are common. It is interesting to note that in the numerical integration literature Gibbs sampling for convex bodies, for example in the VOLESTI library, is referred to as *coordinate hit-and-run* sampling.

A further advantage of Gibbs sampling (beside its unbiasedness) is that it allows us to draw samples directly from (constrained) probability distributions instead of first uniformly drawing samples and then weighting them accordingly. In this context it is also interesting to investigate other techniques that exhibit this advantage [LOVÁSZ; VEMPALA, 2006; AFSHAR; DOMKE, 2015].

Finally, MCAD, is as of now formulated in a WMI setting. It would be intriguing to place MCAD in a broader dynamic programming context. This would also open up the possibility to a wider domain of application for MCAD.

Conclusions

In this part we tackled the following research question:

RQ1: Can we adopt inference algorithms from the purely discrete domain or the purely continuous domain to develop novel WMI solvers?

We started by investigating weighted model integration in combination with knowledge compilation techniques. In Chapter 3 we have shown how standard knowledge compilation can be applied to the task of weighting model integration by leveraging algebraic model counting and thereby presenting a unified formalism for weighted model integration and knowledge compilation. At the same time we introduced an exact and an approximate solver based on this idea and demonstrated their effectiveness.

In Chapter 4 we built on the ideas developed in the Chapter 3 and further formalized the theory underpinning weighted model integration combined with knowledge compilation. This resulted in tge introduction of λ-SMT satisfiability problem, which also allowed us to dissect in detail different state-of-the-art solvers.

Furthermore, we introduced F-XSDD, a novel solver that exploits factorizable weight functions through static circuit analysis and that outperforms or is on par with the state-of-the-art. A promising road for future research would be to realise an XSDD implementation that treats $\mathcal{LRA}$ literals as first-class citizens, e.g. through a top-down knowledge compiler for SMT formulas, combining the strengths of DPLL search and knowledge compilation. Specifically, pruning impossible sub-diagrams (e.g., through top-down compilation and the use of SMT solvers) and clustering inequalities with common variables to enable more efficient symbolic integration. For symbolic integration engines, it is, moreover, essential to determine an ordering of the SMT literals that compressses not only λ-SMT solutions but does equally minimize integration time. We have tackled this problem in a recent publication [Derkinderen et al., 2020].

In Chapter 5 we developed Monte Carlo anti-differentiation, a method that combines

two powerful methods of computer science: Monte Carlo integration and dynamic programming, which we formulated as nested Monte Carlo estimation [Rainforth et al., 2018]. Furthermore, we provide an implementation of MCAD, dubbed F-XSDD(Mcad), based on state-of-the-art sampling techniques in linearly constrained spaces, which enriches the F-XSDD family of algorithms with an approximate solver. F-XSDD(Mcad) is the first inference algorithm for WMI that performs Monte Carlo integration while exploiting structure present in WMI problems, and avoiding prohibitively high sample rejection rates. Our empirical evaluation shows that F-XSDD(Mcad) is able to deliver fast yet reliable approximate inference on a set of representative WMI benchmark problems.

The MCAD paper [Zuidberg Dos Martires; Kolb, 2020], on which Chapter 5 is based, constitutes a first step towards deploying approximate Monte Carlo integration techniques for probabilistic inference in high dimensional hybrid domains. We hope it sparks further research at the intersection of WMI and Monte Carlo methods.

While we have limited ourselves to satisfiability modulo theories over (linear) real arithmetics, there do exist WMI solvers that tackle linear integer arithmetics [Kolb et al., 2018]. Here it might be of interest to investigate links to other existing solver that do not lie directly in the domain of weighted model integration [Ma et al., 2009; Zhou et al., 2015; Gao et al., 2018].

A promising avenue of further research seems to be the exploration of the space of WMI problems and imposing restrictions on it, resulting in a subclass of WMI problems. Much in the direction of the efforts undertaken in [Zeng; Van den Broeck, 2019; Zeng et al., 2020]. Formally identifying classes of WMI problems and which solvers solve them efficiently (and possibly automating the detection of the class) would allow WMI to gain more widespread usage.

While, the theoretical formulation of WMI is elegant and solving WMI problems efficiently touches on a lot of different topics, which are usually not considered together, such as dynamic programming, Boolean satisfiability, Monte Carlo estimation, one major shortcoming of WMI remains. There are no real-world applications! This opens up an interesting avenue for future research: finding domains that can be modeled using SMT formulas and which necessitate a probabilistic approach. Finding such applications will also be beneficial to further improve on state-of-the-art WMI solvers as it would lead to a problem driven development of such solvers.

Probabilistic Logic Programming

Introduction

In the previous part we have seen how to model discrete-continuous probabilistic problems using the language of satisfiability modulo theories. In order to perform inference in these domains, we presented a range of exact and approximate inference algorithms. While weighted SMT formulas are a sound way of expressing probability distributions in discrete-continuous domains, they are rather cumbersome from a user's perspective. In this second part of the thesis, we present **DC-ProbLog**, a high-level probabilistic logic programming tailored towards discrete-continuous domains, which allows users to concisely encode probability distributions. We will answer our second research question.

RQ2: Can we develop a high-level probabilistic logic programming language for which inference reduces to weighted model integration?

DC-ProbLog can be regarded as an extension from two different points of view. On the one hand, it extends weighted SMT formulas to a universal probabilistic programming language, where inference reduces to weighted model integration. On the other hand, it adds continuous random variables to ProbLog, a probabilistic logic programming language over the discrete domain.

A major limitation of ProbLog becomes immediately apparent in the introductory Example 1.1: we need to model the *temperature* as a discrete (Boolean) random variable rather than a continuous probability density.

Poole has characterized probabilistic programming as *independent choices plus deterministic systems* [Poole, 2010]. In the case of probabilistic logic programming with discrete random variables the deterministic system is given by a logic program. DC-ProbLog builds on this idea but relaxes the independence assumption by explicitly distinguishing between two types of dependencies 1) those expressed by the logic program (the deterministic system) and 2) those expressed by using random variables as parameters of distributions for other random variables (the independent random choices). This explicit distinction is not made by any other probabilistic (logic)

programming language including discrete and continuous random variables. Such a programming language is also called *hybrid*.

When writing (hybrid) probabilistic programs, users have a range of options, from encoding entire probability distributions defined by the probabilistic program into a single choice random variable (no deterministic system), all the way to a set of individual independent choice random variables with minimal use of random variables as parameters tied together by a deterministic system. As already discussed by Poole [2010], not all options work equally well with all inference algorithms. DC-ProbLog provides these options within the same semantic framework, leaving it to system developers to focus the syntax towards a subset of options and/or to exploit the different options when integrating different inference approaches into their system.

Developing the DC-ProbLog language, leads to the following specific contributions:

1. We introduce a **type system** for DC-ProbLog, which allows us to introduce a **neat and clean syntax** to extend ProbLog with function symbols for continuous random variables.
2. We sketch a purely **declarative semantics for DC-ProbLog** based on Sato's distribution semantics that allows us to disentangle independent choices and the discrete system underlying a probabilistic program.
3. We **reduce inference** in DC-ProbLog to **weighted model integration** in the algebraic model counting setting [Kimmig et al., 2011; Kimmig et al., 2017], analogous to reducing inference in ProbLog to weighted model counting [Chavira; Darwiche, 2008].
4. We present an **implementation of DC-ProbLog**, which reduces naturally to ProbLog in the absence of function symbols.

DC-ProbLog is heavily influenced by two predecessor languages: 1) ProbLog language and 2) Distributional Clauses [Gutmann et al., 2011; Nitti et al., 2016a]. The latter is a probabilistic logic programming languages that allows for continuous random variables. However, its semantics differ from the semantics of ProbLog as it does not reduce to ProbLog in the absence of continuous random variables.

The work presented on DC-ProbLog is based on a workshop paper in collaboration with Anton Dries and Luc De Raedt and a forthcoming paper in collaboration with Angelika Kimmig and Luc De Raedt.[6]

[6] I have been leading the development of the syntax and the type system, as well as the inference algorithm (reduction to WMI). Furthermore I have implemented completely independently the DC-ProbLog language. Angelika Kimmig and I have been developing the semantics.

Zuidberg Dos Martires, Pedro; Dries, Anton; De Raedt, Luc [2018]. Knowledge Compilation with Continuous Random Variables and its Application in Hybrid Probabilistic Logic Programming. In: *Eigth International Workshop on Statistical Relational AI @ IJCAI*.

Zuidberg Dos Martires, Pedro; Kimmig, Angelika; De Raedt, Luc [2020a]. Extending ProbLog with Random Function Symbols. In: *(in preparation)*.

The reader will spot passages where the exposition might be brief and informal. It is also likely that the text will considerably change for a version published in a peer reviewed paper.

Chapter 6

DC-ProbLog*

6.1 Syntax and Type System

DC-ProbLog is a probabilistic logic programming language derived from the probabilistic logic programming language ProbLog. As such, a lot of the syntax and general structure of DC-ProbLog programs are inherited from ProbLog. A major syntactic difference between ProbLog and DC-ProbLog is the presence of *distributional facts* in the latter. Distributional facts allow a user to express random variables that are distributed according to a certain probability distribution specified by a random function symbol.

Example 6.1. *In this example we model the temperature as a continuous random variable distributed according to a normal distribution with mean* 20 *and standard deviation* 5 *(Line 3).*

```
1  machine(1). machine(2).
2
3  temperature ~ normal(20,5).
4  0.99::cooling(1).
5  0.95::cooling(2).
6
7  works(N):- machine(N), cooling(N).
8  works(N):- machine(N), temperature<25.0.
9
```

*This chapter is based on the following two papers [Zuidberg Dos Martires et al., 2018; Zuidberg Dos Martires et al., 2020a].

```
evidence(works(2)).
query(works(1)).
```

Distributional facts are terms where the predicate ~/2 is written in infix notation. The first argument is a named random variable and the second describes the distribution according to which the random variable is distributed. In the case of Example 6.1 the function symbol is `normal/2`.

Distributional facts are not limited to describing continuous random variables but can, for example, also model discrete probability distributions, of which an example would be the Poisson distribution.

Example 6.2. *In this example we show how to model the size of a group of people as a Poisson distribution and answer queries about the size of the group using the comparison predicates >/2 and =:=/2.*

```
n_people ~ poisson(6).

more_than_five:- n_people>5.
exactly_five:- n_people=:=5.

query(more_than_five).
query(exactly_five).
```

6.1.1 Type System

Before we formally specify the syntax of DC-ProbLog, we specify a hierarchical type system. To this end we will modify the type system given in [Sterling; Shapiro, 1994], one of the main references on Prolog. The base type in the type system of [Sterling; Shapiro, 1994] is the `Term` type, from which other types are subtyped. In other words: everything is a `Term`.

- `Term`
  - `Atomic`
    - `Atom`
    - `Number`
  - `Compound`

Logical variables (technically referred to as *meta-logical variables* [Sterling; Shapiro, 1994, Chapter 10]) are omitted from the type system hierarchy, as they can ground to

any type, depending on the context. In Figure 6.1 we give a diagrammatic representation of the Prolog type system. Note that ProbLog uses a different nomenclature: atom terms in ProbLog correspond to the union of atom terms and compound terms in the Prolog nomenclature of [Sterling; Shapiro, 1994]. In a first step we are going to change the Prolog type system to match the nomenclature of ProbLog.

`Term`

- `Number`
- `Atom`

In Table 6.1 we give a side by side comparison of the two type systems and in Figure 6.1 and 6.2 we give diagrammatic representations of both type systems.

Table 6.1: Comparison of the Prolog type system and the ProbLog type system.

example `Term`	Prolog		ProbLog
3	`Number`	`Atomic`	`Number`
`foo`	`Atom`		`Atom`
`bar(foo)`	`Compound`		

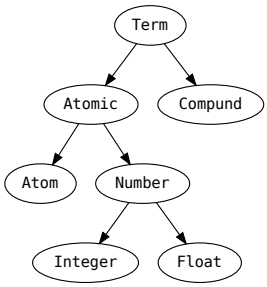


Figure 6.1: Diagrammatic representation of the hierarchical Prolog type system [Sterling; Shapiro, 1994]. The `Number` type is further subtyped into `Integer` and `Float`.

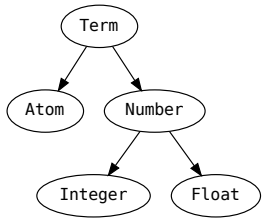


Figure 6.2: Diagrammatic representation of the hierarchical ProbLog type system. The `Number` type is further subtyped into `Integer` and `Float`.

Analogously to the just introduced type system of ProbLog, the base type for DC-ProbLog is again the `Term` type. At this point we modify the ProbLog type system by subtyping terms with a `Distrbution` type and a `Symbol` type. Dividing DC-ProbLog terms into `Distribution` terms and `Symbol` terms reflects the characterization of Poole [2010] of probabilistic programs as being constituted of independent choices (`Distribution`) and a deterministic system (`Symbol`).

Definition 6.1. *The DC-ProbLog type system is a hierarchical type system with each type being a disjoint union of its subtypes.*

Term

1. *Distribution*
   - *DiscreteD*
   - *ContinuousD*
   - *MixtureD*
2. *Symbol*
   - *Arithmetic*
     - *RandomVariable*
     - *Number*
   - *Atom*

A diagrammatic overview of the DC-ProbLog type system is given in Figure 6.3.

Distribution Terms

As already seen in Examples 6.1 and 6.2, `Distribution` terms determine the probability distribution according to which a random variable is distributed.

Definition 6.2 (`Distribution`)**.** *Terms of type `Distribution` are constructed by using a reserved random function symbol and applying it to a (possibly empty) tuple of input arguments, which maps the input tuple to a distribution term. Distribution terms are only allowed to be invoked as a second argument to the reserved distribution predicate `is`/2. This also means that random function symbols are only used there as well. DC-ProbLog subtypes the `Distribution` type into the `DiscreteD`, `ContinuousD`, and `MixtureD` types.*

`poisson(6)` and `normal(20,5)` are for instance of types `DiscreteD` and `ContinuousD`, respectively. Note that distributions of type `ContinuousD` denote *absolutely*

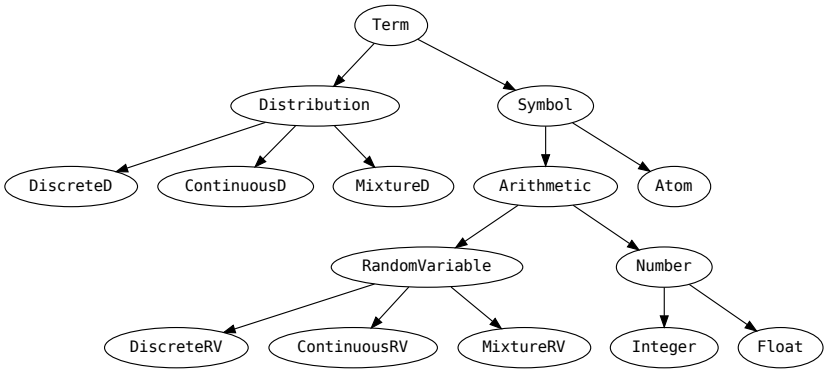


Figure 6.3: Diagrammatic representation of the hierarchical DC-ProbLog type system. Compared to Definition 6.1, the `Number` type is further subtyped into `Integer` and `Float` types and the `RandomVariable` type is subtyped into `DiscreteRV`, `ContinuousRV`, and `MixtureRV`. These subtypes of `RandomVariable` mimic the subtypes of the `Distribution` type.

continuous probability distributions[1]. We give an example of a distribution of type `MixtureD` in Example 6.13.

Symbol Terms

`Symbol` terms are the building blocks of the discrete structure that makes up a logic program and as such determine also the deterministic system of a probabilistic logic program. Beyond the reserved syntax, which is used to define the distribution terms, the user is free to use any identifier (following Prolog conventions) to declare `Symbol` terms – this includes standard Prolog terms such as lists.

Definition 6.3 (`Arithmetic`)**.** *Terms of type* `Symbol` *for which arithmetic operations are defined are of type* `Arithmetic`*.* `Arithmetic` *terms can be used as second argument to the builtin* `is`*/2 functor.* `Arithmetic` *terms are subtyped with* `RandomVariable` *terms and* `Number` *terms.*

[1] DC-ProbLog does not support *singular continuous* probability distributions such as the Cantor distribution.

We talk in more detail about `Arithmetic` terms and their evaluation in Subsection 6.1.4.

Definition 6.4 (`RandomVariable`)**.** *`RandomVariable` terms are named by the programmer using distributional facts with the binary infix predicate ~/2 and inherit their subtype from the distribution term. All random variables present in a DC-ProbLog program are named. `RandomVariable` terms are a subtype of the `Arithmetic` type.*

Examples of terms of type `RandomVariable` would be `temperature` and `n_people` in Examples 6.1 and 6.2, where we declared the random variables through so-called *distributional facts*:

```
temperature ~ normal(20,5).
n_people ~ poisson(6).
```

As mentioned in Defintion 6.4, `RandomVariable` terms inherit their type from the associated `Distribution` term, e.g. a `ContinuousD` term induces a random variable term of type `ContinuousRV`.

Note that not all probabilistic programming languages enforce the naming of random variables. In the functional probabilistic programming language Anglican [Wood et al., 2014], for instance, the user can declare anonymous random variables, much like anonymous functions, i.e. λ-functions.

Definition 6.5 (`Number`)**.** *Terms of type `Number` are used to express numerical data types such as integers and floating point numbers.*

Definition 6.6 (`Atom`)**.** *An atom is of the form `p(`t_1`,... ,` t_n `)` where `p` is a predicate of arity n and the* t_i *are `Symbol` terms [Fierens et al., 2015]. Predicates used to construct `Atom` terms must not clash with the builtin predicates and reserved random function symbols used to declare `Distribution` terms. Terms of type `Atoms` constitute the building blocks of the discrete structure of a probabilistic logic program.*

Examples of atoms terms would be `foo` or `bar(foo,42)`. The latter is constructed using the predicate bar/2 and the tuple of `Symbol` terms `(foo,42)`. `foo` is of type `Atom` and `42` is a `Number` term.

6.1.2 Syntax

Definition 6.7 (Deterministic Rules and Facts)**.** *A (normal) logic program is a set of rules. A rule is a universally quantified expression of the form*

```
h:- b1, ..., bn.
```

where `h` *is an* `Atom` *term and* $\{b_1, \ldots, b_n\}$ *is a set of n literals. Literals are either* `Atom` *terms, or their negation. The* `Atom` *term* `h` *is called the head of the rule and* b_1`, ... ,` b_n *the body. The latter represents the conjunction* $b_1 \wedge \cdots \wedge b_n$*. A fact is a rule that has* `true` *as its body and is simply written as*

```
h.
```

Definition 6.8 (Probabilistic Facts)**.** *Probabilistic facts are of the form* `p::fact`*.* `p` *is a term of type* `Number` *or* `RandomVariable`*, with* $0 \leq p \leq 1$ *and* `fact` *is a ground atom.*

Definition 6.9 (Distributional Facts)**.** *Distributional facts introduce random variable terms and associate them with a distribution using the reserved binary predicate* `~/2` *in infix notation. A distributional fact is a ground fact of the form* `rand ~ distribution`*, where* `distribution` *is a well-typed distribution term with output type* `DistributionX` *and* `rand` *is a ground term whose functor is not reserved. The distributional fact introduces* `rand` *as a random variable of type* `RandomVariableX`*, i.e. the type of the random variable term mimics the type of the distribution term. No two distributional facts can introduce the same random variable (i.e., use the same term as left hand side). Distributional facts are the link between the independent choices (in the form of distributions) and the discrete structure (in the of rules and facts).*

Definition 6.10 (Comparison predicates)**.** *DC-ProbLog (similar to Prolog and ProbLog) uses a fixed set of builtin binary predicates whose truth values depend on the* evaluation *of its arguments. For terms that can be evaluated (*`RandomVariable` *terms and* `Number` *terms) we use the set* $\bowtie = \{=:=, =\backslash=, <, >, =<, >=\}$ *of binary predicates written in infix notation. Each of these has the usual Prolog arithmetic interpretation of evaluating each side and comparing the results. Just as for Prolog arithmetic, truth values of atoms using these predicates are determined through an evaluation process that is external to the (probabilistic) logic program. Such atoms can be used in clause bodies, but never in clause heads (or as facts).*

Example 6.3. *Comparison predicates allow for the comparison of random variables to numbers.*

```
1 x ~ normal(0,3).
2 q:- x>0.
3 query(q).
```

A DC-ProbLog program consists of a countable set of ground distributional facts, a countable set of ground probabilistic facts, and a logic program (i.e. a finite set of facts and normal clauses), where bodies of normal clauses can use comparison predicates.

Definition 6.11. *A* **DC-ProbLog program** *consists of three disjoint sets:*

1. *A countable set of* **rules and facts**.
2. *A countable set of ground* **probabilistic facts**.
3. *A countable set of ground* **distributional facts**.

6.1.3 Multiple Dispatch

The attentive reader might have noticed that in Definition 6.10 we used the same comparison predicates, for `RandomVariable` terms as well as `Number` terms. In DC-ProbLog we resolve this clash through *multiple dispatching* [Castagna et al., 1995]. Multiple dispatching is the process of dynamically selecting at run time[2] a specific implementation of a function based on the types of the arguments that are fed as input to the corresponding function symbol. Such functions are referred to as *multimethods*. An early example of multiple dispatch is the *Common Lisp Object System* [Keene, 1989]. Multiple dispatch is also the programming paradigm on which the Julia programming language is built [Bezanson et al., 2017]. We reuse this idea in the context of probabilistic logic programming with random function symbols to retain a simple and neat syntax. Consider, for instance, the declaration of the normal distribution in Example 6.1. The definition of the function is the following:

$$\texttt{normal} : \texttt{Number} \times \texttt{Number} \to \texttt{ContinuousD}$$

However, the arguments of a normal distribution (its mean and standard deviation) might as well be random variables themselves. Multiple dispatching then allows us to *overload* the `normal` function symbol by using arguments of a different type[3]:

$$\texttt{normal} : \texttt{RandomVariable} \times \texttt{Number} \to \texttt{ContinuousD}$$

$$\texttt{normal} : \texttt{Number} \times \texttt{RandomVariable} \to \texttt{ContinuousD}$$

$$\texttt{normal} : \texttt{RandomVariable} \times \texttt{RandomVariable} \to \texttt{ContinuousD}$$

The exact implementation of these random function symbols is left to the designer of the implementation of the language.

6.1.4 Arithmetic Evaluation

Prolog implementations reserve a set of predicate names for system-related procedures. These system predicates are usually deployed when the goal is to perform efficient

[2] In the context of logic programming we understand as run time the time it takes to ground the logic program part of a DC-ProbLog program.

[3] For simplicity we omit that the standard deviation has to be a strictly positive (real) number, which can be enforced by extending the type system.

arithmetic operations on numbers, either by using specialized algortihms or specialized hardware. For example, when a user wants to add up the integers `3` and `5` they simply write `Sum is 3+5`. The predicate `is/2` takes the second argument, passes it to an external evaluation engine, which performs integer addition, and unifies the result with the logical variable `Sum`. It is rather obvious that the concept of multiple dispatch comes in handy here: the same functors can be used to perform the same operations on terms of different types. The correct functor is only dertermined during the grounding of the logic program, i.e. during runtime.

The key difference between the external arithmetic engine of Prolog (or ProbLog) and the external arithmetic engine of DC-ProbLog is that the external engine of the latter is capable of performing arithmetic operations (perform evaluations) with random variables. Combining this innovation with multiple dispatch allows us to retain the same syntax (same functor) to perform arithmetic operations on `RandomVariable` terms and/or `Number` terms.

Example 6.4. *A DC-ProbLog code snippet showing how random variable terms can be used in an arithmetic expression.*

```
1 x ~ normal(0,3).
2 q:- X is x+3, X>3.
3 query(q).
```

Multiple dispatch allows DC-ProbLog to call specific implementations of the comparison predicates in the external arithmetic engine. In other words, multiple dispatch allows for using the same predicate for semantically different terms. Comparing two terms of type `Number`, for example, materializes as a deterministic fact, whereas a comparison between terms of types `Number` and `RandomVariable` materializes as a probabilistic query. We will see how this extremely useful mechanism works in the next Section 6.2, where we also introduce the semantics of DC-ProbLog.

6.1.5 Relationship of Multiple Dispatch to Typing in Prolog

Traditionally Prolog is *weakly typed*. This means that only at run time (grounding time) it is checked whether a relation with certain arguments is present in the logic database, which is a set of ground relations. If the relation is not present, Prolog implementations do usually not throw an error but simply fail on the the goal that is being proven. For builtin predicates such as `is/2` or any comparison predicates, the run time type system is more strict and the Prolog implementation might actually throw an error. For instance, the presence of the relation `three=:=3` in a proof would throw an error. Such typed arithmetic relations can be thought of as *multirelations*, which have a different interpretations based on their type at run time. In other words, they

are the logic programming equivalent to multimethods: their interpretation depends on their predicate and the types of their arguments.

In DC-ProbLog we extend this idea of multirelations using the framework of multiple dispatch. In comparison to Prolog style multirelations, typed relations in DC-ProbLog can affect the structure of the probabilistic program – contrary to only selecting the right computation in the external arithmetic engine (e.g. floating point addition vs. integer addition). We will encounter examples in Section 6.2, where we rewrite a DC-ProbLog program as a DC-PLP program and where we have rewrite rules that do not only depend on the predicate of a relation but also on the types of a relation's arguments.

Note that in DC-ProbLog not only the builtin comparison predicates and the `is2` have a typed signature but also the random function symbols, e.g. `normal/2`. We can say that interpreted predicated and functors, that is predicates and functors that have a system internal meaning, possess a type signature. This results in a type dependent calculus.

Strongly Typed Prolog

While Prolog is traditionally not strongly/statically typed, there have been efforts to extend Prolog with such a type system. The most interesting effort is probably the one presented in Schrijvers et al. [2008], where the authors develop the idea of partially typing a Prolog program, i.e. typing is optional and untyped code is interpreted instead of compiled. Our efforts to utilize multiple dispatch within a (probabilistic) logic programming language can be understood as an effort orthogonal to the efforts towards statically typed Prolog. Introducing predicates with strongly typed signatures can also be regarded as adding interpreted predicates to the internal Prolog engine.

6.2 DC-PLP

In this section we will briefly sketch the main ideas for defining the semantics of DC-ProbLog. The semantics of DC-ProbLog are defined through the semantics of DC-PLP, a probabilistic logic programming language with both discrete and continuous random variables following Sato's distribution semantics [Sato, 1995]. DC-PLP is an assembly language with minimal syntax, whose main purpose is to define the declarative semantics of the language independently of existing syntax, inference algorithms or systems.

Poole [2010] has characterised probabilistic programming as independent choices plus deterministic systems, with (discrete) probabilistic logic programming being the case where the deterministic system is given by a logic program [Poole, 2010].

DC-PLP builds upon this idea, but relaxes the independence assumption by explicitly distinguishing two types of dependencies: 1) those expressed by the logic program (i.e. deterministic system), and 2) those expressed by using random variables as parameters of distributions for other random variables.

DC-PLP serves as a stepping stone towards DC-ProbLog: the semantics of DC-ProbLog are defined through the semantics of DC-PLP and an additional unique deterministic program transformation that takes DC-ProbLog programs and maps them to DC-PLP programs.

The key idea behind DC-PLP is to build the semantics of a program in three steps:

1. Define a **countable set of random variables** and associate a unique parametric distribution to each random variable through a logic program that defines terms of the form `rv(name,dist)`. We do not use the `is/2` infix notation for distributional clauses to make explicit that this is restricted to fully deterministic bodies. We do not introduce syntax for probabilistic facts, as these can be modeled through Boolean random variables and rules. For valid programs, this will give us a unique joint probability measure over these random variables that factors into the individual distributions.

2. Define a **countable set of measurable Boolean queries** over the random variables. These correspond to the type of tests that are written with comparison predicates (<, =<, >, >=, =:=, =\=). However, we here define these on the term level and wrap them into a single predicate `test/1` for ease of uniform notation.

3. Define a **logic program** that computes consequences of the Boolean queries. To include a Boolean query `q` in the body of a clause, it gets wrapped into a term of the form `test(q)`.

The first step defines all random choices, while the second and third step form the deterministic system. As the second and third step are deterministic functions of the random variables, the measure defined in the first step extends to the Boolean queries and from there to the terms in the logic program.

In contrast to the Distributional Clauses probabilistic programming language [Gutmann et al., 2011], where several conditional distributional clauses can jointly define the distribution of a single random variable through a complex logical factorization, DC-PLP uses a single fact to define a random variable unconditionally and associates a single parameterized distribution to it. As we still have the full power of normal logic programs to model the deterministic system, this does not restrict expressivity, but simplifies the formal definition of the semantics. When modeling conditional distributions, users have a range of choices, from encoding the entire conditional probability distribution into a single term within the defining `rv/2` fact, all the way

to a set of individually named choice random variables with minimal use of random variables as parameters tied together by a deterministic system, or any intermediate factorisation. As already discussed by Poole [2010], not all choices work equally well with all inference algorithms. DC-PLP provides these choices within the same semantical framework, leaving it to system developers to focus the syntax towards a subset of choices and/or to exploit the different choices when integrating different inference approaches into their system.

6.2.1 From DC-ProbLog to DC-PLP

In order to illustrate DC-PLP we are now going to map a high-level probabilistic logic program written in DC-ProbLog to DC-PLP, where we can then identify the three components that constitute a DC-PLP program:

1. countable set of random variables
2. countable set of measurable Boolean queries
3. logic program computing consequences of Boolean queries

We will perform the transformation in two steps.

Example 6.5 (DC-ProbLog Program)**.** *This example program models the correct working of a machine. The probability distribution of the* `temperature` *of the machine depends on whether it is a* `hot` *day or not.*

```
machine(1).

0.2::hot.
0.99::cooling(1).

temperature ~ normal(27,5):- hot.
temperature ~ normal(20,5):- \+hot.

works(N):- machine(N), cooling(N).
works(N):- machine(N), temperature<25.0.

query(works(1)).
```

Note that we introduced distributional clauses *in this example, which let us write down conditional probability distributions.*

Definition 6.12. *Distributional clauses are syntactic sugar to concisely write down conditional probability distributions of random variables. We necessitate the bodies of distributional clauses with the same random variable in their heads to be pairwise logically inconsistent. A DC-ProbLog program containing a random variable* `rand` *involved in N distributional clauses*

$$\texttt{rand} \sim \texttt{dist}_i \texttt{ :- } b_i$$

with mutually logically inconsistent bodies b_i *(*$1 \leq i \leq N$*) is equivalent to a DC-ProbLog program with N fresh random variables* $\texttt{rand}_i$ *(*$1 \leq i \leq N$*) involved in N distinct distributional facts*

$$\texttt{rand}_i \sim \texttt{dist}_i.$$

Each clause initially containing `rand` *is replaced in the equivalent program that does not contain distributional clauses for* `rand` *by N clauses, where* `rand` *is substituted by* $\texttt{rand}_i$ *and its body is conjoined with* b_i *for every i with* $1 \leq i \leq N$.

Example 6.6 (DC-ProbLog Program with Eliminated Distributional Clauses)**.** *In this example we apply the definition of a distributional clause to reduce the program in Example 6.5 to a DC-ProbLog program, according to Definition 6.11.*

```
1  machine(1).
2
3  0.2::hot.
4  0.99::cooling(1).
5
6  temperature(hot) ~ normal(27,5).
7  temperature(not_hot) ~ normal(20,5).
8
9  works(N):- machine(N), cooling(N).
10 works(N):- machine(N), temperature(hot)<25.0, hot.
11 works(N):- machine(N), temperature(not_hot)<25.0, \+hot.
12
13 query(works(1)).
```

Note how the distributional clauses were rewritten as distributional facts with fresh random variables, the clause involving the random variable `temperature` *was duplicated, and* `temperature` *was replaced with the fresh random variables. At the same time the bodies of the duplicated clauses were conjoined with the respective body of the initial distributional clauses (*`hot` *and* `\+hot`*).*

Example 6.7 (DC-PLP Program)**.** *We rewrite the program in Example 6.6 as a DC-PLP program.*

```
rv(hot, flip(0.2)).
rv(cooling(1), flip(0.99)).
rv(temperature(hot), normal(27,5)).
rv(temperature(not_hot),normal(20,5)).


test(cooling(N)=:=1)
test(hot=:=1).
test(hot=:=0).
test(temperature(hot)<25.0).
test(temperature(not_hot)<25.0).



machine(1).
works(N):- machine(N), test(cooling(N)=:=1).
works(N):-
        machine(N),
        test(temperature(hot)<25.0),
        test(hot=:=1).
works(N):-
        machine(N),
        test(temperature(not_hot)<25.0),
        test(hot=:=0).

query(works(1)).
```

We are now able to clearly identify the three components of a DC-PLP program. In the first part we see the random variables (representing independent choices). Here we also also included the probabilistic facts from the intial DC-ProbLog program. DC-PLP encodes probabilistic facts using the `flip/2` *random function symbol, which returns either* 1 *or* 0 *with the probability given as the argument. In the second block of code we see the countable set of measurable Boolean queries, and the last part contains the logic program.*

Multiple Dispatch and Comparison Predicates

As mentioned earlier, multiple dispatching is of outstanding usefulness when applying comparison predicates to different type signatures that result in semantically different

terms. We will illustrate this by example.

Example 6.8. *Consider the following DC-ProbLog code snippet with no random variables.*

```
people(N):- N is 8.
large_group:- people(N), N>6.
```

During grounding, the comparison operator in the second line will dispatch to a deterministic comparison, as both sides of the comparison are of type `Number`*. The arithmetic engine replaces the grounded comparison term with atom* `true`*.*

Example 6.9. *Consider now a DC-ProbLog code snippet where a comparison predicate is present that receives as one of its arguments a term of type* `RandomVariable`*.*

```
people ~ poisson(6).
large_group:- people>6.
```

During grounding the comparison predicate dispatches to a probabilistic comparison. Instead of replacing the comparison by either `true` *or* `false`*, the arithmetic engine defers the evaluation and ultimately adds a* `test(people>6)` *term to the set of measurable Boolean queries.*

6.2.2 Valid DC-PLP Programs

Definition 6.13. *A DC-PLP program $\mathbb{P}$ consists of:*

1. *A countable set P of random variables.*
2. *A countable set Q of measureable Boolean queries over the random variables.*
3. *A logic program L, which has access to the set of Boolean queries.*

Definition 6.14. *A DC-PLP $\mathbb{P}$ program is called valid if the following three conditions hold.*

1. *The program $\mathbb{P}$ is distribution stratisfied, that is, there exists a function rank$(\cdot)$ that maps ground random variables, of the form* `rv(rand,dist(`$d_1,\ldots,d_n$`))`*, to $\mathbb{N}$ and that satisfies rank(*`rand`*) > rank(*d_i*) for every i ($1 \leq i \leq n$).*
2. *The logic program L is* sound *[*Riguzzi; Swift, *2013], i.e. every model of L is a two-valued well-founded model [*Van Gelder *et al., 1991;* Riguzzi; Swift, *2013].*

3. *Each model of L is derivable from a finite set of Boolean queries in Q.*

The first condition ensures that programs do not exhibit cyclic functional dependencies, disallowing programs of the form:

```
x ~ normal(y,1).
y ~ normal(x,1).
q:- y>3.
query(q).
```

The third condition is necessary to satisfy the finite support condition of the distribution semantics [Sato, 1995]. Definition 6.14 follows closely the semantics of ProbLog [Fierens et al., 2015], which guarantees that a DC-ProbLog program without distributional facts is in fact a ProbLog program.

Negation

In order to add negation to the hybrid probabilistic logic programming language Distributional Clauses, Nitti et al. [2016a] resorted to *negation as failure* within a *Selective Linear Definite (SLD) resolution* [Sterling; Shapiro, 1994] inference scheme to patch the initial (negation-free) semantics given in [Gutmann et al., 2011]. The semantics then follow [Przymusinski, 1988]'s perfect model semantics.

The well-founded semantics already account for negation in logic programs and we do not have to add any extra machinery to add negation. On a side note: in the absence of negation the well-founded model of a logic program is identical to the *least Herbrand model*.

Probability of a DC-PLP Program

Definition 6.15. *Let MOD(L) be the set of all two-valued well-founded models of L, which is the logic program of a valid DC-PLP program $\mathbb{P}$, and $Q(l)$ the set of Boolean queries used to derive $l \in MOD(L)$. Furthermore, let $var(Q(l))$ be the set of random variables on which $Q(l)$ depends. The probability of the model l is then given by:*

$$p(l) = \int \left(\prod_{q \in Q(l)} [\![q]\!] \right) \underbrace{\left(\prod_{x_l \in var(Q(l))} D_l(x_l) \right)}_{=: D_l(\mathbf{x}_l)} \underbrace{\left(\prod_{x_l \in var(Q(l))} dx_l \right)}_{=: d\mathbf{x}_l} \tag{6.1}$$

$D_l(x_l)$ is the probability distribution according to which the variable x_l is distributed and which is given by the countable set P.

The probability of the DC-PLP program $\mathbb{P}$ is given by:

$$p(\mathbb{P}) = \sum_{l \in MOD(L)} p(l) = \sum_{l \in MOD(L)} \int \prod_{q \in Q(l)} [\![q]\!] D_l(\mathbf{x}_l) d\mathbf{x}_l \tag{6.2}$$

Let us compare the definition of the probability of a DC-PLP program to the definition weighted model integration, and in particular the form of WMI given in Equation 4.4:

$$\sum_{\langle \phi_i, \omega_i \rangle \in \mathcal{W}_f} \sum_{\mathbf{b}} \int [\![\phi_i(\mathbf{x}, \mathbf{b})]\!] \omega_i(\mathbf{x}) d\mathbf{x} \tag{4.4}$$

We cannot but appreciate the striking similarity between both Equations. The only substantial difference is that Equation 6.2 uses Lebesgue integration to marginalize out Boolean random variables, whereas Equation 4.4 uses a summation. Other than that it is straight forward to map one equation to the other:

- We associate the sum over the models $l \in MOD(L)$ to the sum over different SMT($\mathcal{LRA}$) formulas ϕ_i in weighted model integration
- We associate a specific product of Iverson brackets of Boolean queries $\prod_{q \in Q(l)} [\![q]\!]$ to the Iverson bracket of a specific SMT($\mathcal{LRA}$) formula $[\![\phi_i]\!]$, which is a conjunction of atomic SMT($\mathcal{LRA}$) formulas.
- We associate the distributions in D_l with the weight function ω_i.

Valid DC-ProbLog Programs with Distributional Clauses

Definition 6.16. *A DC-ProbLog program $\mathbb{P}$ with distributional clauses is valid if it can be mapped to an equivalent valid DC-PLP program. For such a mapping to exist the following two criteria have to hold:*

V1 The bodies of ground distributional clauses with the same random variable in their heads are pairwise logically inconsistent.

V2 The program $\mathbb{P}$ is distribution stratisfied, that is, there exists a function rank$(\cdot)$ that maps ground terms to $\mathbb{N}$ and that satisfies the following properties:

1. *for each ground instance of a distributional clause of the form* `rand~dist(`d_1`,...,`d_n`)):-b` *it holds that rank(*`rand`*) > rank(*d_i*) for every i (*$1 \leq i \leq n$*).*
2. *for each ground instance of a distribution clause* `rand~dist:-`b_1`,...,`b_n *it holds that rank(*`rand ~ dist`*) > rank(*b_i*), for all i.*

3. *for each ground instance of another program clause* `h:- b1,...,bn` *it holds that* $rank(h) \geq rank(b_i)$*, for all i.*
4. *for each ground term* `b` *that contains a random variable* `rand`*,* $rank(b) \geq rank(\texttt{rand\~dist})$ *(with* `rand~dist` *the head of the distribution clause defining* `rand`*).*

The criterion V1 is equivalent to the validity criterion V1 given in [Gutmann et al., 2011, Definition 3], which requires that there is a unique ground distribution for each ground random variable `rand`. V2 is a carbon copy of the second validity criterion of [Gutmann et al., 2011, Definition 3], with the exception of the first point of V2. We added it, as distributional clauses with cyclic functional dependencies were not disallowed in [Gutmann et al., 2011].

Mapping distributional clauses to distributional facts is akin to mapping intentional probabilistic facts (e.g `0.2::a:- b.`) to probabilistic facts (and deterministic rules). Note that DC-ProbLog allows, just as ProbLog, the usage of intentional probabilistic facts, with the same semantics as in ProbLog. In absence of any distributional facts and clauses, a DC-ProbLog program reduces to a ProbLog program.

Difference to Distributional Clauses

This is also a good point to illustrate a major difference between the semantics of random variables in Distributional Clauses and DC-ProbLog. Take the code snippet below:

```
x ~ normal(20,2).

q(1):- x>20.
q(2):- y>20.

query(q(1)).
query(q(2)).
```

The first query succeeds and returns as an answer $p($`query(q(1))`$) = 0.5$. The second query, however, throws an error as there is no term of type `RandomVariable` declared that is named `y`. The second query would also fail if there were an `Atom` term named `y`. In this case the lefthand side of `y>20` would be ill-typed.

Let us now write the same program as a Distributional Clauses program:

```
x ~ normal(20,3).
```

```
q(1) ← ≃(x)<20.
q(2) ← ≃(y)<20.

query(q(1)).
query(q(2)).
```

The first query will again return $p(\text{query}(\text{q}(1))) = 0.5$ (or an approximation thereof) but the second query will not throw an error but return a probability of zero. How does this work? The term $\simeq(y)$ tries to unify with the value of the random variable y. As there is no such random variable, this silently fails and the body of the clause whose head is q(2) is simply not satisfied.

If we were to negate the comparison predicate in the body of the q(2) clause, i.e. write:

```
q(2) ← ≃(y)<20.
```

we would obtain for the second query: $p(\text{query}(\text{q}(1))) = 1$. Negating the corresponding comparison predicate in DC-ProbLog would still result in an ill-typed term, i.e. an error would be thrown.

The comparison we just performed does not only illustrate a semantic difference between DC-ProbLog and Distributional Clauses but also a conceptual one. In DC-ProbLog the values of of random variables never enter the realm of the logic program: values of random variables only live in the external arithmetic engine of DC-ProbLog. In contrast, random variables in Distributional Clauses are part of the logic program: the values of random variables are stored in a logic data base and can be accessed through logical unification. Distributional Clauses currently does this via SLD resolution.

6.3 Inference

Similar to following ProbLog when defining the semantics for DC-ProbLog (using well-founded semantics), we will now also design an inference algorithm for our implementation of DC-ProbLog, inspired by the inference mechanisms present in the ProbLog2 system [Dries et al., 2015].

A ProbLog program is first converted into a ground (probabilistic) logic program. This ground program is subsequently transformed into a deterministic and decomposable weighted propositional logic formula. Evaluating this formula, i.e. computing the weighted model count, yields the probability queried by the program. A diagrammatic representation is given in Figure 6.4.

Inference in our implementation of DC-ProbLog follows a similar principle, with the difference that a probabilistic program is transformed to a weighted and compiled

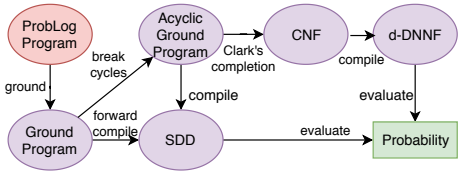


Figure 6.4: Overview of the primary program transformation steps in the ProbLog2 system [Dries et al., 2015; Zuidberg Dos Martires et al., 2019a].

SMT formula. The probability of a query `query(q)` in a DC-ProbLog program is then obtained by computing the probability of an equivalent SMT formula ϕ_q using weighted model integration:

$$p(\texttt{query(q)}) = p(\phi_q) = \text{WMI}(\phi_q, w_q) \tag{6.3}$$

In Algorithm 6.1 we outline the steps enabling inference. They closely follow the steps presented in [Fierens et al., 2015, Section 5].The correctness of the algorithm can be proven by again following [Fierens et al., 2015] and using abstractions of SMT formulas, which are Boolean formulas, instead of Boolean formulas directly.

Algorithm 6.1 (DC-PLP Inference)**.** *The inference algorithm takes as input a DC-PLP program $\mathbb{P}$, consisting of a countable set P of random variables, a countable set of measurable Boolean queries Q, and a purely logic program L. The algorithm then maps $\mathbb{P}$ through the following steps to a WMI problem.*

1. *Ground out the purely logic program L of $\mathbb{P}$ and obtain the ground program L_g. This will also ground the Boolean queries in Q, leading to Q_g.*
2. *Convert the ground logic program consisting of L_g and the set of measurable queries Q_g to an equivalent SMT formula ϕ_q.*
3. *Define a weight function w_q, which corresponds to the countable set of random variables in P.*

We just mapped the probability encoded by a DC-PLP program to a weighted model integral. Inference in DC-PLP/DC-ProbLog can hence be performed by any algorithm capable of computing the weighted model integral in question.

Example 6.10 (Mappint DC-PLP to WMI)**.** *Consider again the DC-PLP program in Example 6.7. If we ground the purely logic component of the program we obtain the ground program.*

```
rv(hot, flip(0.2)).
rv(cooling(1), flip(0.99)).
```

```
3  rv(temperature(hot), normal(27,5)).
4  rv(temperature(not_hot),normal(20,5)).
5
6
7  test(cooling(1)=:=1)
8  test(hot=:=1).
9  test(hot=:=0).
10 test(temperature(hot)<25.0).
11 test(temperature(not_hot)<25.0).
12
13
14 machine(1).
15 works(1):- machine(1), test(cooling(1)=:=1).
16 works(1):-
17         machine(1),
18         test(temperature(hot)<25.0),
19         test(hot=:=1).
20 works(1):-
21         machine(1),
22         test(temperature(not_hot)<25.0),
23         test(hot=:=0).
24
25 query(works(1)).
```

We are now able to map the grounded logic program and the Boolean queries to an SMT formula ϕ_q:

$$\begin{aligned}\phi_q \leftrightarrow\ & machine(1) \wedge (cooling(1) = 1) \vee \\ & machine(1) \wedge (temperrature(hot) < 25.0) \wedge (hot = 1) \vee \\ & machine(1) \wedge (temperature(not\_hot) < 25.0) \wedge (hot = 0) \\ \leftrightarrow\ & (cooling(1) = 1) \vee \\ & (temperrature(hot) < 25.0) \wedge (hot = 1) \vee \\ & (temperature(not\_hot) < 25.0) \wedge (hot = 0)\end{aligned} \tag{6.4}$$

The weight function w_q is simply obtained by multiplying together the functions corresponding to the random variables declared in the DC-PLP program.

6.3.1 Conditional Probabilities

For DC-ProbLog programs that include evidence on Boolean random variables the probability is simply obtained by computing two weighted model integrals, one for the evidence ϕ_e and one for the conjunction of the evidence with the query $\phi_e \wedge \phi_q$:

$$p(\phi_q|\phi_e) = \frac{p(\phi_q, \phi_e)}{p(\phi_e)} = \frac{\text{WMI}(\phi_q \wedge \phi_e, w_{qe})}{\text{WMI}(\phi_e, w_e)} \tag{6.5}$$

This lets us, for instance, compute the probability of the query in Example 6.1, where we saw that we can express a conditional probability by using the reserved `evidence/1` predicate. DC-ProbLog also has a binary predicate `evidence/2`, where the second argument is either the `true` symbol or the `false` symbol. The former is equivalent to using the unary version, while the latter allows one to express negative evidence. These predicates are also present in ProbLog.

6.3.2 Zero Probability Events and Measurements

While the `evidence` predicate allows us to express conditional probabilities where we condition on Boolean random variables, its semantics does not extend directly to conditioning on continuous random variables: a continuous random variable can neither be true nor false. In order to allow a user to condition on continuous random variables in DC-ProbLog, we introduce the `observation/2` predicate.

Example 6.11. *We model the size of a ball as a mixture of different beta distributions, depending on whether the ball is made out of wood or metal (Line 1)*[4]*. We would now like to know the probability of the ball being made out of wood given that we have a measurement of the size of the ball. In order to condition on a continuous random variable we introduce the `observation/2` predicate, which has an analogous functionality as the evidence predicates for Boolean random variables.*

```
1 3/10::material(wood);7/10::material(metal).
2
3 size~beta(2,3):-material(metal).
4 size~beta(4,2):-material(wood).
5
6 observation(size,4/10).
7 query(material(wood)).
```

[4] Annotated disjunctions are used to concisely write down mutually exclusive Boolean random variables. Internally they are compiled down to probabilistic facts and deterministic rules.

This DC-ProbLog program encodes the conditional probability:

$$p(\texttt{material(wood)}|\texttt{size=4/10}) \tag{6.6}$$

In Example 6.11 we can see that the `observation/2` predicate takes for its first argument a *named* random variable and for its second argument a `Term` of type `Number`. Restricting the signature of the `observation/2` predicate in this way, as opposed to many other probabilistic programming languages, has the benefit of inducing unambiguous probability distributions. This problem is related to the Borel-Kolomogorov paradox [Kolmogorov, 1950; Gyenis et al., 2017], which is caused by a zero-division as the probability of the observation is zero:

$$p(\texttt{observation(size,4/10)}) = p(\texttt{size=4/10}) = 0 \tag{6.7}$$

In other words, a random variable with infinitely many outcomes will take a specific value with probability zero.

For a formal discussion we define the conditional probability $p(\phi_q|\phi_e)$ following Kadane [2011]. Let us also, for now, make the explicit disctinction between the probability P and the probability density function $p : \frac{d}{dx}P(X \leq x) = p(x)$ (analogously to Nitti et al. [2016a]).

$$P(\phi_q|\phi_e) = P(\phi_q|rv = v) \qquad (\phi_e \leftrightarrow rv = v) \tag{6.8}$$

$$= \lim_{\Delta v \to 0} \frac{P(\phi_q, rv \in [v - \Delta v/2, v + \Delta v/2])}{P(rv \in [v - \Delta v/2, v + \Delta v/2])} \tag{6.9}$$

$$= \lim_{\Delta v \to 0} \frac{\int [\![x \models \phi_q]\!] p(x, rv = v) dx \cancel{\Delta v}}{p(rv = v) \cancel{\Delta v}} \tag{6.10}$$

$$= \frac{\int [\![x \models \phi_q]\!] p(x, rv = v) dx}{p(rv = v)} \tag{6.11}$$

This means that we do not divide anymore by zero but by the number obtained when evaluating the probability distribution at the observed point. For example, when evaluating `beta(2,3)` (cf. Example 6.11, Line 3) at 4/10 we get 1.7280 instead of zero. Note that this is not a probability between zero and one.

For a more general case, where we do not restrict the signature of the `observation/2` predicate, it might be impossible to take the limit in Equation 6.9 as such a limit might not exist. Nevertheless, some languages lift this restriction, which results in possibly semantically ill-defined programs, such as in Distributional Clauses [Nitti et al., 2016a, Section 3.2]. A more sophisticated method to tackle the non-uniqueness of the conditional probability is *disintegration* [Shan; Ramsey, 2017], where instead

of returning a conditional probability, a family of conditional probabilities is returned (representing an uncountable number of limits). Other languages, such as BLOG, simply assume uniqueness of the limit [Wu et al., 2018].

At this point, the type signature we impose on the `observation/2` predicate might seem too restrictive. Take for example the following conditional probability [Nitti et al., 2016a]:

$$P(\mathit{nationality}|\mathit{height}{=}160 \vee \mathit{height}{=}180) \tag{6.12}$$

This probability cannot be expressed using the `observation/2` predicate. However, if we think about what a conditioning set represents in a probabilistic programming and machine learning context, the example above appears quite odd. The conditioning set tells us which observations we have made about the world, i.e. data that we have collected and we collected this data using a physical device (tape measure for the height of a person). The example in Equation 6.12 now tells us that our measuring device returned two numbers for a single measurement ...

We just said that a measurement consists of retrieving a number from a measuring device. This is not entirely correct. In natural sciences a measurement consists always of a number and an upper and lower error bound, which effectively eliminates the problem of zero probability events. However, given the ubiquity of single point measurement, we feel that a probabilistic programming language that is not able to handle single point measurements is greatly hurt in its expressivity. In Subsection 6.4.1, we will see an example where zero probability events play an important role in computing the correct probability.

To conclude, in DC-ProbLog, the `observation/2` predicate allows a user to condition on zero probability events, while at the same time the imposed restrictions ensure that the encoded probability distribution is well defined.

6.3.3 Algebraic Likelihood Weighting

Probabilistic programming languages that successfully handle mixtures of discrete and continuous random variables, including conditioning with zero probability events deploy either purely symbolic inference algorithms [Gehr et al., 2016; Shan; Ramsey, 2017] or extend the likelihood weighting [Fung; Chang, 1990] algorithm leading to a hybrid symbolic-approximate inference algorithm [Nitti et al., 2016a; Wu et al., 2018].

In this section we frame the *lexicographic likelihood weighting* (LLW) algorithm [Wu et al., 2018] in an algebraic model counting context, which we dub ALW. LLW is based on ideas presented by Nitti et al. [2016a] and combines sampling based probabilistic inference with symbolic inference to correctly handle mixtures of discrete and continuous random variables. The advantage of framing LLW in as an algebraic

model counting problem is that the resulting algorithm becomes agnostic of the underlying inference mechanism (e.g. Monte Carlo sampling). Moreover, we can simply reuse algorithms already developed in the first part of the thesis in the context of weighted model integration in order to perform inference in DC-ProbLog.

Following the steps taken in Section 3.2 we define again a labeling function and a semiring.

Definition 6.17 (ALW labeling function)**.** *Let l be a literal. Then the label of the literal l is given by:*

$$\alpha_{ALW}(l) \coloneqq \begin{cases} (p(l), 0) & \text{if } l \text{ is a Boolean variable} \\ ([\![c(\mathbf{x})]\!], 0) & \text{if } l \text{ is an atomic formula abstraction} \\ (p(v), 1) & \text{if we make the observation } (x = v) \leftrightarrow l \end{cases}$$

In the first case, $p(l)$ denotes the probability for l being true, in the second case, $c(\mathbf{x})$ denotes the condition of which l is the abstraction, and in the third case $p(v)$ denotes the value of the probability distribution evaluated at v.

The label of a negated literal $\neg l$ is given by:

$$\alpha_{ALW}(\neg l) \coloneqq \begin{cases} (1 - p(l), 0) & \text{if } l \text{ is a Boolean variable} \\ ([\![\neg c(\mathbf{x})]\!], 0) & \text{if } l \text{ atomic formula abstraction} \\ (1, 0) & \text{if we make the observation } (x = v) \leftrightarrow l \end{cases}$$

Definition 6.18 (ALW semiring)**.** *The elements of the semiring $\mathcal{S}_{ALW}$ are given by the set*

$$\mathcal{A}_{ALW} \coloneqq \{(a, d)\} \tag{6.13}$$

where a denotes a real-valued weight and d the positive integer number of probability densities (continuous probability distribution) that have contributed to a. The neutral elements $e^{\oplus}$ and $e^{\otimes}$ are defined as:

$$e^{\oplus} \coloneqq (0, 0) \qquad\qquad e^{\otimes} \coloneqq (1, 0) \tag{6.14}$$

For the addition and multiplication we define:

$$(a_1, d_1) \oplus (a_2, d_2) \coloneqq \begin{cases} (a_1 + a_2, d_1) & \text{if } d_1 = d_2 \\ (a_1, d_1) & \text{if } d_1 < d_2 \\ (a_2, d_2) & \text{if } d_1 > d_2 \end{cases} \tag{6.15}$$

$$(a_1, d_1) \otimes (a_2, d_2) \coloneqq (a_1 \times a_2, d_1 + d_2) \tag{6.16}$$

Lemma 6.1. *The structure $\mathcal{S}_{ALW} = (\mathcal{A}_{ALW}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ is a commutative semiring.*

Lemma 6.2. *The pair* $(\oplus, \alpha_{ALW})$ *is neutral.*

Assume now that the algebraic model count of $\phi_q \wedge \phi_e$ and ϕ_e evaluate respectively to:

$$\text{AMC}(\phi_q \wedge \phi_e, \alpha_{ALW}) = (\Psi_{qe}, d_{qe}) \tag{6.17}$$

$$\text{AMC}(\phi_e, \alpha_{ALW}) = (\Psi_e, d_e) \tag{6.18}$$

The conditional probability is then given by:

$$p(\phi_q|\phi_e) = \begin{cases} \frac{\text{WMI}(\phi_q \wedge \phi_e, w_{qe})}{\text{WMI}(\phi_e, w_e)} & \text{if } d_{qe} = d_e \\ 0 & \text{if } d_{qe} > d_e \end{cases} \tag{6.19}$$

Note, in Equations 6.17 to 6.19 we have omitted the dependence of the algebraic model counts and the weighted model integration on the Boolean and/or real variables.

6.4 Two Showcase Examples

We present now two showcase probabilistic programs expressed in DC-ProbLog. The programs and the source code for the implementation of the language can be found online[5]. The DC-ProbLog implementation uses the probabilistic programming language Pyro [Bingham et al., 2018] in the backend to handle continuous random variables.

6.4.1 The Indian GPA problem

The *Indian GPA problem* was initially proposed by Stuart Russell as an example problem that contemporary probabilistic programming languages were not able to handle, as they did not correctly perform probabilistic inference for probability distributions that are mixtures of discrete and continuous random variables.

Example 6.12. *The Indian GPA problem models US-American and Indian students and their GPAs. Both receive scores on the continuous domain. From zero to four (American) and from zero to 10 (Indian), cf. Line 10 and 16. With probability non-zero both student groups can also obtain marks at the extremes of the respective scales (Lines 11, 13, 17, 19). We observe now that a student has a GPA of four and we would like to know the probability of this student being American or Indian. The correct answer is p(*`american`*)=1 and p(*`indian`*)=0, which is what DC-ProbLog returns, too.*

[5] https://github.com/ML-KULeuven/problog/tree/dcproblog_develop/problog/tasks/dcproblog

```
1/4::american;3/4::indian.

19/20::isdensity(a).
99/100::isdensity(i).

17/20::perfect_gpa(a).
1/10::perfect_gpa(i).


gpa(a)~uniform(0,4):- isdensity(a), american.
gpa(a)~delta(4.0):-
        \+isdensity(a), perfect_gpa(a), american.
gpa(a)~delta(0.0):-
        \+isdensity(a), \+perfect_gpa(a), american.

gpa(i)~uniform(0,10):- isdensity(i), indian.
gpa(i)~delta(10.0):-
        \+isdensity(i), perfect_gpa(i), indian.
gpa(i)~delta(0.0):-
        \+isdensity(i), \+perfect_gpa(i), indian.

gpa(student)~delta(A):- A is gpa(a).
gpa(student)~delta(I):- I is gpa(i).

observation(gpa(student),4.0).
query(american).
query(indian).
```

DC [Nitti et al., 2016a] and BLOG [Wu et al., 2018] do return the correct probabilities as well, however, the advantage of using Sampo in conjunction with ALW gives the correct result without drawing any samples.

In Example 6.12 we wrote the probability distribution of `gpa(a)` and `gpa(i)` using uniform and Dirac delta distributions. This allowed us to distribute the random variables `gpa(a)` and `gpa(i)` according to a discrete-continuous mixture distribution. Alternatively, we could also have handed off the discrete-continuous mixture completely to the independent choice system, i.e. using a specific random function symbol for mixture distributions.

Example 6.13. *An alternative formulation of the program in Example 6.12, relying on random variables rather than on logical rules.*

```
1/4::american;3/4::indian.

gpa(student) ~ mixture(
        (uniform(0,4),0.95),
        (4,0.425),
        (0,0.075)):- american.
gpa(student) ~ mixture(
        (uniform(0,10),0.99),
        (10,0.001),
        (0,0.009)):- indian.

observation(gpa(student),4).
query(american).
query(indian).
```

The conditional random variables for `gpa(student)` *are now of type* `MixtureD`*.*

Comparing this Example 6.13 to Example 6.12 shows that there is a choice to be made from the user's perspective of how much one wants to encode in the deterministic system (using logical rules) or in the independent choice system (random function symbols).

6.4.2 Bayesian Learning

We are now also briefly going to show how Bayesian learning can be performed with DC-ProbLog. To this end we also introduce the `query_density/1` predicate, which allows us to retrieve a mixture of probability density functions, instead of a probability. When using Sampo, `query_density/1` returns a list of weighted samples, for each component of the mixture.

Example 6.14. *We model a coin flip scenario where the prior probability of the coin turning up heads is distributed according to a mixture of two beta distributions. DC-ProbLog then allows us to learn the posterior distribution by taking into account the data in Lines 6 to 8.*

```
0.2::a.
b~beta(1,1):- a.
b~beta(1,2):- \+a.
B::coin_flip(N):- B is b.

evidence(coin_flip(1), true).
```

```
evidence(coin_flip(2), false).
evidence(coin_flip(3), true).

query_density(b).
```

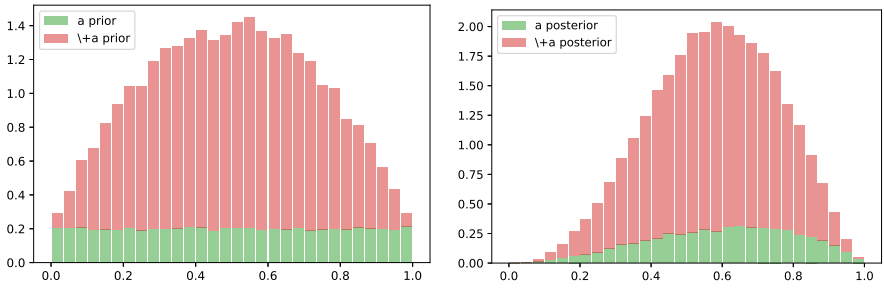


Figure 6.5: Prior (left) and posterior (right) for the coin flip scenario modeled in Example 6.14. The stacked histograms were plotted using 20000 samples for each beta distrbution.

6.5 Related Languages

In recent years a plethora of different probabilistic programming languages have been developed. We will now discuss some related languages and point to similarities and differences between them and DC-ProbLog.

From a semantics perspective the language closest related to DC-ProbLog is obviously ProbLog. DC-ProbLog extends ProbLog with continuous random variables and reduces to ProbLog in the absence of continuous random variables. In other words, valid ProbLog programs are a strict subset of valid DC-ProbLog programs.

From the perspective of expressive power, DC-ProbLog is closely related to Distributional Clauses (DC) [Gutmann et al., 2011; Nitti et al., 2016a], which inspired DC-ProbLog. The main differences can be seen in the cleaned up syntax of DC-ProbLog and a semantics that is based on sound well-founded models (instead of the T_P operator). Furthermore, contrary to DC, the inference algorithms in our implementation of DC-ProbLog are based on knowledge compilation, combined with either exact inference (Symbo) or approximate inference (Sampo). This is advantageous when a probabilistic logic program exhibits a rich Boolean structure. The sampling algorithm of DC might struggle to converge in presence of such rich Boolean structures.

An other probabilistic logic programming language with continuous random variables is *Extended PRISM* [Islam et al., 2012]. Extended PRISM is again based on the distribution semantics and restricts the language such that exact probabilistic inference can be performed using normal and gamma distributions. Extended PRISM can be regarded as logic programming+computer algebra system, which means that is is closely related to DC-ProbLog using Symbo as its inference algorithm. A main difference being that Extended PRISM, as it is based on PRISM, assumes that proofs of logic formulas are mutually exclusive, therewith avoiding the disjoint sum problem.

Notable in the domain of probabilistic logic programming is also the BLOG language [Milch et al., 2005; Wu et al., 2018]. Contrary to the aforementioned probabilistic logic programming languages, BLOG's semantics is not specified using Sato's distribution semantics but uses so-called *measure-theoretic Bayesian networks*. BLOG's default inference algorithm is similar to the one of Distributional Clauses (importance sampling combined with likelihood weighting) but does also provide alternatives, such as Metropolis-Hastings MCMC [Milch; Russell, 2006].

While, all probabilistic programming languages discussed so far in this section adhere to the paradigm of logic programming, most probabilistic programming languages are actually not logic-based. Many follow the functional programming paradigm [Goodman et al., 2008; Wood et al., 2014] or are of an imperative nature [Gehr et al., 2016; Salvatier et al., 2016; Carpenter et al., 2017; Bingham et al., 2018; Ge et al., 2018]. Generally speaking, functional and imperative probabilistic programming languages target first and foremost continuous random variables, and discrete random variables are only added as an afterthought. A notable exception is the imperative probabilistic programming language Dice [Holtzen et al., 2020], which targets discrete random variables exclusively.

Lastly, we would like to point a key feature of DC-ProbLog that sets it apart from any other language incorporating discrete and continuous random variables. But first, let us briefly talk about computing probabilities in probabilistic programming. Roughly speaking, probabilities are computed summing and multiplying weights. These can for example be floating point numbers or symbolic expressions. The collection of all operations that were performed to obtain a probability is called the computation graph of a probabilistic program. Now, the big difference between DC-ProbLog and other languages lies in the structure of the computation graph. DC-ProbLog represents the computation graph as a directed acyclic graph (DAG), while all other languages, with the exception of some purely discrete languages [Fierens et al., 2015; Holtzen et al., 2020], use a tree representation. DC-ProbLog is the first language in the discrete-continuous domain to use DAGs! In cases where the computation graph can be represented as a DAG the size of the representation might be exponentially smaller compared to tree representations. Smaller computation graphs can also lead to faster inference times and lower error rates for approximate solvers.

Conclusions

In this chapter we presented DC-ProbLog, a strict extension of the probabilistic logic programming language (and SRL system) ProbLog. DC-ProbLog handles in addition to discrete/Boolean random variables also continuous random variables. In comparison to other probabilistic logic programming languages that are based on the distribution semantics [Gutmann et al., 2010; Nitti et al., 2016a; Speichert; Belle, 2019], DC-ProbLog's syntax eliminates bloated language constructs by relying on the newly introduced type system, leading to a more concise and intuitive syntax. To the best of our knowledge, DC-ProbLog's type system is the first type system that borrows ideas from multiple dispatching and applies them to logic programming. In this context it would be interesting to further investigate type systems that have already been developed for logic programming [Schrijvers et al., 2008] and study their usefulness for probabilistic logic programing.

Moreover, we exhibited an implementation of DC-ProbLog whose inference algorithms are based on weighted model integration, which we presented in Chapter 3. This constitutes an elegant analogue to the reduction of probabilistic inference in ProbLog to weighted model counting. In order to correctly handle zero probability event we did also develop the algebraic likelihood weighting inference algorithm. This answers the second research question of the thesis.

RQ2: Can we develop a high-level probabilistic logic programming language for which inference reduces to weighted model integration?

Immediate future work will have to focus on specifying the semantics DC-ProbLog in more detail and formality. Investigating the semantics in the long run could consist of exploring links of the distribution semantics in DC-ProbLog to the denotational semantics of functional or imperative probabilistic programming languages [Staton et al., 2016; Heunen et al., 2017; Holtzen et al., 2020], or links to the semantics of the probabilistic logic programming lanugage BLOG, which is not based on the distribution semantics [Wu et al., 2018].

From the viewpoint of a practitioner the expressive power of a language is more important than the fine details of the semantics. At the moment, DC-ProbLog does not natively support multivariate probability distributions and adding them would indeed tremendously increase DC-ProbLog's expressive power. Including support for multivariate distributions is only one possible such extension.

Probabilistic Perceptual Anchoring

Introduction

Example. *Consider the classical shell game where a ball is hidden under one of three identical cups. The performer of the game rapidly moves the cups and the task of the observer is to follow the movement of the cups and to identify under which cup the ball is located. For an observer to successfully identify the right cup, they must successfully handle a number of subtasks. First, despite that each of the cups are visually similar, the observer must create an individual notion of each cup as a unique object so that it can be identified (e.g., "the cup in the middle"). Likewise, the observer must recognize the ball as a unique object. Secondly, even though the ball is hidden under one of the cups, the observer makes the assumption that although the ball is not perceived it should still be present under the cup. Third, as the performer rapidly moves the cups, the observer should track the cup under which the ball is hidden. And finally, the observer also needs to realize that cups can contain balls, and therefore as the cup moves, so does the ball. Depending on the level of skill of the performer (and perhaps some additional tricks) the shell game can be a difficult one to solve.*

How could an autonomous agent handle this task as the observer? For this to be achieved, autonomous agents in real-world scenarios need to maintain a consonance between the perceived world (through sensory capabilities) and their internal representation of the world. One way to contend with this challenge is *perceptual anchoring*. Perceptual anchoring, by definition, handles the problem of creating and maintaining, in time and space, the correspondence between symbols and sensor data that refer to the same physical object in the external world.

In this third part of the thesis we study the combination of perceptual anchoring and probabilistic inference, which results in an anchoring system capable of reasoning about the world. This will answer our third research question.

RQ3: Can we equip a cognitive robotics system with probabilistic reasoning capacities?

In Chapter 8 we first propose an architecture to combine perceptual anchoring and probabilistic inference based on the following paper [6]:

Persson, Andreas; Zuidberg Dos Martires, Pedro; Loutfi, Amy; De Raedt, Luc [2020b]. Semantic Relational Object Tracking. In: *IEEE Transactions on Cognitive and Developmental Systems* 12.1, pp. 84–97.

In Chapter 9, we further develop the proposed architecture such that it can handle multi-modal probability distributions and combine the anchoring system with statistical relational learning methods. Chapter 9 is based on the paper below[7].

Zuidberg Dos Martires, Pedro; Kumar, Nitesh; Persson, Andreas; Loutfi, Amy; De Raedt, Luc [2020b]. Symbolic Learning and Reasoning with Noisy Data for Probabilistic Anchoring. In: *Frontiers in Robotics and AI* 7, p. 100.

The anchoring system developed in the previous two publications is also presented in a demo paper:

Persson, Andreas; Zuidberg Dos Martires, Pedro; De Raedt, Luc; Loutfi, Amy [2020a]. ProbAnch: a Modular Probabilistic Anchoring Framework. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

[6] Andreas Persson and I share first authorship. Andreas Persson was responsible for the design and development of the perceptual anchoring framework, the definition of the proposed anchoring matching function, and the collection of data and the evaluation of the proposed anchoring procedure. My contributions are, jointly with Andreas Persson, the development of the architecture that combines perceptual anchoring with probabilistic reasoning and the implementation thereof. Furthermore, I carried out the experimental evaluation of the combined system. Jointly with the other authors I contributed to developing the problem setup writing the text.

[7] Together with Andreas Persson, I outlined the extension of the anchoring framework to include probabilistic properties and multi-modal states. Nitesh Kumar and I integrated SRL with perceptual anchoring. Nitesh Kumar, Andreas Persson and I performed the experimental evaluation. With the other co-authors I, furthermore contributed to the development of the notions and ideas in the paper as well as the writing of the text.

Chapter 7

Background

In this section, we outline the general background of our used methods. We will both present the traditional definition of *perceptual anchoring*, an overview of *Dynamic Distributional Clauses*, as well as a brief overview of the literature on *object occlusions*.

7.1 Perceptual Anchoring

Perceptual anchoring, originally introduced by [Coradeschi; Saffiotti, 2000; Coradeschi; Saffiotti, 2001], addresses a subset of the symbol grounding problem in robotics and intelligent systems. The notion of perceptual anchoring has been extended and refined since its first definition. Some notable refinements include the integration of *conceptual spaces* [Chella et al., 2004], the addition of *bottom-up anchoring* [Loutfi et al., 2005], extensions for *multi-agent systems* [LeBlanc; Saffiotti, 2008], considerations for non-traditional sensing modalities and *knowledge based anchoring* given full scale knowledge representation and reasoning systems [Loutfi, 2006; Loutfi; Coradeschi, 2006; Loutfi et al., 2008], and *perception and probabilistic anchoring* [Blodow et al., 2010]. All these approaches to perceptual anchoring share, however, a number of common ingredients from [Coradeschi; Saffiotti, 2000; Coradeschi; Saffiotti, 2001], including:

- A *symbolic system* (including: a set $\mathcal{X} = \{x_1, x_2, \dots\}$ of *individual symbols*; a set $\mathcal{P} = \{p_1, p_2, \dots\}$ of *predicate symbols*).
- A *perceptual system* (including: a set $\Pi = \{\pi_1, \pi_2, \dots\}$ of *percepts*; a set $\Phi = \{\phi_1, \phi_2, \dots\}$ of *attributes* with values in the domain $D(\phi_i)$).

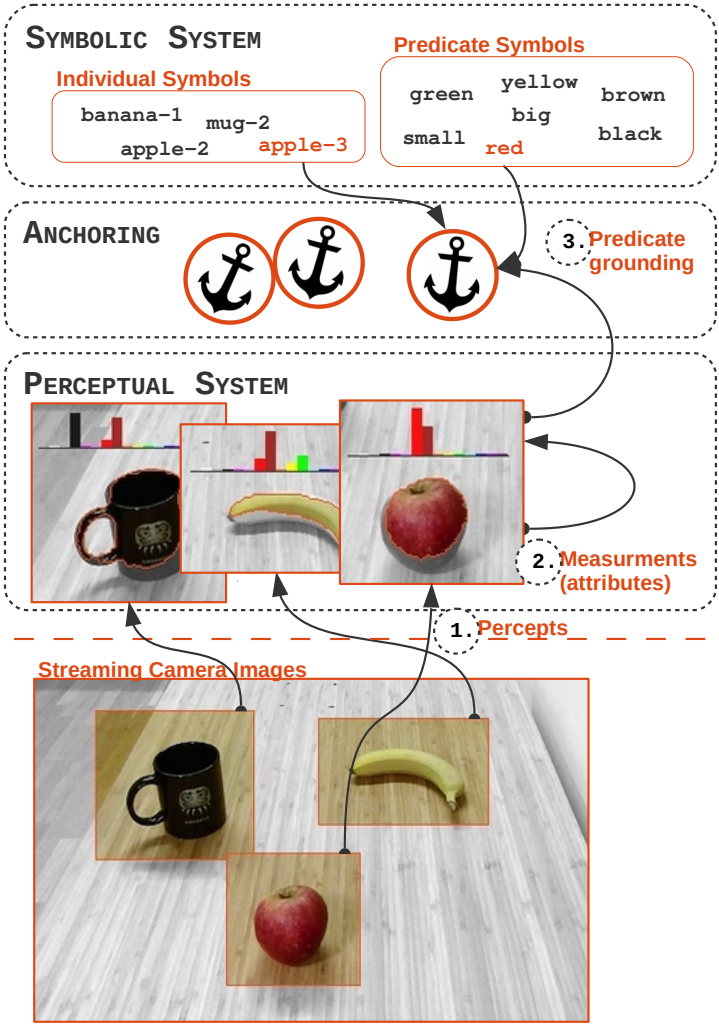


Figure 7.1: A graphical illustration of the anchoring components and their interconnections. Illustrated components are further exemplified in Example 7.1.

- *Predicate grounding relations* $g \subseteq \mathcal{P} \times \Phi \times D(\Phi)$ that encode the correspondence between unary predicates and values of measurable attributes (i.e., the relation g maps a certain predicate to compatible attribute values).

Example 7.1. *Consider the captured camera image with segmented image regions, seen in Figure 7.1. Each segmented region corresponds to an individual percept captured by the perceptual system, see Figure 7.1 – № 1. We denote the percepts* π_1, π_2 *and* π_3*, which corresponds to observed physical objects banana, apple and mug, respectively. Subsequently, a number of attributes is measured, e.g. color, size, etc. One such attribute is a color attribute measured as a normalized color histogram over the masked area of percept* π_2*, illustrated in Figure 7.1 – № 2. For clarity, we denote the measured color attribute as attribute* ϕ_2^{color}*, which have values in a domain that is equal to the n number of histogram bins. Finally, the* predicate grounding relation *g, illustrated in Figure 7.1 – № 3, for the aforementioned color attribute can be seen as the encoded correspondence between specific peeks in the color histogram and certain predicate symbols, e.g.:*

$$g(\texttt{red}, color, \arg\max_{i=1\ldots n}(\phi_{2,i}^{color})) \textit{ iff } i = 6$$

While the traditional definition of in [Coradeschi; Saffiotti, 2000; Coradeschi; Saffiotti, 2001] assumes *unary* encoded perceptual-symbol correspondences, this does not support the maintenance of anchors with different attribute values at different times. To address this problem, Persson et al. [2017] distinguish two different types of attributes:

- *Static attributes* ϕ, which are unary within the anchor according to the traditional definition.
- *Volatile attributes* ϕ_t, which are individually indexed by time t, which are maintained in a set of attribute instances φ, such that $\phi_t \in \varphi$.

Without loss of generality, we assume from here on that all *attributes stored in an anchor* are volatile, i.e., that they are indexed by a time step t. Static attributes are trivially converted to volatile attributes by giving them the same attribute value in each time step.

An anchor is, consequently, an internal data structure α_t^x, indexed by time t and identified by a unique individual symbol x (e.g., `mug-4`, `apple-2`, etc.), which encapsulates and maintains the correspondences between percepts and symbols that refer to the same physical object. Following the definition presented in [Loutfi et al., 2005], the principle functionalities to create and maintain anchors in a bottom-up fashion, i.e., functionalities triggered by a perceptual event, are:

- *Acquire* – initiates a new anchor whenever a candidate object is received that does not match any existing anchor α^x. This functionality defines a structure α_t^x, index by time t and identified by a unique identifier x, which encapsulates and stores all perceptual and symbolic data of the candidate object.

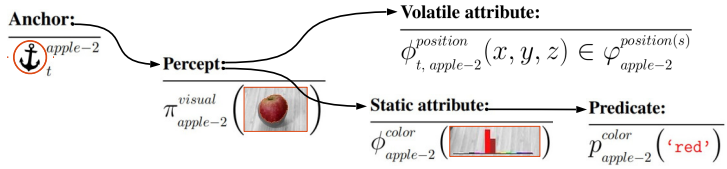


Figure 7.2: A conceptual illustration of the internal data structure that constitutes a single anchor, and which is first initiated by a percept π from a raw image. The volatile and static attributes are derived from this percept, while predicates such as `red`, are derived from static attributes (which are not indexed by time), e.g. the static color histogram attribute.

- *Re-acquire* – extends the definition of a matching anchor α^x from time $t-k$ to time t. This functionality assures that the percepts pointed to by the anchor are the most recent and adequate perceptual representation of the object.

Based on the functionalities above, it is evident that an *anchoring matching function* is essential to decide whether a candidate object matches an existing anchor or not.

7.2 Dynamic Distributional Clauses

Dynamic Distributional Clauses (DDC) is probabilistic logic programming language; an extension of the logic programming language Prolog [Sterling; Shapiro, 1994][1]. DDC is capable of representing discrete and continuous random variables and to perform probabilistic inference. Moreover, DDC explicitly models time, which makes it predestined to model dynamic systems. The underpinning concepts of DDC are related to ideas presented in [Milch et al., 2005] but embedded in logic programming. Programs written in DDC allow for high-level (discrete variables) reasoning with low-level sensor input (continuous variables).

In logic programming, reasoning happens through the usage of symbols. These are either terms or predicates. The latter are often referred to as relations. Terms can be constants, logical variables or n-ary functors applied to an n-tuple of terms. Constants can only have one assigned value to them, which means that only one interpretation is possible. This is in contrast to logical variables, which have multiple interpretations. More concretely, a logical variable `X` is a variable ranging over the set of all possible ground terms. Lastly, we also have terms of the form $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$ with $\mathtt{p}/n$ being an

[1] A comprehensive treatise on the field of probabilistic logic programming can be found in [De Raedt; Kimmig, 2015] and [Riguzzi, 2018].

n-ary predicate and all $\mathtt{t}_i$'s being terms themselves. This last kind of terms are dubbed atoms.

In the static case, i.e., when there is no explicit dependency on time in the terms, DDC programs reduce to *Distributional Clauses* (DC) [Gutmann et al., 2011; Nitti et al., 2013] programs. A distributional clause is of the form $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b}_1, \ldots, \mathtt{b}_n$, where $\sim$ is a predicate in infix notation and $\mathtt{b}_i$'s are literals, i.e., atoms or their negation. $\mathtt{h}$ is the name of a random variable and $\mathcal{D}$ tells us how the random variable is distributed – both are formally terms. The meaning of such a clause is that each grounded instance of a clause $(\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b}_1, \ldots, \mathtt{b}_n)\theta$ defines a random variable $\mathtt{h}\theta$ that is distributed according to $\mathcal{D}\theta$. A grounding substitution $\theta = \{\mathtt{V}_1/\mathtt{t}_1, \ldots, \mathtt{V}_n/\mathtt{t}_n\}$ is a transformation that simultaneously substitutes all logic variables $\mathtt{V}_i$ in a distributional clause with non-variable terms $\mathtt{t}_i$. Here we see that random variables and distributions are themselves not necessarily grounded by definition. The mean of a normal distribution can, for instance, depend on random variables. For the atom $\mathtt{h}\theta$ to be defined it is necessary that all atoms $\mathtt{b}_i\theta$ in the distributional clause evaluate to true. Labeling a distributional clause with time indexes allows for declaring dynamic models via defining a transition model from time step $\mathtt{t}$ to time step $\mathtt{t}+1$. DDC can be viewed as a language that defines conditional probabilities for discrete and continuous random variables: $p(\mathtt{h}\theta|\mathtt{b}_1\theta, \ldots, \mathtt{b}_n\theta) = \mathcal{D}\theta$.

Example 7.2. *Consider the following DDC program:*

```
n ~ poisson(6).
pos(P):0 ~ uniform(0,100) ← n~=N, between(1,N,P).
pos(P):t+1 ~ gaussian(X+3, Σ) ← pos(P):t~=X.
left(O1,O2):t ~ finite([0.99:true, 0.01:false]) ←
        pos(O1):t~=P1, pos(O2):t~=P2, P1<P2.
```

The first rule states that the number of objects n in the world is distributed according to a Poisson distribution with mean 6*. The second rule states that the position of the n objects, which are identified by a number P between 1 and n, are distributed according to a uniform distribution between 0 and 100. Here, the notation* `n~=N` *means that the logical variable N takes the value of our random variable n. The label* 0 *(resp. t) in the program denotes the point in time. So,* `pos(P):0` *denotes the position of object P at time 0. Next, the program describes how the position evolves over time: at each time step the object moves three units of length, giving it a velocity of* 3 [*length*]/[*time*]*. Finally, the example program defines the* `left` *predicate, through which a relationship between each object is introduced at each time step. DDC then allows for querying this program through its builtin predicate:*

```
query((left(1,2):t~=true, pos(1):t>0), Probability)
```

`Probability` *in the second argument unifies with the probability of object* 1 *being to the left of object* 2 *and having a positive coordinate position.*

Example 7.3. *One possible world of the uncountably many possible worlds encoded by the program in Example 7.2. The sampled number* n *determines that* 2 *objects exists, for which the ensuing distributional clauses then generate a position and the* left /2 *relationship:*

```
n ~= 2.
pos(1):t ~= 30.5. pos(2):t ~= 63.2.
pos(1):t+1 ~= 32.4. pos(2):t+1 ~= 58.8.
left(1,2):t ~= true. left(2,1):t ~= false.
```

When performing inference within a specific time step, DDC deploys importance sampling combined with backward reasoning (SLD-resolution), likelihood weighting and Rao-Blackwellization [Nitti et al., 2016a]. Inferring probabilities in the next time given the previous time step is achieved through particle filtering [Nitti et al., 2013]. If the DDC program does not contain any predicates labelled with a time index the program represents a *Distributional Clauses* (DC) [Gutmann et al., 2011] program, where filtering over time steps is not necessary.

7.3 Occlusions

Object occlusion is a challenging problem in visual tracking and a plethora of different approaches exist that tackle different kinds of occlusions; a thorough review of the field is given in [Meshgi; Ishii, 2015]. The authors use three different attributes of an occlusion to categorize it: the *extent* (partial or full occlusion), the *duration* (short or long), and the *complexity* (simple or complex)[2]. Another classification of occlusions separates occlusions into *dynamic occlusions*, where objects in the foreground occlude each other, and *scene occlusions*, where objects in the background model are located closer to the camera and occlude target objects by being moved between the camera and the target objects[3].

Meshgi; Ishii [2015] report that the majority of research on occlusions in visual tracking has been done on partial, temporal and simple occlusions. Furthermore, they report that none of the approaches examined in the comparative studies of Smeulders et al. [2013] and Wu et al. [2013], handle either partial complex occlusions or full long complex occlusions.

In order to handle full, long and complex occlusions [Nitti et al., 2013] used a model-based object tracking approach. They describe possible occlusions through a *theory of*

[2] An occlusion of an object is deemed complex if during the occlusion the occluded object considerably changes one of its key characteristics, e.g. position, color, size). An occlusion is simple if it is not complex.

[3] Further categories exist, we refer the reader to [Vezzani et al., 2011; Meshgi; Ishii, 2015].

occlusions (ToO)[4] expressed as dynamic distributional clauses. Declaring such a ToO allowed the authors to perform *occlusion reasoning* and to track objects not by directly observing them but by reasoning about relationships that occluded objects had entered with detected objects (see Example 1.5).

Example 7.4. *An excerpt from the set of clauses that constitute a ToO. The example clause describes the conditions under which an object is considered a potential* Occluder *of an other object* Occluded*.*

```
occluder(Occluded,Occluder):t+1~finite(1.0:true) ←
        observed(Occluded):t,
        \+observed(Occluded):t+1,
        position(Occluded):t ~= (X,Y,Z),
        position(Occluder):t+1 ~= (XH,YH,ZH),
       D is sqrt((X–XH)^2+(Y–YH)^2), Z<ZH, D<0.3.
```

Out of all the potential Occluder*s the actual occluding object is then sampled uniformly:*

```
occluded_by(Occluded,Occluder):t+1 ←
        sample_occluder(Occluded):t+1 ~= Occluder.
sample_occluder(Occluded):t+1~uniform(ListOccluders) ←
        findall(O,occluder(Occluded, O):t+1,
            ListOccluders).
```

A limitation of Nitti et al.'s approach is them assuming the data association problem to be solved (by using AR tags), i.e. each object is unambiguously identified through a easily visible AR tag. We will show how to tackle the problem of handling full, long and complex occlusions while at the same time tackling the data association problem (by means of perceptual anchoring).

[4] The term theory of occlusion was coined in [Zuidberg Dos Martires et al., 2020b].

Chapter 8

Semantic World Modeling*

This chapter addresses the topic of semantic world modeling by conjoining probabilistic reasoning and object anchoring. The proposed approach uses a so-called bottom-up object anchoring method that relies on rich continuous attribute values measured from perceptual sensor data. A novel anchoring matching function learns to maintain object entities in space and time and is validated using a large set of humanly annotated ground truth data of real-world objects. For more complex scenarios, a high-level probabilistic relational object tracker has been integrated with the anchoring framework and handles the tracking of occluded objects via reasoning about the state of unobserved objects.

8.1 Introduction

Given the definition of anchoring in Section 7.1, it is evident that an initial matching function is essential to determine if a candidate object is matching an existing anchor or not. In prior work on perceptual anchoring, the problem of matching anchors has mostly been addressed through a simplified approach based on the use of symbolic values (or left out entirely), where the predicate grounding relation mapping between symbolic predicate values and measured attribute values commonly is facilitated by the use of conceptual spaces [Chella et al., 2004]. Conceptual spaces can be thought of as a discretization of the continuous perceptual attribute space, which consequently also result in a loss of information. The procedure of creating and maintaining anchors, based on the discretized symbolic values, must accordingly either be handled by a probabilistic system, as in the case of [Elfring et al., 2013], or through the use of additional knowledge and the use of a reasoning system, as in the case of [Daoutis

* This chapter is based on [Persson et al., 2020b].

et al., 2012]. In this paper, we move the matching function down to the perceptual level by presenting a novel matching approach that facilitates the rich information found within the measured attribute values.

Moving the anchoring matching function to the perceptual level will, inevitably, also introduce another level of complexity, since measured attributes must be compared based on continuous attribute values. The system must, subsequently, both recognize previously observed objects and detect (or anchor) new and previously unknown objects based on the result of the initial matching function. In the case of open-world scenarios without a fixed number of possible objects classes that the system might encounter, this is undoubtedly a challenging issue. In this chapter, we address this issue in the context of bottom-up perceptual anchoring, and present an evaluation that addresses the problem of learning how to determine if an object has previously been perceived (or not).

We present an approach to create a semantic world model where object entities are maintained and tracked over time. We further integrate a probabilistic reasoning component which is able to support the tracking of objects in case of occlusions due to limitation in perception or due to interactions with other objects. Our approach is based on perceptual anchoring. Perceptual anchoring, by definition, handles the problem to create and maintain, in time and space, the correspondence between symbols and sensor data that refer to the same physical object in the external world [Coradeschi; Saffiotti, 2000]. This problem has, subsequently, been defined as the *anchoring problem* [Coradeschi; Saffiotti, 2003]. We use *bottom-up* anchoring [Loutfi et al., 2005], whereby anchors (object representations) can be created by perceptual observations derived from interactions with the environment. For a practicable bottom-up anchoring system it is essential to have a robust anchoring matching function that accurately matches perceptual observations against the perceptual data of previously maintained anchors. For this purpose, we introduce a novel method that replaces a traditionally hand-coded anchoring matching function by a learned model (cf. [Persson et al., 2017]).

Apart from the two anchoring functionalities in Section 7.1 (acquire and re-acquire), there has also been a third anchoring functionality; a *track* functionality [1]. This track functionality has recently been revised and explored in the interest of integrating object tracking into the concept of anchoring, introduced in [Persson et al., 2017], which suggested a tracking functionality highly integrated with a point cloud-based particle filter tracking approach on the lowest perceptual sensory level [Rusu; Cousins, 2011]. However, the performance of the previously suggested framework was heavily affected by the computational load of the used object tracking approach, which requires tracking of computational demanding 3-*D* point cloud data. In this paper, we further

[1] The *track* functionality was formally integrated with the *re-acquire* functionality for the extension to sensor-driven bottom-up anchoring [Loutfi et al., 2005], such that no distinction was made between extending the definition for an anchor from time $t-1$ or $t-k$.

explore the integration between object tracking and perceptual anchoring, presented in Section 8.2.4. This integration is based upon the belief that the integration should be loosely coupled in order to sustain the benefits of both the ability to maintaining individual instances on objects at a larger scale, as in the case of perceptual anchoring, as well as efficiently and logically track object instances over time, as in the case of probabilistic reasoning. In particular, we utilize Dynamic Distributed Clauses [Nitti et al., 2016a] to facilitate reasoning about object entities at a symbolic level. DDC can handle continuous random variables in addition to discrete ones, which predestines DDC to be utilized for reasoning within robotics, where the world is inherently continuous and uncertain. Integrating a reasoning system into the anchoring framework allows us to dynamically feed back information to the anchoring system and update the retained semantic world model with information stemming from our probabilistic model of the world.

8.2 Anchoring + Inference

Our suggested combined framework architecture, seen in Figure 8.1, is a modularized architecture that utilizes libraries and communication protocols available in the Robot Operating System (ROS)[2]. Each module/sub-system (described in detail below) has a dedicated task, while the overall goal of the combined framework is to create and maintain coherent and accurate representations (anchors) of perceived real-world objects. The same anchor representations also provides the historical background information, i.e., information about objects last perceived at time $t - k_x$, which is used by the *anchoring system* to process and anchor objects perceived at the present time t, illustrated in Figure 8.1–(2.). Furthermore, the framework utilizes an *inference system*, illustrated in Figure 8.1–(3.), which aids the anchor system in complex dynamic scenes. Finally, the architecture is a sensor-driven architecture that is triggered by perceptual data, i.e. sensor readings, which initially are pre-processed by a *perceptual pipeline*, illustrated in Figure 8.1–(1.).

8.2.1 Implementation Details: Pre-processing Pipeline

The overall background *pre-processing pipeline*, with the goal of detecting and segmenting objects, extract features from detected objects, classifying and symbolically grounding each object instances, is illustrated in Figure 8.1 – (1.). For this purpose, our system setup relies upon publicly available core libraries and systems, including: the

[2]http://www.ros.org/

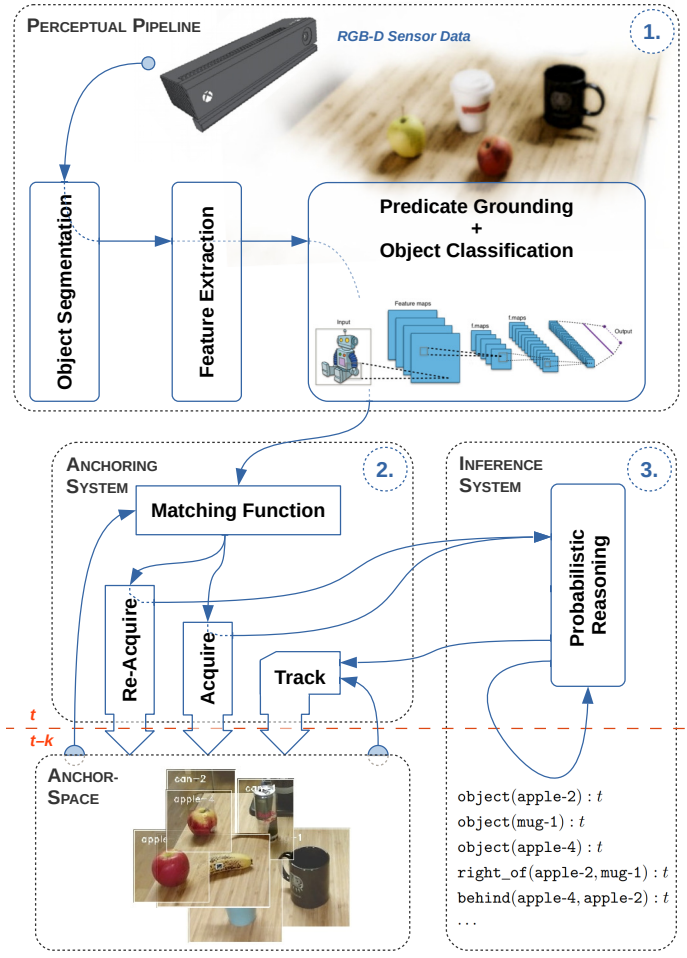


Figure 8.1: Overview of our combined framework architecture. The overall framework is modularized architecture that consist of three core sub-systems (each described in further details in this section): *1)* an initial *perceptual pre-processing pipeline* with the purpose of detecting, segmenting and processing perceived objects, *2)* an *anchoring system* with the purpose of creating and maintaining updated and consistent representations (anchors) of perceived objects, and *3)* an *inference system* with the purpose aiding the anchor system and track objects in complex dynamic scenarios.

Point Cloud Library[3] (PCL), the Open Computer Vision library[4] (OpenCV), and the

[3]http://pointclouds.org/
[4]http://opencv.org/

Robot Operating System (ROS). It should also be noted, all methods and techniques covered in this section are considered to be replaceable *black-box* approaches that are used for the means of providing background for the subsequent theoretical Section 8.2.2. For example, the used object segmentation method could be replaced with a convolutional network-based semantic segmentation approach [Long et al., 2015]. This requires, however, an adequate dataset of pixel-wise mask annotations for training the network to detect the objects of interest, which is something that is not always publicly available and must, therefore, further be addressed, e.g., through weakly supervised learning [Khoreva et al., 2017]. Nevertheless, the details on used techniques, for the presented architecture, are here covered for completeness and reproducibility.

More specifically, the initial step of our pre-processing pipeline is an *object segmentation* method, which is performed with the purpose of detecting arbitrary objects of interest in the scene. The deployed object segmentation method is based on organized point cloud data (i.e., the organization of point cloud data is identical to the rows and columns of the imagery data from which the point cloud originates), which are given as input data by a Kinect2 RGB-D sensor. To establish the connection between the ROS environment and the Kinect2 sensor, we integrated the ROS-Kinect2 bridge [Wiedemeyer, 2015] in the presented framework architecture. The segmentation procedure can briefly be described using the following steps:

- Estimate 3-*D* surface normals based on integral images [Holzer et al., 2012]. This function uses the algorithm for calculating average 3-*D* gradients over six integral images, where the horizontal and vertical 3-*D* gradients are used to compute the normal as the cross-product between two gradients.
- Planar segmentation based on the calculated surface normals.
- Object segmentation through clustering of the remaining points (points that are not part of the detected planar surfaces). This segmentation uses a connected component segmentation, presented in [Trevor et al., 2013], where a Euclidean comparison function is used to connect the components that constitute the cloud cluster of an individual object.

Moreover, provided that object instances have been segmented based on the full spectrum of available RGB-D data (as described above), we are further able to exploit the advancements in deep learning for *image classification* in the final *object classification* procedure of our pre-processing pipeline. The *Convolutional Neural Networks* (CNN) architecture used in this case is based on the 1 K *GoogLeNet* model, developed by Szegedy et al. [2015], and which originally was trained on the ILSVRC 2012 visual challenge dataset [Russakovsky et al., 2015]. For this work, we have, however, extracted a subset of the *ImageNet* database [Deng et al., 2009] and fine-tuned

the model for 101 objects categories that are relevant for a household domain, e.g., `mug`, `spoon`, `banana`, `tomato`, etc., where the model was trained for a *top-1* accuracy of 73.4% (and a *top-5* accuracy of 92.0%).

8.2.2 Theoretical Aspects: Precepts, Attributes and Symbols

The resulting output of the object segmentation is m point cloud clusters (where m varies between frames). For consistency with the definition of anchoring, we denote segmented clusters as percepts: $\{\pi_1^{spatial}, \pi_2^{spatial}, \dots \pi_m^{spatial}\}$, which each corresponds to the spatial 3-D point cloud data of an individual object. Subsequent to the segmentation of the point cloud clusters, the same RGB-D data is also used for segmenting corresponding visual 2-D imagery data of each detected object. This image segmentation is entirely based on the prior point cloud clusters and a projection between the 3-D point cloud frame and the 2-D visual RGB frame of the RGB-D sensor. Also, we denote visual data as percepts: $\{\pi_1^{visual}, \pi_2^{visual}, \dots \pi_m^{visual}\}$, which each corresponds to the visual 2-D imagery data of a segmented object.

Next, both segmented perceptual 3-D point cloud data and 2-D visual data are further forwarded to a *feature extraction* procedure. The first step of this feature extraction procedure is to extract both a *position attribute* as the point at the geometrical center of each segmented percept $\pi_y^{spatial}$, and a *size attribute* as the 3-D bounding box around each percept $\pi_y^{spatial}$, where $\pi_{y|y=1,2\dots m}^{spatial} \in \{\pi_1^{spatial}, \pi_2^{spatial}, \dots \pi_m^{spatial}\}$. The extracted *position attribute* is here denoted $\phi_y^{pos} \in \mathbb{R}^3$, while the corresponding *size attribute* is denoted $\phi_y^{size} \in \mathbb{R}^3$. Furthermore, a *color attribute* ϕ_y^{color} is extracted for each visual percept π_y^{visual}, which is measured as a color histogram (in the HSV color space).

Finally, extracted attributes together with the perceptual data are further forwarded to a combined *predicate grounding* and *object classification* procedure. This procedure has the purpose of both grounding and associating a symbolic value with each extracted attribute, as well as classifying each object and further associating each object with an object category label. The predicate grounder is here responsible for grounding each measured attribute ϕ_y (of the set Φ_y, that originates from the same physical object) to a predicate grounding symbol p_y. For example, a certain peek in a color histogram, measured as a ϕ_y^{color} attribute, is grounded to the symbol `red`, such that p_y^{color}=`red` (cf. Example 7.1). In the context of anchoring, we further assume that all trained object categories (e.g., `mug`, `spoon`, `tomato`, etc.) of used GoogLeNet model are part of the set of possible predicate symbols $\mathcal{P}$. The input for the object classification procedure are, subsequently, the segmented visual percepts π_y^{visual}, while resulting object categories together with predicted category probabilities are denoted by $p_y^{class} \in \mathcal{P}$ and ϕ_y^{class}, respectively.

8.2.3 Anchoring Management

The entry point for the *anchoring system*, seen in Figure 8.1–(2.), is a *matching function*. This function assumes a bottom-up approach to perceptual anchoring, described in [Loutfi et al., 2005], where the system constantly receives candidate objects and invokes a number of different matching algorithms (one matching algorithm for each measured attribute in the set $\Phi_y = \{\phi_y^{class}, \phi_y^{color}, \phi_y^{size}, \phi_y^{pos}\}$) in order to determine if an anchor, α^x, has previous been perceived or not.

Matching Function

More specifically, an unknown set of attributes Φ_y is compared against the set of attributes Φ_x of an existing anchor α^x. The combined result of all individual invoked matching algorithm determines, subsequently, if an anchored object has previously been perceived or not. In details, a classification attribute ϕ_y^{class} and symbol p_y^{class} of a candidate object is firstly compared against the classification attribute and symbol of a previously stored anchor according to:

$$d_{x,y}^{class}(\phi_x^{class}, \phi_y^{class}) = \begin{cases} \exp\left(-\frac{|\phi_x^{class} - \phi_y^{class}|}{\phi_x^{class} + \phi_y^{class}}\right) & \textbf{if } p_x^{class} \equiv p_y^{class} \\ 0 & \textbf{else} \end{cases} \tag{8.1}$$

We interpret the $d_{x,y}^{class}$ as the exponentially decaying relative L^1-distance between the two attribute values ϕ_x^{class} and ϕ_y^{class}. This means that we exponentially penalize the distance between two objects in the class attribute space.

Secondly, the color histogram of a color attribute ϕ_y^{color} of a candidate object is compared (assuming normalized color histograms) according to the *color correlation*:

$$d_{x,y}^{color}(\phi_x^{color}, \phi_y^{color}) = \frac{1}{2} + \frac{\sum_{i=1}^{n}(\phi_{x,i}^{color} - \mu_x)(\phi_{y,i}^{color} - \mu_y)}{2\sqrt{\sum_{i=1}^{n}(\phi_{x,i}^{color} - \mu_x)^2 \sum_{i=1}^{n}(\phi_{y,i}^{color} - \mu_y)^2}} \tag{8.2}$$

Where n is the number of histogram bins, the index i gives the i-th histogram bin value of ϕ_x^{color} and ϕ_y^{color} (respectively), and μ_x and μ_y are the *color mean value* of each histogram, given according to:

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n}\phi_{x,i}^{color}, \qquad \mu_y = \frac{1}{n}\sum_{i=1}^{n}\phi_{y,i}^{color}$$

Next, the distance between a position attribute ϕ_y^{pos} and the position ϕ_x^{pos} of a previously stored anchor α^x, is calculated according to the L^2-distance (in 3-D spatial space). Inspired by the work presented by Blodow et al. [2010], this distance is then mapped to a *normalized similarity distance* according to:

$$d_{x,y}^{pos}(\phi_y^{pos}, \phi_x^{pos}) = e^{-L^2(\phi_y^{pos}, \phi_x^{pos})} \tag{8.3}$$

Furthermore, the size attribute ϕ_y^{size} of a candidate object is compared according to the *generalized Jaccard similarity* (for the bounding boxes in 3-D space):

$$d_{x,y}^{size}(\phi_x^{size}, \phi_y^{size}) = \frac{\sum_{i=1}^{3} min(\phi_{x,i}^{size}, \phi_{y,i}^{size})}{\sum_{i=1}^{3} max(\phi_{x,i}^{size}, \phi_{y,i}^{size})} \tag{8.4}$$

Motivated by the importance of the time within the concept of anchoring, the difference in time since last recorded observation of a previously stored anchor α^x, defined at $t - k_x$, is finally mapped to a similar normalized distance according to:

$$d_{x,y}^{time}(t, t-k_x) = \frac{2}{1+e^{t-(t-k_x)}} = \frac{2}{1+e^{k_x}} \tag{8.5}$$

Consequently, all given matching distance values, Equations 8.1 to 8.5, are given in the interval [0.0, 1.0], and all distance values are, therefore, also commensurable.

Creating and Maintaining Anchors

Combining all matching distance values, given by Equations 8.1 to 8.5, and determining whether a candidate anchor has previously been perceived (or not), is not a trivial task. Especially not in the context of *bottom-up* anchoring in real-world scenarios with unlimited possibilities of objects together with continuous distance values given by the initial *matching function*. The matching distance values can be combined in many different ways, e.g.: through a *min* or *max* function, by the *weighted average* with different weights, etc. Nonetheless, a threshold value is ultimately required in order to

determine if the combined result is to be considered as a match (or not). In this paper, we will shed some light upon this issue and present our work on the topic of learning the anchoring matching function, which determines if an object is a novel object or a previously observed object.

At this point we would like to stress that the architecture of the anchoring system is completely agnostic towards how the matching function was learned. This means that the anchoring system considers the matching function to be an exchangeable *black-box* approximation of the true anchor-percept matching. In Section 8.3.1 we compare different classifiers to each other that could be potentially used to approximate the matching.

Regardless of the used classification algorithm, the process of the anchoring is to ultimately create or maintain anchors through either one of the two principal functionalities: *acquire* or *re-acquire*, respectively. The *anchor-space*, in which the anchors are maintained and stored, is in this case expressed as a *permanent world model* (PWM). We further enhance the traditional *acquire* functionality by utilizing the deep learning classifier such that a unique identifier x is further generated based on the classification symbol p^{class}, e.g., for an object classified as a `cup`, a corresponding unique identifier could be generated as x = `cup-4`.

8.2.4 Integration of the Inference System

In order to prevent the curse of dimensionality from propagating from a probabilistic inference system into the perceptual anchoring system, we opt for only loosely integrating the *inference system* with the *anchoring system*. This linkage has as a consequence that we need to maintain two distinct databases for representing our belief of the world.

The above-described anchoring system database plays the role of maintaining a *permanent world model* (PWM), remembering all objects that have appeared over time ($t - k_x$). By contrast, the database of the inference system, implemented in DDC, operates on a *temporary world model* (TWM). The latter does, however, not only retain which objects are present in the scene but also how the single objects relate to each other. A `cup` might, for example, be remembered as standing at a certain point in 3-D space but also by the fact that it stands left to some other `cup`. This representation of the real world is obviously a lot more expensive but adds valuable information to the scene description when carrying out high-level object tracking. The relational nature of the TWM enables us to reason about the world and additionally to track objects on a high level.

By working with two distinct databases we take advantage of the databases for specialized tasks, e.g., reasoning with a relational database. However, it also means

that we need to maintain two distinct databases, and more importantly, the databases have to reflect the same world. Therefore, in order to retain the integrated framework in a coherent state, we need to place the loosely coupled inference system and anchoring system in a tight feedback loop. This feedback loop is represented by the incoming and outgoing arrows in panel (3.), Figure 8.1. It hence consists of two distinct steps:

1. Sending anchor information from the *anchoring system* to the *inference system* and initiating or updating the belief of the world in the *inference system*. More specifically, the anchoring system tells the inference system whether

   - a new anchor was acquired
   - an anchored was re-acquired
   - an anchor was deleted (this happens when a previously acquired anchor is classified as a glitch)

   When the anchoring system sends information about acquired and re-acquired anchors, the properties of the anchors are sent along as well (e.g. position, color).

2. Sending back the updated belief of the world in the *inference system* and update the belief of the world in the *anchoring system*, accordingly. More specifically, the inference system tells the anchoring system whether

   - an anchor that was not is being tracked by the inference system

   When the inference system sends information about tracked anchors, the properties of the (inferred) properties of the anchors are sent along as well. As of now this is only the position of the tracked anchors.

When exchanging information between the two databases it is crucial that the two databases are consistent. This means that both databases (of the anchoring system and the inference system) talk about the same objects. Anchors in the two databases might disagree on the properties of the anchors (this is actually unavoidable by design) but they must not disagree on the existence of an anchor. We ensure this by limiting the creation and deletion of anchors to the anchoring database. In the probabilistic database, anchors are only created via information obtained from the anchoring database.

The initial belief in the TWM is initiated by the belief of the PWM of the scene. The anchor information, originating from the PWM, is treated as observations in the TWM. For each initial observation, DDC clauses are added to the TWM database, which constitutes the temporary internal representation of the world. For a `cup` in the scene, for example, a rule is added that describes the initial belief of its position and velocity:

$$\texttt{pos(cup):0} \sim \texttt{normal}(\vec{R}, \vec{0}, \vec{\Sigma}) \leftarrow \texttt{obs(percept\_pos(cup)):0} \sim= \vec{R}.$$

Where $\vec{R}$ is the observed 3-*D* position of the geometric center of the spatial percept of an object (the `cup` in this case). $\vec{\Sigma}$ is the covariance matrix that specifies the Gaussian and the $\vec{0}$ corresponds to the initial velocity which is set to 0 in each dimension. For all the following time steps, we define an observation model that takes into account uncertainty in the measurement process itself. We adopt the approach of Nitti et al. [2014] of expressing the measurement model as the product of Gaussian densities around the position of each object. Assuming independently and identically distributed measurements of the objects allows for this factorization. This idealization assumes that observing an object does not depend on the observation of any other object (in the same time step). For the an object with `cup` this means:

$$\begin{aligned} &\texttt{obs(percept\_pos(cup)):t+1} \sim \texttt{normal}(\vec{R}, \Sigma_{obs}) \leftarrow \\ &\qquad \texttt{pos(cup):t} \sim= \vec{R} \end{aligned}$$

In the case that all objects in the scene are observed, i.e., none of the objects is occluded by another one, the TWM and PWM are now in a state of cognitive consonance, as we used the anchor information as observations for the inference system. If, however, objects get occluded due to manipulations of the world, the occluded objects do not produce any perceptual data anymore. Hence, the perceptual anchoring system can no longer update its belief of the world, and no updated belief is sent to the inference system. In this case, the inference system needs to reason about what might have happened to the object that is not perceived anymore by the anchoring system. Considering the world at time step $t-1$, and the observations of the world at time step t, we can speculate about the world in time step t. We infer the state of an occluded object through its relations with perceived objects in the world. This inferred updated belief of the world is then sent back to the anchoring system where the state of occluded objects is also updated.

This approach allows us to propose a modified high-level anchoring *track* functionality (cf. [Persson et al., 2017]), such that:

- *Track* – extends the definition of an anchor α^x from time $t-1$ to time t. This functionality is directly responding to the state of the probabilistic object tracker, which assures that the percepts pointed to by the anchor are the adequate perceptual representation of the object, even though the object is currently not perceived.

With the (re)introduction of the anchoring *track* functionality, we also need to ensure cognitive consonance at the anchoring side by updating the PWM based on the updated belief of the world, established by the inference system. More specifically, the 3-*D* *position* attribute ϕ_x^{pos} of an anchor α_t^x is updated according to the inferred position of the corresponding object maintained in the TWM of the inference system. This

exchange of information between both systems, as described in this section, is further facilitated by sharing the unique identifier x of an anchored object. Hence, we are able to differentiate between specific instances of objects and we can express the rules for an object instance that is identified by the unique symbol, e.g., `cup-1`, `apple-4`, etc.

The details on how the probabilistic inference is carried out are given in [Nitti et al., 2013; Nitti et al., 2016a]. Our contribution lies in coupling a probabilistic inference system with an anchoring system, which enables the conjoined system to probabilistically reason on low-level sensor data. This is for example not the case in [Nitti et al., 2014], where they used AR-tags to observe objects in the world.

8.3 Evaluation and Results

Evaluating a real-world operating anchoring framework, with several interacting components as described in Section 8.2, is undoubtedly a challenging task. Noisy sensor readings and erroneous attribute measurements are inevitably present and will propagate through the components of the processing pipeline. The evaluation presented in this section is, therefore, limited to: *1)* the performance of the suggested anchoring matching approach (presented in Section 8.3.1), and *2)* the integrated combined anchoring and reasoning system (presented in Section 8.3.2).

8.3.1 Learning the Anchoring Matching Function

The evaluation presented in this section has a two-folded purpose: *1)* collect annotated ground truth data about objects in dynamic scenarios, and *2)* learning to determine which of the the two anchoring functionalities *acquire* or *re-acquire* (cf. Section 8.2.3 and Figure 8.1–(2.), to initiate based on the matching distances values given of the initial anchoring matching function (given by Equations 8.1 to 8.5, as described in Section 8.2.3).

Data Collection

A benefit of using perceptual anchoring is that the percepts pointed to by the anchor are the most recent and adequate perceptual representation of an object. For the evaluation presented in this paper, we have exploited these updated and maintained representations, found in anchors, in order to collect human-annotated ground truth data. This data collection was conducted through a *human-annotation interface* that was queued with segmented perceptual sensor data given by the perceptual pre-processing pipeline, presented in Section 8.2.1. By utilizing this interface, all data about unknown candidate

objects, together with the perceptual data of possible matching anchored objects, could be presented and visualized for the human user. Hence, the human was able to provide feedback about the action that the human counterpart would consider as the appropriate anchoring action for each presented candidate object (i.e., *acquire* a new anchor for a queued object, or *re-acquire* an existing anchor). The procedure for collecting our ground truth data is further described and exemplified in Figure 8.2.

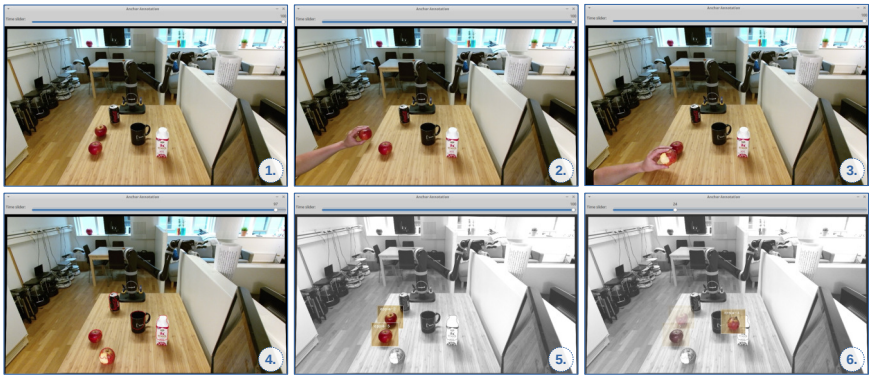


Figure 8.2: A depiction of our *human-annotation interface* that was used in order to collect ground truth data of anchored objects. In conjunction with changes in the scene, as illustrated by (1.) to (3.), the human user has the possibility to *freeze* the execution of the framework and providing feedback about what he/she would consider as the appropriate anchoring action for a candidate objects. Once the execution is frozen, the human user can select segmented candidate objects, e.g., the moved `apple` as illustrated in (4.), after which the framework is responding by displaying an updated representation of a number of already anchored objects, shown in (5.), which best (attribute-wise) corresponds to the selected object. The human user can then provide positive feedback about a matching anchored object (by selecting the representation of the matching anchored object), or negative feedback (simply by clicking anywhere else on the screen). Also, to covering the time aspect, and to suggest possible matching anchored objects that have not been perceived recently, we have further added a time slider, illustrated in the top part of (6.). Through this time slider can the user adjust the time factor k for the purpose of selecting a matching anchored object that was last observed at a time $t - k$.

Behind the scene of proposed human-annotation interface, exemplified in Figure 8.2, the data that in reality was collected and stored was matching distance values, provided by Equations 8.1 to 8.5. Together with each set of distance values (as result of comparing the attributes of an unknown candidate object against the attributes of an existing anchored object), was further an annotated label of 1 stored if the user

considered an existing object as a matching object, or 0 otherwise. Worth noting is that the collected data purely represent that the human user was considering as the most appropriate action for each presented scene. Hence, we were also able to gather samples of objects in, for example, ambiguous situations where an identical (but physically different) instance of an object was introduced while the similar counterpart was still observed. Furthermore, given such ambiguous situations with a number of possible matching anchored objects that could match a selected candidate object, as depicted in Figures 8.2–(5.) and 8.2–(6.), we assumed that there could only exist *one* true match (labeled 1), while the reaming candidates were non-matching candidates (labeled 0). As a result, we were able to collect several samples for each human action.

Experimental Evaluation

With the use of the human-annotation interface, as described in the previous section, we were able to collect a dataset of a total of 5400 samples[5]. A dataset that we, subsequently, have used for this particular evaluation in order to train the anchoring system to initiate proper anchoring functionality for different situations. During the data collection, several typical problematic anchoring scenarios, e.g., scenarios there new ambiguous objects are introduced in the scene, scenarios with partly occluded objects, scenarios where existing objects were disappearing and reappearing in the scene, etc., were executed in order to cover a broad range of different situations. Moreover, the data collection was conducted on several occasions for the purpose of capturing changes in the environmental conditions, e.g., changes in light conditions.

Given the collected data, which was comprised of sets of matching distance values together with corresponding labels (with a label of 1 for a matching set of distance values, or a label of 0 for a non-matching set), our approach for learning how to correctly anchoring objects, and thereby learn to invoke correct anchoring functionality (*acquire* or *re-acquire*), was through the evaluation of different classification algorithms. More specifically, for this evaluation we have tested and trained the following classification algorithms (parameters used for each classifier were, initially, determined through *trial-and-error*):

- Support Vector Machine (SVM) [Burges, 1998], with ν-Support Vectors (trained with $\nu = 0.1$), and with a *Histogram intersection* kernel function
- Multi Layer Perceptron (MLP), with *back-propagation* training, two hidden layers and a layer configuration, according to: $x - 10 - 15 - 2$

[5]The collected data set is available under: http://reground.cs.kuleuven.be, and the *human-annotation interface* is available under:
https://bitbucket.org/reground/anchoring.

- k-Nearest Neighbor (k-NN), trained and tested with $k = 3$
- Normal Bayes Classifier (Bayes) [Fukunaga, 2013]

Collected dataset was randomly divided 70/30 into training/test samples, giving us a total of 3780 *training* samples and 1620 *testing* samples. Resulting average classification accuracy and F1 score for each trained classifier is listed in Figure 8.3.

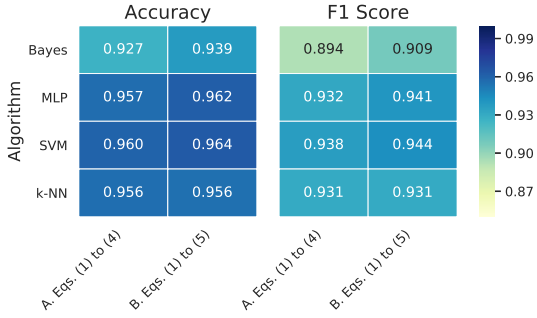


Figure 8.3: Resulting average *classification accuracy* together with *F1 score* for each used model for our approach to learn the anchoring functionalities.

Given the result, presented in Figure 8.3, it is seen that the best average *classification accuracy* of 96.4% was achieved by the use of the SVM classifier. The highest average *F1 score* (for a *true match*) of 94.4% was, likewise, achieved with the same SVM classifier. By the results seen in Figure 8.3, it should, however, also be noted that the differences in accuracy between the MLP classifier and the SVM classifier are close to insignificant (only 0.2%). Nevertheless, the best trained resulting SVM model was formally integrated as a part of the initial *matching function* of the anchoring system such that the predicted result of the SVM model was used to determine if an unknown candidate object was matching an existing anchor (i.e., if the object should be *re-acquired* as an existing matching anchor), or if no current anchors were matching the candidate object (i.e., if a new anchor should be *acquired* for the object). Integrated classification approach was, subsequently, used for the remaining experiments presented in the following Section 8.3.2.

By comparing the results between omitting (*column A*) or considering (*column B*) the time difference as an additional attribute (mapped to a time distance according to Equation 8.5), it is also evident that the time t, in fact, is a relevant factor for the concept of anchoring. The intuition behind including this additional time attribute in the evaluations presented in this section was to capture time-dependent changes in the environment while learning how to anchor objects, e.g., the position of an object can only change with a limited velocity between sequential frames. Through examining

the results for the best resulting SVM models, it is seen that our intuition was correct and that we achieved 0.4% better classification accuracy and 0.6% better F1 score, as a result of including the time difference as a feature. Worth noted, it can further be seen by the results in Figure 8.3, that the k-NN classifier was the only classifier that did not benefit from increasing the dimensionality of the input data by including the time difference as an additional attribute.

Finally, it should also be noted that the integrated classification approach can, in some cases, returning several matching candidate anchors (i.e., a candidate object can be *re-acquired* as more than one existing matching anchor). It is, therefore, important to globally consider all possible candidates for all observed objects in each frame in order to determine the best matching candidate anchor for each observed object. For the work presented in this paper, we used an SVM classifier with continuous output values such that the globally best matching candidate anchor was determined in a *winner takes all* manner.

8.3.2 Tracking of Occluded Objects

Despite the accuracy of the anchoring system, presented in previously Section 8.3.1, there are scenarios where the pure anchoring system fails to correctly *acquire* or *re-acquire* an object, e.g., when an object is occluded and moved by another object. For a changing and adaptable system to handle the world modeling of such scenarios, the system must further incorporate *model-based object tracking* in order to maintain objects that are not perceived by the input sensors. We achieve this by expressing a theory of occlusion in form of dynamic distributional clauses. In this section, we will exemplify how our approach of integrating DDC into the anchoring framework (cf. Section 8.2.4 and Figure 8.1–(2.) and (3.)) can handle such scenarios with occluded objects, and as a subsequent result further improve the anchoring accuracy.

Proof of Concept

To demonstrate how the *anchoring system* benefits from the feedback of the *inference system* (and consequently how the inference system benefits from the same integration), we plot the particles in form of point positions (representing the belief of the world in the inference system, as described in Section 8.2.4), concurrently with the output of the anchoring system, as exemplified in Figure 8.4. The mean position in 3-D space of the particles for each object that was not directly perceived at time t, e.g., objects occluded by another object, was subsequently fed back to the reinstated *track* functionality of the *anchoring system* such that the position of an anchor (even though the anchored object was not observed), was updated to the most probable position according to the *inference system*.

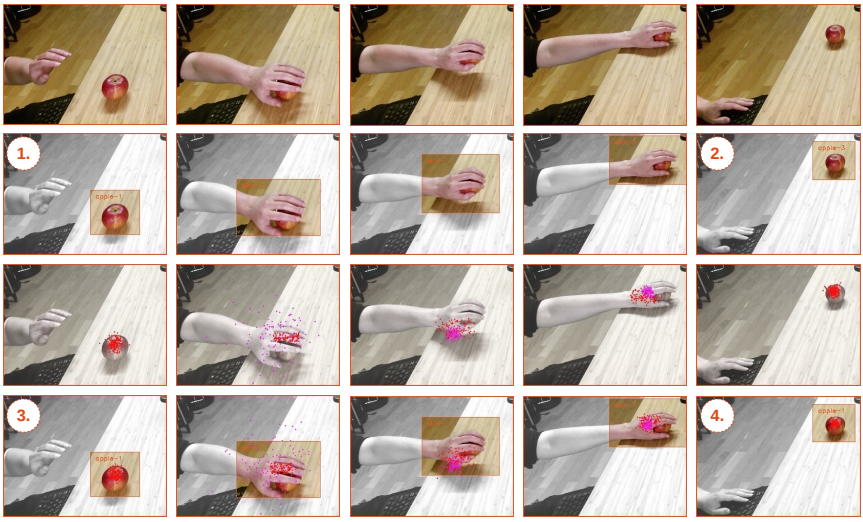


Figure 8.4: A depiction of how suggested system benefits of combined object anchoring and probabilistic object tracking. *Rows in order from the top: 1st)* representing screenshots of a scenario where a human hand is occluding an apple while the apple is moved, *2nd)* corresponding resulting anchored objects while *only* using the *anchoring system* (note that the original `apple-1` object is lost while it is occluded and moved by the `skin-1` object, and a new `apple-3` object is, therefore, *acquired* in the end of the scenario), *3rd)* plotted particles given by the *inference system* during execution of suggested integrated approach, and *4th)* corresponding resulting anchored objects of the *anchoring system* supported by the feed back of the *inference system* (note that in this case is the position of `apple-1` object tracked while it is occluded and moved by the `skin-1` object, and the `apple-1` object is, accordingly, *re-acquired* in the end of the scenario).

Comparing the resulting anchored objects, seen in Figure 8.4, it is evident that there is a significant difference in resulting anchors. In the case where *only* the *anchoring system* was used (Figure 8.4 – *2nd row from top*), it can be seen that the initial `apple-1` object (seen in Figure 8.4–(1.)) is lost while the object is occluded and moved by the `skin-1` object. Consequently, when the apple object reappears in the scene, the anchoring system cannot determine if the object is a new `apple` or the previously anchored `apple-1`, and as a result *acquire* an new anchor `apple-3` (seen in Figure 8.4–(2.)). However, in the case where both the *anchoring system* and the *inference system* are used (Figure 8.4 – *bottom row*), and where the position of the *tracked* `apple-1` object (seen in Figure 8.4 –(3.)) is fed back to anchoring system while the object is moved,

it can seen that the apple object, instead, is correctly *re-acquired* as `apple-1` once the object reappears in the scene (seen in Figure 8.4–(4.)). Note that, rather than using a dedicated classifier for recognizing different human body parts, we have, instead, fine-tuned our object classification *GoogLeNet* model to recognize human `skin` objects as one of the object categories.

Exemplifying Scenarios

Given the exemplified proof-of-concept (presented in the previous Section 8.3.2), we will in this section further demonstrate a number of scenarios where our suggested integrated system excels (compared to an anchoring approach that exclusively is based on perceptual observations of objects):

1. *Simple occlusion* – we start with two objects (among other objects) that are both visible. We then hide the smaller one of the objects behind the bigger one. The occluded object does not produce any sensor data. We can, however, reason about it. Then the smaller object reappears in the scene, we should be able to associate the reappearing object with the one from before (same anchor with high probability).
2. *Moving an occluded object* – we now want to *track* an object for which no sensor data is available. We start again with two objects that are both visible. We then hide the smaller object underneath the bigger object, move the bigger object and, subsequently, reveal the smaller object. We should be able to associate the reappearing object with the one from before.
3. *Moving occluded objects with unexpected revealing* – similar scenario as before only this time we start out with also having an unknown object hidden which the observer initially does not know about. We now hide again the (visible) smaller object underneath the bigger object, move the bigger object, but this time reveal the initially unknown hidden object. The system should recognize this as a new object. Then the other (initially visible) smaller object is revealed, the system should recognize this object as the previously anchored object.
4. *Shell game* – we start with three identical containers and a smaller object. We then hide the smaller object underneath one of the three containers and start shuffling the containers around. We should now be able to ask the system under which of the three containers the hidden objects is located.

In Figure 8.5, we exemplify our results of stated scenarios with a number of screen-shots during the execution of each scenario[6]. The scenarios were performed in near

[6]Full videos are available under: http://reground.cs.kuleuven.be/

real-time (8-10 frames per second) on a laptop Intel(R) i7 CPU 2.60GHz with 16 GB memory and an NVIDIA Quadro M1000M. This constitutes a promising feature of our approach as we do not need access to high-performance machines to deploy our system.

In the first example with *simple occlusion* (Figure 8.5 – *1st row from top*), it can be seen that as soon as the `cup` occludes the smaller `ball` object, the object is immediately tracked and maintained by the probabilistic reasoner (seen in Figure 8.5–(1.)). Opposite, once the `ball` objects reappear in the scene, it is no longer any need to probabilistically track the object, and the object is, once again, maintained through anchoring (seen in Figure 8.5–(2.)).

Through the second example (Figure 8.5 – *2nd row from top*), it is illustrated how the combined system handles *movements during occlusions*. In this example, a `glove` object (a human hand) is occluding while moving an `apple` object. As soon as the `apple` object is occluded by the `glove`, the `apple` object is tracked and maintained though probabilistic reasoning (seen in Figure 8.5–(3.)). The tracked position of the occluded object is continuously fed back to the anchoring system (through the newly instated anchoring *track* functionality), and the `apple` object is, consequently, *re-acquired* as the same `apple-1` once the object reappears in the scene (seen in Figure 8.5–(4.)).

In the third example (Figure 8.5 – *3rd row from top*), we demonstrate how our combined systems truly works in symbiosis. In this case, a similar scenario of *moving an occluded object* is exemplified where a `ball` (`ball-1`) is occluded while moved by a `glove`. However, another unknown `ball` is initially also hidden underneath the `glove` (in the human hand). This hidden `ball` is later introduced in the scene during the execution of the scenario (seen in Figure 8.5–(5.)). Nevertheless, since the second `ball` is different in appearance (compared to `ball-1`), this newly introduced object is correctly *acquired* as an new `ball-2` object, while the first `ball-1` object correctly remains tracked, and is subsequently *re-acquired* as the same `ball-1` object once reappearing in the scene (seen in Figure 8.5–(6.)).

Finally, in the fourth example (Figure 8.5 – *4th-6th row from top*), we reconnect with our initial motivation statement, through presented screen-shots captured during execution of a *shell game* scenario. In this example, a smaller `block` object (`block-3`) is hidden underneath one of three identical larger `block` objects (`block-1`), seen in Figure 8.5–(7.). All the larger `block` objects are, subsequently, moved around and shuffled. Nevertheless, during all movements is the hidden `block-3` object tracked though the relation with the occluding counterpart (`block-1`), i.e., the inference system is repeatedly speculating about the position of the hidden `block-3`, and the tracked position of is continuously fed back to the anchoring system. Consequently, once the hidden object is revealed and reappear in the scene (seen in Figure 8.5–(8.)), the object

Figure 8.5: Examples of screen-shots captured during the execution of stated scenarios. Visual perceived anchored objects are symbolized with the unique anchor id (e.g., `ball-2`), while occluded hidden objects are depicted by plotted particles that represent possible positions of the occluded object in the inference system. *Rows in order from the top: 1st)* example of *simple occlusion* where a `ball` is hidden behind a `cup`, *2nd)* depicts the *movement* of an *occluded object* where a `glove` (or human hand) is occluding while moving an `apple`, *3rd)* similar example of *moving an occluded object* where a `glove` is occluding while moving a `ball` (`ball-1`), but in this case is also another `ball` object (`ball-2`) introduced during the execution of the scenario, *4-6th)* illustrate a *shell game* scenario where a smaller object (`block-3`) is hidden under one of three identical containers (`block-2`), and where the containers, subsequently, are shuffled around.

is correctly *re-acquired* as the same `block-3` object.

8.4 Related Work

The importance of data association and object tracking in relation to perceptual anchoring was widely discussed by LeBlanc in his Ph.D. thesis on the topic of cooperative anchoring [LeBlanc, 2010]. Around the same time, and as an alternative to traditional anchoring, early work on perceptual and probabilistic anchoring was presented by Blodow et al. [2010]. The history of objects was maintained as computationally complex scene instances and the approach was, therefore, mainly intended for solving the problem of anchoring and maintaining coherent instances of objects in object kidnapping scenarios, i.e., when an object disappears from the scene and later reappears in a different location.

The idea of probabilistic anchoring was subsequently further explored by Elfring et al., which introduced probabilistic multiple hypothesis anchoring [Elfring et al., 2013]. This approach utilizes *Multiple Hypothesis Tracking*-based data association [Reid, 1979], in order to maintain changes in anchored objects, and thus, maintain an adaptable world model. In similarity with their work, we acknowledge that a proper data association is important for object anchoring, and we support the requirements identified by the authors for a changing and adaptable world modeling algorithm, which are formulated to include: 1) *appropriate anchoring*, 2) *data association*, 3) *model-based object tracking*, and 4) *real-time execution*. However, contrary to the work of Elfring et al., we address the tasks of appropriate anchoring and data association in a holistic fashion by introducing a learned anchoring matching function that administers both tasks. Moreover, in contrast to the work presented in [Elfring et al., 2013], we do not encourage a highly integrated approach that supports a tight coupling between object anchoring/probabilistic data association and object tracking. Instead, our approach maintains a loose coupling, which is motivated by the fact that a MHT procedure will inevitably suffer from the *curse of dimensionality* [Bellman, 1957]. A purely *probabilistic anchoring* approach, as presented in [Elfring et al., 2013], will, therefore, further propagate the curse of dimensionality into the concept of anchoring.

The limitation in the use of MHT for world modeling has also been acknowledged in a recent publication on the topic of data association for semantic world modeling [Wong et al., 2015]. While this work inherits the same problem formulation, it substantially differs in approach. The authors discuss and exemplify issues related to the use of a tracking-based approach for world modeling, such as intractable branching of the tree-structured tracks of possible hypothesis, and instead, suggests a *clustering* approach based on *Markov Chain Monte Carlo Data Association* [Oh et al., 2009]. In the same work on data association for semantic world modeling, Wong et al. also pointed

out some characteristics that differentiate world modeling from target tracking. For example, an appropriate world modeling algorithm should take into consideration that the state of most objects do not change between frames, while a tracking algorithm must consider unchanged objects as possible valid targets. For the approach presented in this paper, we are assuming similar characteristics through high-level tracking of objects (and those objects only) that are not directly perceived by the sensory input data.

From the relational point of view, which enables us to carry out reasoning, some research has been conducted on utilizing relations to improve the tracking of real world entities and state estimation ([Manfredotti, 2009; Mösenlechner; Beetz, 2009; Tenorth; Beetz, 2013; Nitti et al., 2014]). The most expressive of theses approaches is by Nitti et al. in [Nitti et al., 2014]. They utilize a relational particle filter, expressed in DDC, to carry out the tracking of objects and handle occlusions. In a box packaging scenario, where boxes are placed inside each other, they showed that binary predicates like `inside`/2 are helpful when tracking objects that are not directly observable. However, Nitti et al. assumed the data association problem to be solved by identifying the objects (boxes) by *augmented reality* (AR) tags – hence, very strictly limiting the usage of their framework in real-world scenarios.

In another recently published work on anchoring, Ruiz-Sarmiento et al. [2017] focus on spatial features and distinguish *unary* object features, e.g., the position of an object, from *pairwise* object features, e.g., the distance between two objects, in order to build a graph-based world model that can be exploited by a *probabilistic graphical model* [Koller; Friedman, 2009] in order to leverage contextual relations between objects to support 3-*D* object recognition. Günther et al. [2018] have further exploited this graph-based model on spatial features and propose, in addition, to learning the matching function through the use of a *Support Vector Machine* (trained on samples of object pairs manually labeled as *"same or different object"*), in order to approximate the *similarity* between two objects. The assignment of candidate objects to existing anchors is, subsequently, calculated using prior similarity values and a *Hungarian* method [Kuhn, 1955]. However, in contrast to Günther et al. [2018], the matching function that we introduced does not only rely upon spatial features (or attributes), but can also take into consideration visual features (such as color features), as well as semantic object categories, in order to approximate the anchoring matching problem.

Conclusions

In this chapter we presented an anchoring approach that combines bottom-up anchoring and probabilistic reasoning. Coupling a probabilistic inference system to the anchoring system enables the tracking of objects that are not directly observed. The presented

system is, however, not fully probabilistic but provides only an approximation of the entire probability distribution by enforcing certain assumptions. For instance, the design of the anchoring system assumes that the object segmentation and object classification are independent of each other. Furthermore, the design of the anchoring system does also assume that the object segmentation and classification are deterministic and that there is no noise. Obviously this is not true, there is always noise. One could argue now, as our model of the world (in form of the anchoring system) is wrong, that our anchoring approach is wrong as well. Here we would like to quote Box *"all models are wrong, but some are useful"* [Box, 1976; Box, 1979]. The question now becomes not whether we have a *correct* model of the world but whether we have a useful model. We have shown in this chapter that we can answer the latter of the two question affirmatively. More precisely, we presented an anchoring system able of performing object anchoring in (near) real-time. For situations where, real-time anchoring is not necessary but where, for example, more robustness towards uncertainty in the sensors is needed our model of the world, i.e the anchoring system will not be a *good* fit, in the sense that it won't be useful. In such situations the world has to be represented through other models.

Chapter 9

A Two-Fold Extension*

The anchoring system as described in Chapter 8 is not capable of handling probabilistic states, which means that the theory of occlusion has to describe unimodal probability distributions. In this chapter, we repair this deficiency (cf. Section 9.2.2). Moreover, the theory of occlusion had to be hand-coded (also the case for [Nitti et al., 2013]). We replace the hand-coded theory of occlusion by a learned one (cf. Section 9.3).

9.1 Introduction

In the previous chapter, we coupled the probabilistic logic programming language DDC to a perceptual anchoring system [Persson et al., 2020b], which endowed the perceptual anchoring system with probabilistic reasoning capabilities. A major challenge in combining perceptual anchoring with a high-level probabilistic reasoner, and which is still an open research question, is the administration of *multi-modal* probability distributions in anchoring[1]. In this chapter, we extend the anchoring notation in order to handle additionally multi-modal probability distributions.

A second point that we have not addressed in the previous chapter (based on [Persson et al., 2020b]), is the learning of probabilistic rules that are used to perform probabilistic

*This chapter is based on [Zuidberg Dos Martires et al., 2020b].

[1]A multi-modal probability distribution is a continuous probability distribution with strictly more than one local maximum. The key difference to a uni-modal probability distribution, such as a simple normal distribution, is that summary statistics do not adequately mirror the actual distribution. In perceptual anchoring these multi-modal distributions do occur, especially in the presence of object occlusions, and handling them appropriately is critical for correctly anchoring objects. This kind of phenomenon is well known when doing filtering and is the reason why particle filters can be preferred over Kalman filters.

logic reasoning. We show that, instead of hand-coding these probabilistic rules, we can adapt existing methods present in the SRL literature to learn them from raw sensor data. In other words, instead of providing a model of the world to a robotic agent, it learns this model in form of probabilistic logical rules. These rules are then used by the robotic agent to reason about the world around it, i.e. perform inference. More concretely: we learn the theory of occlusion instead of hand-coding it.

We evaluate the these two extensions of perceptual anchoring on three showcase examples that exhibit the following characteristics: 1) object occlusion induces a multi-modal probability distributions 2) the theory of occlusion is learned.

9.2 Anchoring of Objects in Multi-Modal States

In this section, we present a *probabilistic anchoring framework* that can handle multi-modal probability distributions. An overview of our proposed framework, which is implemented utilizing the libraries and communication protocols available in the Robot Operating System, can be seen in Figure 8.1. Note, our previous anchoring system, seen in Figure 8.1–(2.), was unable to handle probabilistic states of objects. While the probabilistic reasoning module, seen in Figure 8.1–(3.), was able to model the position of an object as a probability distribution over possible positions, the anchoring system only kept track of a single deterministic position: the expected position of an object. Therefore, we extend the anchoring notion towards a probabilistic anchoring approach in order to enable the anchoring system to handle multi-modal probability distributions.

9.2.1 Requirements

Before presenting our proposed probabilistic anchoring approach, we first introduce the necessary requirements and assumptions (which partly originate in the previous chapter):

1. We assume that unknown anchor representations, α_t^y, are supplied by a *black-box* perceptual processing pipeline, as exemplified in Figure 8.1–(1.). They consist of extracted *attribute measurements* and corresponding grounded *predicate symbols*. We further assume that for each perceptual representation of an object, we have the following attribute measurements: *1)* a *color attribute* (ϕ_y^{color}), *2)* a *position attribute* (ϕ_y^{pos}), and *3)* a *size attribute* (ϕ_y^{size}).

   Example 9.1. *In this paper we use the combined Depth Seeding Network (DSN) and Region Refinement Network (RNN), as presented by* Xie *et al. [2019], for*

the purpose of segmenting arbitrary object instances in tabletop scenarios. This two-stage approach leverages both RGB and depth data (given by a Kinect V2 RGB-D sensor), in order to first segment rough initial object masks (based on depth data), followed by a second refinement stage of these object masks (based on RGB data). The resulting output for each segmented object, is then both a 3*-D spatial percept* ($\phi_y^{spatial}$)*, as well as a* 2*-D visual percept* (ϕ_y^{visual})*. For each segmented spatial percept, and with the use of the Point Cloud Library (PCL), are both a position attribute measured as the* 3*-D geometrical center, and a size attribute measured as the* 3*-D geometrical bounding box. Similarly, using the Open Computer Vision Library (OpenCV), a color attribute is measured as the discretized color histogram (in HSV color-space) for each segmented visual percept, as depicted in Figure 9.1.*

2. In order to semantically categorize objects, we further assume that a Convolutional Neural Network (CNN), such as the GoogLeNet model [Szegedy et al., 2015], is available. In the context of anchoring, the inputs for this model are segmented *visual percepts* (π_y^{visual}), while resulting object categories, denoted by the predicate $p_y^{category} \in \mathcal{P}$, are given together with the predicted probabilities $\phi_y^{category}$.

   Example 9.2. *For this work, we have used the same fine-tuned model as used in [*Persson *et al., 2020b], which is based on the network architecture of the* 1 K *GoogLeNet model, developed by* Szegedy *et al. We have, however, fine-tuned the model to classify* 101 *objects categories that are only relevant for a household domain, e.g.,* `mug`*,* `ball`*,* `box`*, etc., where the model was trained for a top-1 accuracy of* 73.4% *(and a top-5 accuracy of* 92.0%*). An example of segmented objects together with the 3-top best object categories, given by the integrated GoogLeNet model, is illustrated in Figure 9.2.*

   In addition, this integrated model is also used to enhance the traditional *acquire* functionality such that a unique identifier x is generated based on the object category symbol $p^{category}$. For example, if the anchoring system detects an object it has not seen before and classifies it as a `cup`, a corresponding unique identifier $x =$ `cup-4` could be generated (where the 4 means that this is the forth distinct instance of a `cup` object perceived by the system).

3. We require the presence of a *probabilistic inference system* coupled to the *anchoring system*, as illustrated in Figure 8.1–(3.). The anchoring system is responsible for maintaining objects perceived by the sensory input data and for maintaining the observable part of the world model. Maintained anchored object representations are then treated as observations in the inference system, which uses relational object tracking to infer the state of occluded objects through their relations with perceived objects in the world. This inferred belief of the world is

Figure 9.1: Examples of measured *color attribute* (measured as the *discretized color histogram* over each segmented object).

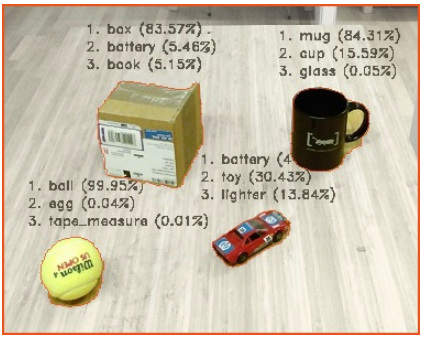


Figure 9.2: Examples of *semantically categorized objects* (depicted with the *3-top best object categories* for each segmented object).

then sent back to the anchoring system, where the state of occluded objects is updated. The feedback-loop between the anchoring system and the probabilistic reasoner results in an additional anchoring functionality [Persson et al., 2020b]:

- *Track* – extends the definition of an anchor α^x from time $t-1$ to time t. This functionality is directly responding to the state of the probabilistic object tracker, which ensures that the percepts pointed to by the anchor are the adequate perceptual representation of the object, even though the object is currently not perceived.

9.2.2 Probabilistic Anchoring System

The entry point for the anchoring system, seen in Figure 8.1–(2.), is a learned *matching function*. This function assumes a bottom-up approach to perceptual anchoring, described in [Loutfi et al., 2005], where the system constantly receives candidate anchors and invokes a number of attribute specific matching similarity formulas (i.e., one matching formula for each measured attribute). More specifically, a set of attributes Φ_y of an unknown candidate anchor α_t^y (given at current time t) is compared against the set of attributes Φ_x of an existing anchor α_{t-k}^x (defined at time $t-k$) through attribute specific similarity formulas. For instance, the similarity between the *positions attributes* ϕ_y^{pos} of an unknown candidate anchor, and the last updated position $\phi_{t-k,x}^{pos}$ of an existing anchor, is calculated according to the L^2-norm (in 3-D space), which is further mapped to a *normalized similarity value* [Blodow et al., 2010]:

$$d^{pos}(\phi^{pos}_{t-k,x}, \phi^{pos}_{t,y}) = e^{-L^2(\phi^{pos}_{t-k,x}, \phi^{pos}_{t,y})} \tag{9.1}$$

Hence, the similarity between two *positions attributes* is given in interval [0, 1], where a value of 1 is equivalent with perfect correspondence. Likewise, the similarity between two *color attributes* are calculated by the *color correlation*, while the similarity between *size attributes* is calculated according to the *generalized Jaccard similarity* (for further details regarding similarity formulas, we refer to our previous work [Persson et al., 2020b]). The similarities between the attributes of a known anchor and an unknown candidate anchor are then fed to the learned matching function to determine whether the matching function classifies the unknown anchor to be *acquired* as a new anchor, or *re-acquired* as an existing anchor. This matching function is utilized by a support vector machine, which has been trained with the use of 5400 samples of *humanly annotated data* (i.e., human users have provided feedback about what they think is the appropriate anchoring action for objects in various scenarios), to a *classification accuracy* of 96.4%. It should, however, be noted that the inputs for this classifier are the various similarity values between attributes (cf. Equation 9.1), and that the classifier learns to interpret, combine and weight different similarity values between attributes in order to correctly determine whether a new anchor should be acquired, or if an existing anchor should be re-acquired. By omitting similarity values of specific attributes during training, we can also estimate the importance of different attributes. For example, excluding the similarity values between *color attributes* during training reduces the classification accuracy to 92.5%, while excluding the similarity values between *position attributes*, instead, decreases the accuracy to 72.8%. This illustrating example of the importance of the position of an object, in the context of anchoring, is a further motivation for reasoning about possible states once the position of an object changes during the absence of observations (e.g., in the case of movements during occlusions).

In our prior work on anchoring, the attribute values have, in addition, always been assumed to be deterministic within a single time step. This assumption keeps the anchoring system de facto deterministic even though it is coupled to a probabilistic reasoning module. We, therefore, extend the anchoring notation with two distinct specifications of (volatile) attributes:

1. An attribute $\phi_t \in \varphi$ is *deterministic at time t* if it takes a single value from the domain $D(\phi_t)$.
2. An attribute $\phi_t \in \varphi$ is *probabilistic at time t* if it is distributed according to a probability distribution $Pr(\phi_t)$ over the domain $D(\phi_t)$ at time step t.

Having a probabilistic attribute value ϕ_t (e.g., $\phi^{pos}_{t-k,x}$ in Equation 9.1), means that the similarity calculated with the probabilistic attribute values (e.g., the similarity value

d^{pos}), will also be probabilistic. Next, in order to use an anchor matching function together with probabilistic similarity values, two extensions are possible: *1)* extend the anchor matching function to accept random variables (i.e., probabilistic similarity values), or *2)* retrieve a point estimate of the random variable.

We chose the second option as this allows us to reuse the anchor matching function learned in [Persson et al., 2020b] without the additional expense of collecting data and re-training the anchor matching function. The algorithm to produce the set of matching similarity values that are fed to the anchor matching function is given in Algorithm 9.1, where lines 4-5 are the extension proposed in this work.

Algorithm 9.1 Attribute Compare

Input: Φ_x, Φ_y – *sets of anchor attribute values*
Output: $\mathcal{D}_{x,y}$ – *set of matching similarity values*

1: **function** AttributeCompare(Φ_x, Φ_y)
2: $\quad\mathcal{D}_{x,y} \leftarrow$ *empty set*
3: $\quad$**for each** $\phi_{t,x} \in \Phi_x$ **and** $\phi_{t,y} \in \Phi_y$ **do**
4: $\quad\quad$**if** $\phi_{t-1,x}$ is probabilistic **then**
5: $\quad\quad\quad\mathcal{D}_{x,y} \overset{+}{\leftarrow} \underset{\phi_{t-1,x}}{point\_estimate}(d(\phi_{t-1,x}, \phi_{t,y}))$
6: $\quad\quad$**else** ▷ deterministic case
7: $\quad\quad\quad\mathcal{D}_{x,y} \overset{+}{\leftarrow} d(\phi_{t-k,x}, \phi_{t,y})$
8: $\quad$**return** $\mathcal{D}_{x,y}$

The *point_estimate* function in Algorithm 9.1 (line 5) is attribute specific (indicated by the subscript ($\phi_{t-1,x}$)), i.e. we can chose a different point estimation function for *color attributes* than for *position attributes*. An obvious attribute upon which reasoning can be done is the position attribute, for example, in the case of possible occlusions. In other words, we would like to perform probabilistic anchoring while taking into account the *probability distribution of an anchor's position*. A reasonable goal is then to match an unknown candidate anchor with the most likely anchor, i.e., with the anchor whose position attribute value is located at the highest mode of the probability distribution of the position attribute values. This is achieved by replacing Line 5 in Algorithm 9.1 with:

$$\mathcal{F}_x^{pos} \leftarrow \left\{ \phi_{t-1,x}^{pos} \middle| \frac{\partial Pr(\phi_{t-1,x}^{pos})}{\partial \phi_{t-1,x}^{pos}} = 0 \right\} \tag{9.2}$$

$$\mathcal{D}_{x,y} \overset{+}{\leftarrow} \max_{\phi^{pos} \in \mathcal{F}_x^{pos}} (d^{pos}(\phi^{pos}, \phi_{t,y}^{pos})) \tag{9.3}$$

$\mathcal{F}_x^{pos}$ is the set of positions situated at the modes of the probability distribution $Pr(\phi_{t-1,x}^{pos})$. In Equation 9.3 we take the max as the co-domain of the position similarity value d^{pos}

is in [0, 1], where 1 reflects perfect correspondence (cf. Equation 9.1).

In [Persson et al., 2020b], we approximated the probabilistic state of the world in the *inference system* (cf. Figure 8.1–3.) by N particles, which are updated by means of particle filtering. The precise information that is passed from the inference system to the anchoring system is a list of N particles that approximate a (possible) multi-modal belief of the world. More specifically, an anchor α_t^x is updated according to the N particles of possible states of a corresponding object, maintained in the inference system, such that N possible 3-D positions are added to the volatile *position attributes* φ_x^{pos}. In practice we assume that samples are only drawn around the modes of the probability distribution, which means that we can replace line 5 of Algorithm 9.1 with:

$$\mathcal{D}_{x,y} \overset{+}{\leftarrow} \max_i \left(d^{pos}(\phi^{pos}_{t-1,x,i}, \phi^{pos}_{t,y}) \right) = \max_i \left(e^{-L^2(\phi^{pos}_{t-1,x,i}, \phi^{pos}_{t,y})} \right) \tag{9.4}$$

Where $\phi_{t-1,x,i}$ is a sampled position and i ranges from 1 to the number of samples N.

Performing probabilistic inference in the coordinate space is a choice made in the design of the probabilistic anchoring system. Instead, the probabilistic tracking could also be done in the HSV color space, for instance. In this case, the similarity measure used in Algorithm 9.1 would have to be adapted accordingly. It is also conceivable to combine the tracking in coordinate space and color space. This introduces, however, the complication of finding a similarity measure that works on the coordinate space and the color space at the same time. A solution to this would be to, yet again, learn this similarity function from data [Persson et al., 2020b].

9.3 Learning Dynamic Distributional Clauses

While several approaches exist in the SRL literature that learn probabilistic relational models, most of them focus on parameter estimation [Sato, 1995; Friedman et al., 1999; Taskar et al., 2002; Neville; Jensen, 2007] and structure learning has been restricted to discrete data. Notable exceptions include the recently proposed hybrid relational formalism by Ravkic et al. [2015], which learns relational models in a discrete-continuous domain but has not been applied to dynamics or robotics, and the related approach of Nitti et al. [2016b], where a relational tree learner DDC-TL learns both the structure and the parameters of distributional clauses. DDC-TL has been evaluated on learning action models (pre- and post-conditions) in a robotics setting from before and after states of executing the actions. However, there were several limitations of the approach. It simplified perception by resorting to AR tags for identifying the objects, it did not consider occlusion, and it could not deal with uncertainty or noise in the observations.

A more general approach to learning distributional clauses, extended with *statistical models* proposed in [Kumar et al., 2020][2]. Such a statistical model relates continuous variables in the body of a distributional clause to parameters of the distribution in the head of the clause. The approach simultaneously learns the structure and parameters of (non-dynamic) distributional clauses, and estimates the parameters of the statistical model in clauses. A DC program consisting of multiple distributional clauses is capable of expressing intricate probability distributions over discrete and continuous random variables. A further shortcoming of DDC-TL (also tackled by Kumar et al.) is the inability of learning in the presence of background knowledge — that is, additional (symbolic) probabilistic information about objects in the world and relations (such as spatial relations) among the objects that the learning algorithm should take into consideration.

However, until now, the approach presented in [Kumar et al., 2020] has only been applied to the problem of autocompletion of relational databases by learning a (non-dynamic) DC program. We now demonstrate with an example of how this general approach can also be applied for learning dynamic distributional clauses in a robotics setting. A key novelty in the context of perceptual anchoring is that we learn a DDC program that allows us to reason about occlusions.

Example 9.3. *Consider again a scenario where objects might get fully occluded by other objects. We would now like to learn the ToO that describes whether an object is occluded or not given multiple observations of the before and after state. In DDC we represent observations through facts as follows*

```
pos(o1_exp1):t ~= 2.3.
pos(o1_exp1)t+1 ~= 9.3.
pos(o2):t ~= 2.2.
pos(o2):t+1 ~= 9.2.
occluded_by(o1_exp1,o2_exp1):t+1.
pos(o3_exp1):t ~= 8.3.
⋮
```

For the sake of clarity, we have considered only one-dimensional positions in this example.

Example 9.4. *Given the data in form of dynamic distributional clauses, we are now interested in learning the ToO instead of relying on a hand-coded one, as in Example 7.4. An excerpt from the set of clauses that constitute a learned ToO is given below. As in Example 7.4, the clauses describe the circumstances under which an object (*Occluded*) is potentially occluded by another object (*Occluder*).*

[2] https://github.com/niteshroyal/DreaML

```
occluder(Occluded,Occluder):t+1 ~ finite(1.0:false) ←
occluded_by(Occluded,Occluder):t,
observed(Occluded):t+1.
occluder(Occluded,Occluder):t+1 ~ finite(0.92:true
    ,0.08:false) ←
        occluded_by(Occluded,Occluder):t,
        \+observed(Occluded):t+1.
occluder(Occluded,Occluder):t+1 ~ finite(P1:true,P2:
    false) ←
        \+occluded_by(Occluded,Occluder):t,
        \+observed(Occluded):t+1,
        distance(Occluded,Occluder):t~=Distance,
        logistic([Distance],[-16.9,0.8],P1),
        P2 is 1-P1.
```

Note that, in the second but last line of the last clause above the arbitrary threshold on the Distance *is superseded by a learned statistical model, in this case a logistic regression, which maps the input parameter* Distance *to the probability* P1*:*

$$\mathrm{P1} = \frac{1}{1 + e^{16.9 \times \mathrm{D} - 0.8}} \tag{9.5}$$

Replacing the hand-coded occluder *rule with the learned one in the theory of occlusion allows us to track occluded objects with a partially learned model of the world.*

In order to learn dynamic distributional clauses, we first map the predicates with subscripts that refer to the current time step t and the next time step t+1 to standard predicates, which gives us an input DC program. For instance, we map pos(o1_exp1) :t to pos_t(o1_exp1), and occluder(o1_exp1,o2_exp2):t+1 to occluder_t1 (o1_exp1 ,o2_exp2). The method introduced in [Kumar et al., 2020] can now be applied for learning distributional clauses for the target predicate occluder_t1 (o1_exp1,o2_exp2) from the input DC program.

Clauses for the target predicate are learned by inducing a distributional logic tree. An example of such a tree is shown in Figure 9.3. The key idea is that the set of clauses for the same target predicate are represented by a distributional logic tree, which satisfies the mutual exclusiveness property of distributional clauses. This property states that if there are two distributional clauses defining the same random variable, their bodies must be mutually exclusive. Internal nodes of the tree correspond to atoms in the body of learned clauses. A leaf node corresponds to a distribution in the head and to a statistical model in the body of a learned clause. A path beginning at the root node and proceeding to a leaf node in the tree corresponds to a clause. Parameters of the distribution and the statistical model of the clause are estimated by maximizing the

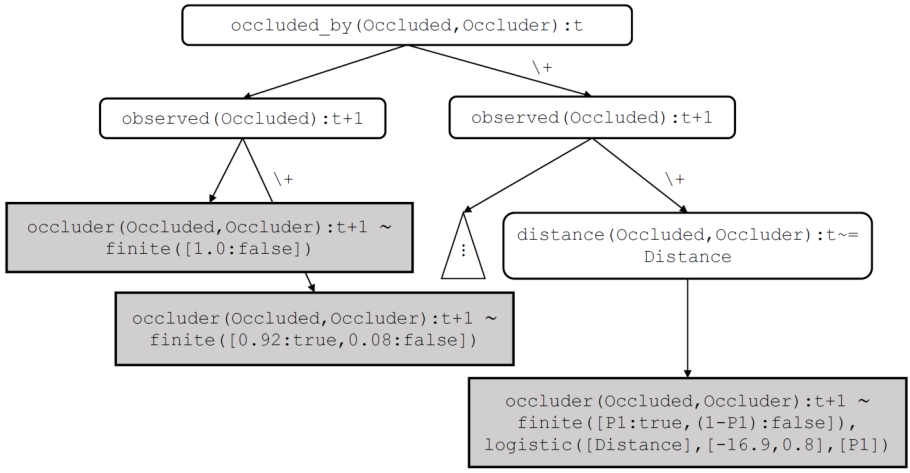


Figure 9.3: A distributional logic tree that represents learned clauses for the target occluder(Occluded,Occluder):t+1. The leftmost path corresponds to the first clause, the rightmost path corresponds to the last clause for occluder(Occluded,Occluder):t+1 in Example 9.3. Internal nodes such as occluder(Occluded,Occluder):t and observed(Occluded):t+1 are discrete features, whereas, internal nodes such as distance(Occluded,Occluder):t+1~=Distance is a continuous feature.

expectation of the log-likelihood of the target in partial possible worlds. The worlds are obtained by proving all possible groundings of the clause in the input DC program. The structure of the induced tree defines the structure of the learned clauses. The approach requires declarative bias to restrict the search space while inducing the tree. Note that the fragment of programs that can be learned by the algorithm described in [Kumar et al., 2020] does not include recursive programs, as only tree structured programs can be learned.

In summary, the *input* to the learner of Kumar et al. [2020] is a DC program consisting of

- background knowledge, in the form of DC clauses;
- observations, in the form of DC clauses — these constitute the training data;
- the declarative bias, which is necessary to specify the hypothesis space of the DC program [Adé et al., 1995];
- the target predicates for which clauses should be learned.

The *output* is:

- a set of DC clauses represented as a tree for each target predicate specified in the input.

In contrast to learning algorithms that tackle discrete data, the declarative bias used to learn rules with continuous random variables has to additionally specify whether a random variable is distributed according to a discrete probability distribution or a continuous probability distribution. In other words, the declarative bias specifies whether a leaf in the learned tree represents continuous or a discrete probability distribution. Currently the algorithm of Kumar et al. [2020] only supports normal distributions for continuous random variables and finite categorical distributions for discrete random variables.

Once the clauses are learned, predicates are mapped back to predicates with subscripts to obtain dynamic distributional clauses. For instance, occluder_t1(Occluded, Occluder) in the learned clauses is mapped back to occluder(Occluded,Occluder) :t+1.

The data used for the learning of the theory of occlusion consists of training points of before-after states of two kinds. The first kind are pairs describing a transition of an object from being observed to being occluded. Here, the data set contained 58 data point pairs, with 13 pairs describing the transition of an object from being observed to being occluded and the remaining 45 describing situations with an object being observed in the before state, as well as in the after state. Examples of two raw data points for the first kind can be seen in Figure 9.4. The second kind of data pairs describe an object being occluded in the before state as well as in the after state. Here we had 425 positively labeled data pairs, i.e. an object was occluded in the after state. For 416 of these pairs the labeling was correct while the remaining were mislabeled (the occluded object in the after state was labeled as observed in the before state). For negative data points (objects not occluded in the after state) we had 1152 data pairs. For 473 of these pairs the non-occluded object was labeled as not occluded in the before state as well, for 2 it was labeled as occluded and for the remaining there was no label in the before state. While for the first kind we did not have any mislabeled data, the data points for the second kind did exhibit a small percentage of inaccurately labeled data pairs, for example approximately $\approx 3\%$ for positive data pairs. Noise in the data was also present in the position of the objects – originating from the perceptual anchoring system.

The predicate specifying whether an object is occluded in the after state or not was the target predicate of the learner. In the declarative bias we specified the predicates to be used as features. These included predicates specifying whether an object is occluded in the before state, position predicates, and distance predicates between objects. Furthermore, the rule learner automatically decides on which statistical models,

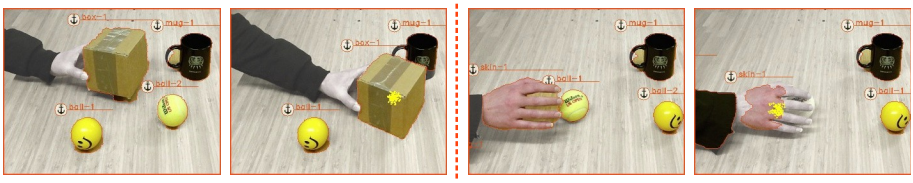


Figure 9.4: Depicted are two training points in the data set that were used to learn the transition rule of an object to another object. The panels on the left show a `ball` that is being occluded by a `box`, and on the right, the same `ball` that is being grabbed by a hand (or a `skin` object, as we have only trained our used GoogLeNet model to recognize general human skin objects instead of particular human body parts, cf. Section 9.2.1). The plotted dots on top of the occluding object represent samples drawn from the probability distribution of the occluded object, in other words the object that is labeled in the data set to transition into the occluding counterpart.

if any, to use in the learned rules. The available statistical models are linear, softmax, and logistic models.

The processed data that was fed to the distributional clauses learner is available online[3] as well as models with the learned theory of occlusion[4]. The learned theory of occlusions is conceptually close to the one shown in Example 9.4. The first-order nature of the learned rules enable the usage of the rules in situations of object occlusions with specific objects that were not present in the training data set.

9.4 Evaluation

A probabilistic anchoring system that is coupled to an inference system (cf. Section 9.2.2) is comprised of several interacting components. This turns the evaluation of such a combined framework, with many integrated systems, into a challenging task. We, therefore, evaluate the integrated framework as a whole on representative scenarios that demonstrate our proposed extensions to perceptual anchoring. In Section 9.4.1, we demonstrate how the extended anchoring system can handle probabilistic multi-modal states (described in Section 9.2). In Sections 9.4.2 and 9.4.3, we show that semantic relational object tracking can be performed with the probabilistic logic rules (in form of a DDC program) instead of handcrafted ones.

[3] https://bitbucket.org/reground/anchoring/downloads/
[4] https://bitbucket.org/reground/anchoring

9.4.1 Multi-Modal Occlusions

We present the evaluation in the form of screenshots captured during the execution of a scenario where we obscure the stream of sensor data. We start out with three larger objects (two `mug` objects and one `box` object), and one smaller `ball` object. During the occlusion phase, seen in Figure 9.5–(1.), the RGB-D sensor is covered by a human hand and the smaller `ball` is hidden underneath one of the larger objects. In Figure 9.5–(1.), it should also be noted that the anchoring system preserves the latest update of the objects, which is here illustrated by the outlined contour of each object. At the time that the sensory input stream is uncovered, and there is no longer any visual perceptual input of the `ball` object, the system can only speculate about the whereabouts of the missing object. Hence, the belief of the `ball`'s position becomes a multi-modal probability distribution, from which we draw samples, as seen in Figure 9.5–(2.). At this point, we are, however, able to track the smaller `ball` through its probabilistic relationships with the other larger objects. During all the movements of the larger objects, the probabilistic inference system manages to track the modes of the probability distribution of the position of the smaller `ball`. The probability distribution for the position of the smaller `ball` (approximated by N samples) is continuously fed back to the anchoring system. Consequently, once the hidden `ball` is revealed and reappears in the scene, as seen in Figures 9.5–(3.) and 9.5–(4.), the `ball` is correctly *re-acquired* as the initial `ball-1` object. This would not have been possible with a non-probabilistic anchoring approach.

9.4.2 Uni-Modal Occlusions with Learned Rules

The conceptually easiest ToO is one that describes the occlusion of an object by another object. Using the method described in Section 9.3, we learned such a ToO, which we demonstrate in Figure 9.6. Shown are two scenarios. In the one in the upper row a `can` gets occluded by a `box` — shown in the second screenshot. The `can` is subsequently tracked through its relation with the observed `box` and successfully re-anchored as `can-1` once it is revealed. Note that in the second screenshot, the `mug` is also briefly believed to be hidden under `box`, shown through the `black` dots, as the `mug` is temporally obscured behind the `box` and not observed by the vision system. However, once the `mug` is again observed the black dots disappear.

In the second scenario, we occlude one of two `ball` objects with a `box` and track the `ball` again through its relation with the `box`. Note that some of the probability mass accounts for the possibility for the occluded `ball` to be occluded by the `mug`. This is due to the fact that the learned rule is probabilistic.

In both scenarios, we included background knowledge that specifies that a `ball` cannot be the an occluder of an object (it does not *afford* to be the occluder). This is also

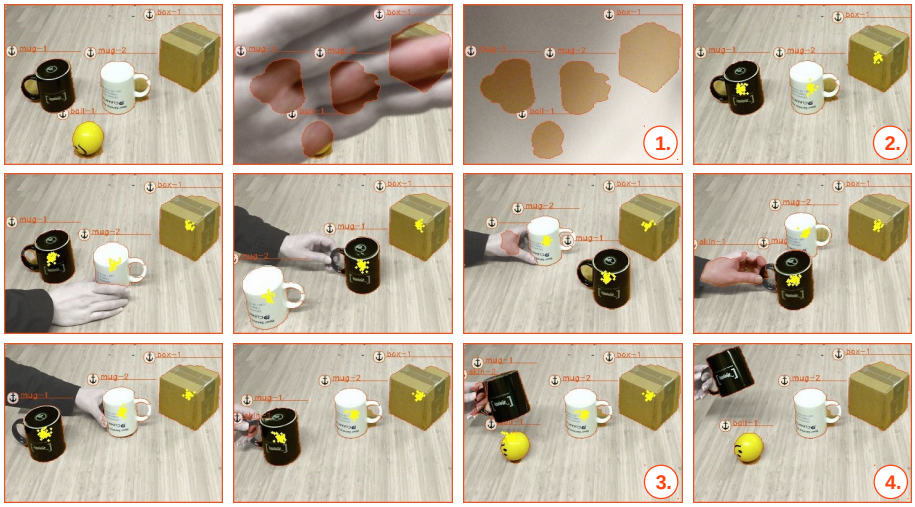


Figure 9.5: Screen-shots captured during the execution of a scenario where the stream of sensor data is obscured. Visually perceived anchored objects are symbolized by a unique anchor identifiers (e.g., `mug-1`), while occluded hidden objects are depicted by plotted particles that represent possible positions of the occluded object in the inference system. The screenshots illustrate a scenario where the RGB-D sensor is covered and a `ball` is hidden under either one of three larger objects. These larger objects are subsequently shuffled around before the whereabouts of the hidden `ball` is revealed.

why we see a probability mass of the occluded `ball` at the `mug`'s location and not at the observed `ball`'s location in the second scenario.

9.4.3 Transitive Occlusions with Learned Rules

```
occluded_by(Occluded, Occluder):t+1 ←
        occluded_by(Occluded, Occluder):t,
        \+observed(Occluded):t+1,
        \+observed(Occluder):t+1,
        occluded_by(Occluder, _):t+1.
```

Extending the ToO from Section 9.4.2 with the above rule, enables the anchoring system to handle recursive occlusions. We demonstrate such a scenario in Figure 9.7. Initially, we start this scenario with a `ball`, a `mug` and a `box` object (which in the beginning is miss-classified as `block` object, cf. Figure 9.2). In the first case of occlusion, seen in

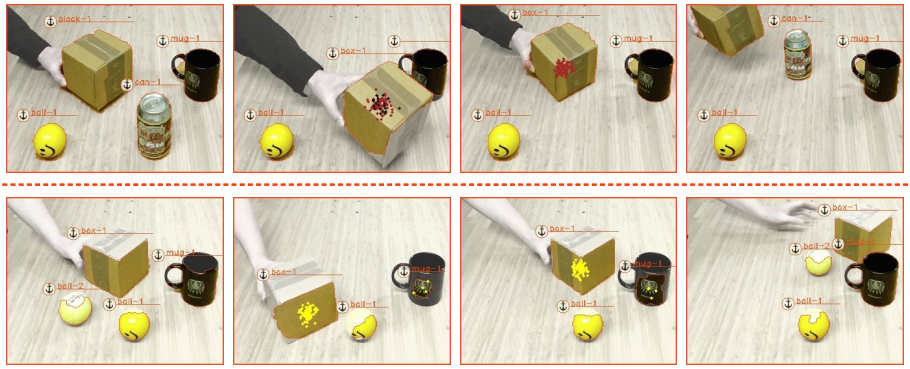


Figure 9.6: The two scenario show how a learned ToO is used to perform semantic relational object tracking. In both scenarios, an object is occluded by a `box` and successfully tracked before the occluded object is being revealed and again *re-acquired* as the same initial object.

Figures 9.7–(1.), we have the same type of uni-modal occlusion as described in the previous Section 9.4.2, where the `mug` occludes the `ball` and, subsequently, triggers the learned relational transition (where plotted `yellow` dots represent samples drawn from the probability distribution of the occluded `ball` object). In the second recursive case of occlusion, seen in Figure 9.7–(2.), we proceed by also occluding the `mug` with the `box`. Above rule administers this *transitive occlusion* — triggered when the `ball` is still hidden underneath the `mug` and the `mug` is occluded by the `box`. This is illustrated here by both `yellow` and `black` plotted dots that represent samples drawn from the probability distributions of occluded `mug` and the transitively occluded `ball` object, respectively. Consequently, once the `box` is moved, both the `mug` and the `ball` are tracked through the transitive relation with the occluding `box`. Reversely, it can be seen, in Figure 9.7–(3.), that once the `mug` object is revealed the object is correctly *re-acquired* as the same `mug-1` object, while the relation between the `mug` and the occluded `ball` object is still preserved. Finally, as the `ball` object is revealed, in Figure 9.7–(4.), it can be also seen that the object is, likewise, correctly *re-acquired* as the same `ball-1` object.

9.5 Future Work

With the approach presented in this chapter for probabilistic anchoring we are merely able to perform MAP inference. In order to perform full probabilistic anchoring, one would need to render the anchor matching function itself fully probabilistic, i.e. the anchor matching function would need to take as arguments random variables and again

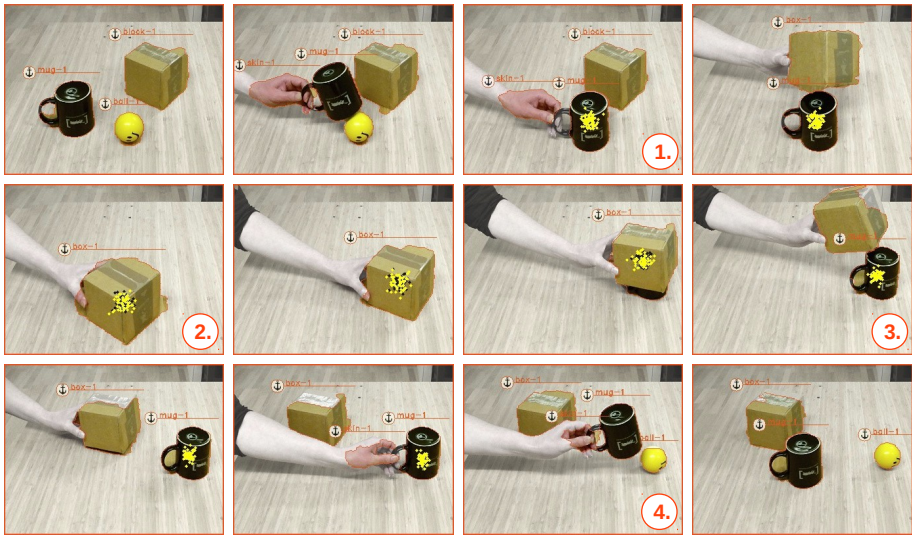


Figure 9.7: A scenario that demonstrates transitive occlusions based on learned rules for handling the theory of occlusions. First the `ball` is occluded by the `mug` (indicated by the yellow dots) and subsequently the `mug` is occluded in turn by the `box` (indicated by the black dots). Once the `mug` is observed again the `ball` is still believed to be occluded by the `mug`.

output probability distributions instead of point estimates — ideas borrowed from multi-hypothesis anchoring [Elfring et al., 2013] might, therefore, be worthwhile to be considered in future work.

Conclusions

In this chapter we presented a two-fold extension to the work in the previous, where we proposed an approach that couples an anchoring system to a probabilistic inference system. Firstly, we extended the notions of perceptual anchoring towards the probabilistic setting by means of probabilistic logic programming. Secondly, we have deployed methods from statistical relational learning to the field of anchoring. While we have shown in a set of showcase scenarios that both extensions improve our approach presented in Chapter 8, it can be criticized that the experimental evaluation is not very extensive. This is in fact a criticism that can be made more generally to research in robotics and related fields, such as object anchoring. Additionally the experimental evaluation is also often lacking reproducibility. A possible avenue forward

is the use of simulators for the experimental evaluation. Simulators are already being used for machine learning in robotics, e.g. [Tobin et al., 2017; Peng et al., 2018]. This is referred to as *sim2real*. A similar approach could be used for reproducible experimental set-ups. An important feature of such as set-up will have to be the faithful simulation of uncertainty of the real-world. For example, when simulating an anchoring system the simulated camera through which the simulated world is observed has to match faithfully the noise of the real-world camera.

Conclusions

In the last part of the thesis we tackled our thirds research question:

RQ3: Can we equip a cognitive robotics system with probabilistic reasoning capacities?

In Chapter 8, we presented how we are able to improve the overall anchoring process by introducing a post-anchoring high-level probabilistic reasoning procedure with the purpose of predicting the state of objects that are not directly perceived through the perceptual sensor data, e.g., in case of object occlusions. To retain the integrated framework in a coherent cognitive state, we have suggested a loosely coupled integration between proposed inference system and anchoring system, while a tight feedback loop is preserved in order to maintain consented tracked positions of objects. We have presented the proof-of-concept of how this integrated framework is used to model and manage a consistent semantic world model of perceived objects in dynamic scenarios. To the best of our knowledge this constitutes the first system capable of handling occlusions that are full, long and complex.

Furthermore, we introduced a novel anchoring matching approach based on classification of humanly annotated ground truth data of real-world objects for determining whether a perceived object has previously been observed (or not), and, subsequently, invoke correct anchoring functionality (*acquire* or *re-acquire*) in order to correctly anchor perceived objects. Through the presented results, we have shown that our learned anchoring matching approach is able to accurately anchor objects and maintain consistent representations of objects.

In Chapter 9, we presented a two-fold extension to our previous work on semantic world modelling [Persson et al., 2020b] (presented in Chapter 8). Firstly, we extended the notions of perceptual anchoring towards the probabilistic setting by means of probabilistic logic programming. This allowed us to maintain a multi-modal probability distribution of the positions of objects in the anchoring system and to use it for matching and maintaining objects at the perceptual level. We illustrated the benefit of this

approach with the scenario in Section 9.4.1, which the anchoring system was able to resolve correctly only due to its ability of maintaining a multi-modal probability distribution.

Secondly, we have deployed methods from statistical relational learning to the field of anchoring. This approach allowed us to learn, instead of handcraft, rules needed in the reasoning system. A distinguishing feature of the applied rule learner [Kumar et al., 2020] is its ability to handle both continuous and discrete data. We then demonstrated that combining perceptual anchoring and SRL is also feasible in practice by performing relational anchoring with a learned rule. This scenario did also exhibit a further strength of using SRL in anchoring domains, namely that the resulting system becomes a highly modularizable. In our evaluation, for instance, we were able to integrate an extra rule into the ToO, which enabled us to resolve recursive occlusions. A possible future direction of this work is to exploit how anchored objects and their spatial relationships facilitate the learning of both the function of objects, as well as object affordances – similar to previously presented work on learning objects affordances from visual data [Kjellström et al., 2011; Moldovan et al., 2012; Koppula et al., 2013; Koppula; Saxena, 2014].

Conclusions

Conclusions

The first step in successfully embarking on the voyage of scientific inquiry is often choosing the right level of abstraction for one's problem. Finding the right level of abstractions that fit one's problem setting is of tremendous help for finding the correct answer and even more importantly asking the right question in the first place. Of course, the scientific process of finding abstractions, questions, and answers is not linear in nature but iterative.

In this thesis we investigated probabilistic modeling with discrete and continuous random variables at three levels of abstractions.

1. the microscopic level
2. the macroscopic level
3. the cognitive level

These three different levels of abstraction allowed us to ask different questions and give answers adequate to the level of abstraction that we were operating at. We would like to stress that even though we present these three instances of abstraction, question, and answer in a linear fashion, arriving at this formulation took many iterations and revisions.

For the remainder of this concluding chapter, we will first briefly summarize the contributions presented in the thesis along the three levels of abstraction. Secondly, we will give an outlook on future work that has not yet been mentioned in intermediate concluding chapters.

Summary

Inference at the Microscopic Level

In the first part of the thesis we focused, on the one hand, on a clean and stringent formulation of weighted model integration, which resulted in the introduction of the λ-SMT problem. This allowed us to compare the zoo of existing weighted model integration solvers on a conceptual level.

On the other hand, we developed a series of state-of-the-art weighted model integration solvers, exact as well as approximate (summarized in Table 9.1). To this end we combined techniques from various different fields of computer science and artificial intelligence, such as knowledge compilation, satisfiability modulo theories, symbolic computer algebra systems, Monte Carlo techniques, dynamic programming, and GPU parallelization.

Table 9.1: Summary of developed WMI solvers.

	exact	approximate
Symbo	✓	
Sampo		✓
F-XSDD(BR)	✓	
F-XSDD(MCAD)		✓

A Probabilistic Logic Programming Language at the Macroscopic Level

The second part is concerned with extending the probabilistic logic programming language ProbLog with continuous random variables. With DC-ProbLog we introduced a probabilistic logic programming language in the discrete-continuous domain with a strict superset of ProbLog. More concretely, we made the following contributions.

Furthermore, we linked probabilistic inference in DC-ProbLog back to techniques that we developed for weighted model integration. As a consequence, inference in DC-ProbLog is reduced to weighted model integration, which constitutes a compilation of DC-ProbLog programs to weighted SMT formulas.

1. We introduced a type system for DC-ProbLog, which allows us to introduce a neat and clean syntax to extend ProbLog with function symbols for continuous random variables.

2. We sketched a purely declarative semantics for DC-ProbLog based on distribution semantics.
3. We reduced inference in DC-ProbLog to weighted model integration in the algebraic model counting setting.
4. We presented an implementation of DC-ProbLog, which reduces naturally to ProbLog in the absence of function symbols.

The presented inference algorithm is the first algorithm for the discrete-continuous domain that uses directed acyclic graphs instead of trees as an underlying data structure – potentially leading to exponentially faster inference times.

Finalizing the specification of the semantics of DC-ProbLog remains to be the last considerable task to accomplish.

Intelligent Agents at the Cognitive Level

In the third part we demonstrated that ideas from cognitive robotics and probabilistic programming can successfully be combined. We developed a cognitive robotics system that is capable of resolving intricate object occlusion scenarios in the presence of a considerable amount of uncertainty. We introduced a systems architecture that combines perceptual anchoring and probabilistic programming.

Additionally, we showed that techniques from statistical relational learning can be deployed in such a probabilistic cognitive system. This means that, instead of explicitly providing a model that describes an autonomous agent's uncertainty of the world, such a model can also be learned from data.

Our probabilistic perceptual anchoring system constitutes a neuro-symbolic approach for tackling the challenging problem of equipping autonomous agents that have to perceive the real world through raw sensor data, while at the same time being capable of reasoning about the world and the objects present.

Future Work

While the intermediate concluding chapters did mention future work, those were of a rather technical nature. We would like now to briefly mention possible future work with a broader scope, with a more visionary taste to it.

Approximate Weighted Model Integration

Solving weighted model integration problems exactly is a computationally hard task and trying to develop efficient, general and exact inference algorithms is hopeless. In the first part of the thesis we proposed, inter alia, two approximate inference algorithms. This constitutes, however, only a first step. An interesting avenue for future research would be to further explore approximate inference techniques. The goal would be, on the one hand, to adapt techniques for approximate weighted model counting and on the other hand techniques from approximate Monte Carlo estimation. In the former case, first promising results have already been presented by Belle et al. [2015b] and Abboud et al. [2020]. In the latter case, our own efforts and the work by Afshar et al. [2016] constitute first steps in this direction. What is left is to further explore the space of approximate WMI inference algorithms and combine approximation schemes that originate in these two different schools of thought.

Probabilistic Programming and Compilation

Modern high-performance programming languages rely heavily on compiling down code written in a high-level language such as C++ or Julia to efficient machine code. Nowadays, compilers do generally *write* more efficient machine code than trained experts. This is made possible through compiler toolchain technologies such as LLVM or GCC.

Even though first attempts exist to develop compilers for probabilistic programming, a holistic approach to compilation for probabilistic programming is as of now an open question. In the second part of this thesis we introduced one specific compilation technique: rewriting DC-ProbLog programs to weighted model integration problems and perform probabilistic inference for the latter. Further, complementary techniques exist, see for example [Li et al., 2015] or [Huang et al., 2017]. However, a lot of work lies still ahead before producing a modern and mature compiler toolchain for probabilistic programs. It seems unavoidable that the probabilistic programming community and the programming languages community will need to join forces on this endeavor.

In this context it is noteworthy that there has recently also been a surge in work that aims at compiling down data structures, which represent probability distribution, to specific hardware designs. such as FPGAs [Sommer et al., 2018], tailor made chips [Shah et al., 2020], or GPUs [Peharz et al., 2020]. Actually running probabilistic inference on silicon can be considered as an additional level of abstraction, a *physical* level.

Dynamic Probabilistic Programming

In the third part of the thesis we encountered the the probabilistic programming language Dynamic Distributional Clauses (DDC) [Nitti et al., 2013], which allows users to explicitly model *time*. Given the importance of time in modeling the world, it should not come as a surprise that also other probabilistic programming languages have been developed that do also allow one to model time explicitly, either as discrete time steps such as Dynamic BLOG [Erol et al., 2013], or as continuous time as in CTPL [Pfeffer, 2009].

Performing inference in a probabilistic programming language with discrete and continuous random variables, which has native support for time, is an exceptionally hard problem. Therefore, specific probabilistic programming languages usually tackle specific problems for which they provide efficient inference algorithms. Consequentially, the implementation of probabilistic programming languages ends often up to be rather tightly linked to the provided inference algorithms.

Future work could consists of disentangling the specifications of a high-level temporal probabilistic programming language and the underlying inference algorithms, rendering the high-level language agnostic of the deployed inference mechanisms. A promising research avenue to this end seems to be investigating and combining already existing ideas such as the structural interface algorithm [Vlasselaer et al., 2016] and ordinary/partial/stochastic differential equations in a setting with discrete and continuous random variables.

Neuro-Symbolic Cognitive Robotics

Although neuro-symbolic techniques [Garcez et al., 2019], which aim at bridging the symbolic/sub-symbolic divide, have experienced a recent uptake in interest from artificial intelligence researchers, these techniques have so far only been applied to (cognitive) robotics in a limited fashion. Our work on probabilistic perceptual anchoring, presented in the third part of this theses, is a first step in the direction of exploring neuro-symbolic AI being deployed to cognitive robotics. Recent work by Manhaeve et al. [2018] that combines SRL and neural methods might lead to a *fully differentiable perceptual anchoring system*. This would allow us to propagate back symbolic information to sub-symbolic representations of the world.

Opportunities for future research are plentiful and span from learning, to planning, to actually building cognitive robots. At this point, we would especially like to stress the challenge of building cognitive robotics systems. At the moment the challenge in cognitive robotics is not to be found on the theoretical side, producing yet an other theoretical cognitive robotics framework, but getting started with delivering actual

systems that interact with the world through raw and noisy data. We presented one possible research avenue to follow but a lot of work has yet to happen.

Bibliography

Abadi, Martıén; Barham, Paul; Chen, Jianmin; Chen, Zhifeng; Davis, Andy; Dean, Jeffrey; Devin, Matthieu; Ghemawat, Sanjay; Irving, Geoffrey; Isard, Michael, et al. (2016). Tensorflow: A system for large-scale machine learning. In: *12th USENIX Symposium on Operating Systems Design and Implementation OSDI16)*.

Abboud, Ralph; Ceylan, Ismail Ilkan; Dimitrov, Radoslav (2020). On the Approximability of Weighted Model Integration on DNF Structures. In: *arXiv preprint arXiv:2002.06726*.

Adé, Hilde; De Raedt, Luc; Bruynooghe, Maurice (1995). Declarative bias for specific-to-general ILP systems. In: *Machine Learning* 20.1-2, pp. 119–154.

Afshar, H.; Sanner, S.; Abbasnejad, Ehsan (2015). Linear-Time Gibbs Sampling in Piecewise Graphical Models. In: *AAAI*.

Afshar, Hadi Mohasel; Domke, Justin (2015). Reflection, Refraction, and Hamiltonian Monte Carlo. In: *Advances in neural information processing systems*, pp. 3007–3015.

Afshar, Hadi Mohasel; Sanner, Scott; Webers, Christfried (2016). Closed-form Gibbs sampling for graphical models with algebraic constraints. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Antanas, Laura; Can, Ozan Arkan; Davis, Jesse; De Raedt, Luc; Loutfi, Amy; Persson, Andreas; Saffiotti, Alessandro; Unal, Emre; Yuret, Deniz; Martires, Pedro Zuidberg dos (2017). Relational Symbol Grounding through Affordance Learning: an Overview of the ReGround Project. In: *International Workshop on Grounding Language Understanding @ INTERSPEECH*.

Barrett, Clark; Tinelli, Cesare (2018). Satisfiability modulo theories. In: *Handbook of Model Checking*. Springer.

Belle, Vaishak; Passerini, Andrea; Van den Broeck, Guy (2015a). Probabilistic Inference in Hybrid Domains by Weighted Model Integration. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Belle, Vaishak; Van den Broeck, Guy; Passerini, Andrea (2015b). Hashing-based approximate probabilistic inference in hybrid domains. In: *In P*.

Belle, Vaishak; Van den Broeck, Guy; Passerini, Andrea (2016). Component Caching in Hybrid Domains with Piecewise Polynomial Densities. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Bellman, Richard (1957). *Dynamic Programming*. Princeton University Press.

Beltagy, Islam; Roller, Stephen; Cheng, Pengxiang; Erk, Katrin; Mooney, Raymond J (2016). Representing meaning with a combination of logical and distributional models. In: *Computational Linguistics* 42.4, pp. 763–808.

Bezanson, Jeff; Edelman, Alan; Karpinski, Stefan; Shah, Viral B (2017). Julia: A fresh approach to numerical computing. In: *SIAM review*.

Bingham, Eli; Chen, Jonathan P.; Jankowiak, Martin; Obermeyer, Fritz; Pradhan, Neeraj; Karaletsos, Theofanis; Singh, Rohit; Szerlip, Paul; Horsfall, Paul; Goodman, Noah D. (2018). Pyro: Deep Universal Probabilistic Programming. In: *Journal of Machine Learning Research*.

Birnbaum, Elazar; Lozinskii, Eliezer L (1999). The Good Old Davis-Putnam Procedure Helps Counting Models. In: *Journal of Artificial Intelligence Research*.

Blodow, Nico; Jain, Dominik; Marton, Zoltan-Csaba; Beetz, Michael (2010). Perception and probabilistic anchoring for dynamic world state logging. In: *2010 10th IEEE-RAS International Conference on Humanoid Robots*. IEEE.

Box, George EP (1976). Science and statistics. In: *Journal of the American Statistical Association* 71.356, pp. 791–799.

Box, George EP (1979). Robustness in the strategy of scientific model building. In: *Robustness in statistics*. Elsevier, pp. 201–236.

Bryant, Randal E. (1986). Graph-Based Algorithms for Boolean Function Manipulation. In: *IEEE Transactions on Computers*.

Burges, Christopher JC (1998). A tutorial on support vector machines for pattern recognition. In: *Data mining and knowledge discovery* 2.2, pp. 121–167.

Can, Ozan Arkan; Zuidberg Dos Martires, Pedro; Persson, Andreas; Gaal, Julian; Loutfi, Amy; De Raedt, Luc; Yuret, Deniz; Saffiotti, Alessandro (2019). Learning from Implicit Information in Natural Language Instructions for Robotic Manipulations. In: *Combined Workshop on Spatial Language Understanding and Grounded Communication for Robotics @ NAACL.*

Carpenter, Bob; Gelman, Andrew; Hoffman, Matthew D; Lee, Daniel; Goodrich, Ben; Betancourt, Michael; Brubaker, Marcus; Guo, Jiqiang; Li, Peter; Riddell,

Allen (2017). Stan: A probabilistic programming language. In: *Journal of statistical software* 76.1.

Castagna, Giuseppe; Ghelli, Giorgio; Longo, Giuseppe (1995). A calculus for overloaded functions with subtyping. In: *Information and Computation* 117.1, pp. 115–135.

Ceylan, Ismail Ilkan; Darwiche, Adnan; Van den Broeck, Guy (2016). Open-World Probabilistic Databases. In: *Description Logics*. Citeseer.

Chavira, Mark; Darwiche, Adnan (2008). On probabilistic inference by weighted model counting. In: *Artificial Intelligence*.

Chella, Antonio; Coradeschi, Silvia; Frixione, Marcello; Saffiotti, Alessandro (2004). Perceptual anchoring via conceptual spaces. In: *Proceedings of the AAAI workshop on anchoring symbols to sensor data*.

Chistikov, Dmitry; Dimitrova, Rayna; Majumdar, Rupak (2015). Approximate counting in SMT and value estimation for probabilistic programs. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer.

Choi, Arthur; Darwiche, Adnan (2013). Dynamic Minimization of Sentential Decision Diagrams. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Choi, Arthur; Kisa, Doga; Darwiche, Adnan (2013). Compiling probabilistic graphical models using sentential decision diagrams. In: *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*. Springer.

Cimatti, Alessandro; Griggio, Alberto; Schaafsma, Bastiaan Joost; Sebastiani, Roberto (2013). The mathsat5 SMT Solver. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*.

Coradeschi, Silvia; Saffiotti, Alessandro (2000). Anchoring symbols to sensor data: preliminary report. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 129–135.

Coradeschi, Silvia; Saffiotti, Alessandro (2001). Perceptual anchoring of symbols for action. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 407–416.

Coradeschi, Silvia; Saffiotti, Alessandro (2003). An introduction to the anchoring problem. In: *Robotics and autonomous systems*.

Cousins, Ben; Vempala, Santosh (2016). A practical volume algorithm. In: *Mathematical Programming Computation*.

Daoutis, Marios; Coradeschi, Silvia; Loutfi, Amy (2012). Cooperative knowledge based perceptual anchoring. In: *International Journal on Artificial Intelligence Tools*.

Darwiche, Adnan (1999). Compiling knowledge into decomposable negation normal form. In: *Proceedings of the 16th international joint conference on Artifical intelligence*.

Darwiche, Adnan (2001). On the tractable counting of theory models application to truth maintenance and belief revision. In: *Journal of Applied Non-Classical Logics*.

Darwiche, Adnan (2003). A differential approach to inference in Bayesian networks. In: *Journal of the ACM (JACM)*.

Darwiche, Adnan; Marquis, Pierre (2002). A knowledge compilation map. In: *Journal of Artificial Intelligence Research*.

De Campos, Cassio Polpo; Cozman, Fabio Gagliardi (2005). The inferential complexity of Bayesian and credal networks. In: *IJCAI*. Vol. 5. Citeseer, pp. 1313–1318.

De Loera, Jesús A; Dutra, Brandon; Koeppe, Matthias; Moreinis, Stanislav; Pinto, Gregory; Wu, Jianqiu (2013a). Software for exact integration of polynomials over polyhedra. In: *Computational Geometry*.

De Loera, Jesus; Dutra, Brandon; Koeppe, Matthias; Moreinis, Stanislav; Pinto, Gregory; Wu, Jianqiu (2013b). Software for Exact Integration of Polynomials over Polyhedra. In: *Comput. Geom.*

De Moura, Leonardo; Bjørner, Nikolaj (2008). Z3: An efficient SMT solver. In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer.

De Raedt, Luc; Dumančić, Sebastijan; Manhaeve, Robin; Marra, Giuseppe (2020). From Statistical Relational to Neuro-Symbolic Artificial Intelligence. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*.

De Raedt, Luc; Kersting, Kristian; Natarajan, Sriraam; Poole, David (2016). Statistical relational artificial intelligence: Logic, probability, and computation. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 10.2, pp. 1–189.

De Raedt, Luc; Kimmig, Angelika (2015). Probabilistic (logic) programming concepts. In: *Machine Learning*.

De Raedt, Luc; Kimmig, Angelika; Toivonen, Hannu (2007). ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. 7. Hyderabad, pp. 2462–2467.

De Salvo Braz, Rodrigo; O'Reilly, Ciaran; Gogate, Vibhav; Dechter, Rina (2016). Probabilistic Inference Modulo Theories. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Deng, Jia; Dong, Wei; Socher, Richard; Li, Li-Jia; Li, Kai; Fei-Fei, Li (2009). Imagenet: A large-scale hierarchical image database. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Derkinderen, Vincent; Heylen, Evert; Pedro, Zuidberg Dos Martires; Kolb, Samuel; De Raedt, Luc (2020). Ordering Variables for Weighted Model Integration. In: *Proceedings of the Uncertainty in Artificial Intelligence (UAI) Conference*.

Dries, Anton; Kimmig, Angelika; Meert, Wannes; Renkens, Joris; Van den Broeck, Guy; Vlasselaer, Jonas; De Raedt, Luc (2015). Problog2: Probabilistic logic programming. In: *Joint european conference on machine learning and knowledge discovery in databases*. Springer.

Dyer, Martin E.; Frieze, Alan M. (1988). On the complexity of computing the volume of a polyhedron. In: *SIAM Journal on Computing* 17.5, pp. 967–974.

Dyer, Martin; Frieze, Alan; Kannan, Ravi (1991). A random polynomial-time algorithm for approximating the volume of convex bodies. In: *Journal of the ACM (JACM)*.

Elfring, Jos; Dries, Sjoerd van den; Van De Molengraft, MJG; Steinbuch, Maarten (2013). Semantic world modeling using probabilistic multiple hypothesis anchoring. In: *Robotics and Autonomous Systems*.

Emiris, Ioannis Z; Fisikopoulos, Vissarion (2014). Efficient random-walk methods for approximating polytope volume. In: *Proceedings of the thirtieth annual symposium on Computational geometry*. ACM.

Emiris, Ioannis Z; Fisikopoulos, Vissarion (2018). Practical polytope volume approximation. In: *ACM Transactions on Mathematical Software (TOMS)*.

Erol, Yusuf Bugra; Li, Lei; Ramsundar, Bharath; Stuart, Russell (2013). The extended parameter filter. In: *International Conference on Machine Learning*, pp. 1103–1111.

Fierens, Daan; Van den Broeck, Guy; Renkens, Joris; Shterionov, Dimitar; Gutmann, Bernd; Thon, Ingo; Janssens, Gerda; De Raedt, Luc (2015). Inference and learning in probabilistic logic programs using weighted Boolean formulas. In: *Theory and Practice of Logic Programming*.

Friedman, Nir; Getoor, Lise; Koller, Daphne; Pfeffer, Avi (1999). Learning probabilistic relational models. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. 99, pp. 1300–1309.

Fubini, Guido (1907). Sugli integrali multipli. In: *Rend. Acc. Naz. Lincei*.

Fukunaga, Keinosuke (2013). *Introduction to statistical pattern recognition*. Elsevier.

Fung, Robert; Chang, Kuo-Chu (1990). Weighing and integrating evidence for stochastic simulation in Bayesian networks. In: *Machine Intelligence and Pattern Recognition*. Elsevier.

Gao, Wei; Lv, Hengyi; Zhang, Qiang; Cai, Dunbo (2018). Estimating the Volume of the Solution Space of SMT (LIA) Constraints by a Flat Histogram Method. In: *Algorithms*.

Garcez, Artur d'Avila; Gori, Marco; Lamb, Luis C; Serafini, Luciano; Spranger, Michael; Tran, Son N (2019). Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. In: *Journal of Applied Logics*.

Gardner, Matt; Talukdar, Partha; Krishnamurthy, Jayant; Mitchell, Tom (2014). Incorporating vector space similarity in random walk inference over knowledge bases. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 397–406.

Ge, Hong; Xu, Kai; Ghahramani, Zoubin (2018). Turing: A language for flexible probabilistic inference. In:

Gehr, Timon; Misailovic, Sasa; Vechev, Martin (2016). PSI: Exact symbolic inference for probabilistic programs. In: *International Conference on Computer Aided Verification*. Springer.

Getoor, Lise (2013). Probabilistic soft logic: a scalable approach for markov random fields over continuous-valued variables. In: *International Workshop on Rules and Rule Markup Languages for the Semantic Web*. Springer, pp. 1–1.

Getoor, Lise; Taskar, Ben (2007). Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning). In:

Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016). *Deep learning*. MIT press.

Goodman, Noah D; Mansinghka, Vikash K; Roy, Daniel; Bonawitz, Keith; Tenenbaum, Joshua B (2008). Church: a language for generative models. In: *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 220–229.

Graf, Susanne; Saïdi, Hassen (1997). Construction of Abstract State Graphs with PVS. In: *International Conference on Computer Aided Verification*. Springer.

Günther, Martin; Ruiz-Sarmiento, JR; Galindo, Cipriano; Gonzalez-Jimenez, Javier; Hertzberg, Joachim (2018). Context-aware 3D object anchoring for mobile robots. In: *Robotics and Autonomous Systems*.

Gutmann, Bernd; Jaeger, Manfred; De Raedt, Luc (2010). Extending ProbLog with continuous distributions. In: *International Conference on Inductive Logic Programming*. Springer.

Gutmann, Bernd; Thon, Ingo; Kimmig, Angelika; Bruynooghe, Maurice; De Raedt, Luc (2011). The magic of logical inference in probabilistic programming. In: *Theory and Practice of Logic Programming*.

Gyenis, Zalán; Hofer-Szabó, Gábor; Rédei, Miklós (2017). Conditioning using conditional expectations: the Borel–Kolmogorov Paradox. In: *Synthese*.

Hailperin, Theodore et al. (1984). Probability logic. In: *Notre Dame Journal of Formal Logic*.

Heunen, Chris; Kammar, Ohad; Staton, Sam; Yang, Hongseok (2017). A convenient category for higher-order probability theory. In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, pp. 1–12.

Holtzen, Steven; Broeck, Guy Van den; Millstein, Todd (2020). Dice: Compiling Discrete Probabilistic Programs for Scalable Inference. In: *arXiv preprint arXiv:2005.09089*.

Holzer, Stefan; Rusu, Radu Bogdan; Dixon, Michael; Gedikli, Suat; Navab, Nassir (2012). Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 2684–2689.

Howson, Colin (2003). Probability and logic. In: *Journal of Applied Logic*.

Huang, Daniel; Tristan, Jean-Baptiste; Morrisett, Greg (2017). Compiling Markov chain Monte Carlo algorithms for probabilistic modeling. In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 111–125.

Huang, Jinbo; Darwiche, Adnan (2005). DPLL with a Trace: from SAT to Knowledge Compilation. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Hüllermeier, Eyke; Waegeman, Willem (2019). Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction. In: *arXiv preprint arXiv:1910.09457*.

Islam, Muhammad Asiful; Ramakrishnan, CR; Ramakrishnan, IV (2012). Inference in Probabilistic Logic Programs with Continuous Random Variables. In: *Theory and Practice of Logic Programming*.

Iverson, Kenneth E (1962). A programming language. In: *Proceedings of the May 1-3, 1962, spring joint computer conference*.

Jackson, Peter (1998). *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc.

Kadane, Joseph B (2011). *Principles of uncertainty*. CRC Press.

Keene, Sonya E (1989). Object-Oriented programming in Common Lisp; a programmer's guide to clos. Tech. rep.

Khoreva, Anna; Benenson, Rodrigo; Hosang, Jan; Hein, Matthias; Schiele, Bernt (2017). Simple does it: Weakly supervised instance and semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Kimmig, Angelika; Van den Broeck, Guy; De Raedt, Luc (2011). An algebraic Prolog for reasoning about possible worlds. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Kimmig, Angelika; Van den Broeck, Guy; De Raedt, Luc (2017). Algebraic model counting. In: *Journal of Applied Logic*.

Kjellström, Hedvig; Romero, Javier; Kragić, Danica (2011). Visual object-action recognition: Inferring object affordances from human demonstration. In: *Computer Vision and Image Understanding*.

Knuth, Donald E. (1992). Two Notes on Notation. In: *American Mathematical Monthly*.

Kolb, Samuel; Mladenov, Martin; Sanner, Scott; Belle, Vaishak; Kersting, Kristian (2018). Efficient Symbolic Integration for Probabilistic Inference. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Kolb, Samuel; Morettin, Paolo; Zuidberg Dos Martires, Pedro; Sommavilla, Francesco; Passerini, Andrea; Sebastiani, Roberto; De Raedt, Luc (2019a). The pywmi Framework and Toolbox for Probabilistic Inference using Weighted Model Integration. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Kolb, Samuel; Zuidberg Dos Martires, Pedro; De Raedt, Luc (2019b). How to Exploit Structure while Solving Weighted Model Integration Problems. In: *Proceedings of the Uncertainty in Artificial Intelligence (UAI) Conference*.

Koller, Daphne; Friedman, Nir (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Kolmogorov, AN (1950). Foundations of the theory of probability. In:

Koppula, Hema S; Saxena, Ashutosh (2014). Physically grounded spatio-temporal object affordances. In: *European Conference on Computer Vision*. Springer, pp. 831–847.

Koppula, Hema Swetha; Gupta, Rudhir; Saxena, Ashutosh (2013). Learning human activities and object affordances from rgb-d videos. In: *The International Journal of Robotics Research*.

Koutis, Ioannis (2003). On the hardness of approximate multivariate integration. In: *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*. Springer, pp. 122–128.

Kuhn, Harold W (1955). The Hungarian method for the assignment problem. In: *Naval research logistics quarterly*.

Kumar, Nitesh; Kuzelka, Ondrej; De Raedt, Luc (2020). Learning Distributional Programs for Relational Autocompletion. In: *arXiv:2001.08603*.

Lafitte, Frédéric (2018). CryptoSAT: a tool for SAT-based cryptanalysis. In: *IET Information Security*.

LeBlanc, Kevin (2010). Cooperative anchoring: sharing information about objects in multi-robot systems. PhD thesis. Örebro universitet.

LeBlanc, Kevin; Saffiotti, Alessandro (2008). Cooperative anchoring in heterogeneous multi-robot systems. In: *2008 IEEE International Conference on Robotics and Automation*. IEEE.

Levesque, Hector J (1986). Knowledge representation and reasoning. In: *Annual review of computer science*.

Li, Lei; Wu, Yi; Russell, Stuart J (2015). SWIFT: Compiled inference for probabilistic programs. In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-12*.

Long, Jonathan; Shelhamer, Evan; Darrell, Trevor (2015). Fully convolutional networks for semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440.

Loutfi, Amy (2006). *Odour recognition using electronic noses in robotic and intelligent systems*. Universitetsbiblioteket.

Loutfi, Amy; Coradeschi, Silvia (2006). Smell, think and act: A cognitive robot discriminating odours. In: *Autonomous Robots*.

Loutfi, Amy; Coradeschi, Silvia; Daoutis, Marios; Melchert, Jonas (2008). Using knowledge representation for perceptual anchoring in a robotic system. In: *International Journal on Artificial Intelligence Tools*.

Loutfi, Amy; Coradeschi, Silvia; Saffiotti, Alessandro (2005). Maintaining coherent perceptual information using anchoring. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Lovász, László; Vempala, Santosh (2006). Fast algorithms for logconcave functions: Sampling, rounding, integration and optimization. In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, pp. 57–68.

Ma, Feifei; Liu, Sheng; Zhang, Jian (2009). Volume computation for boolean combination of linear arithmetic constraints. In: *International Conference on Automated Deduction*. Springer.

Manfredotti, Cristina (2009). Modeling and inference with relational dynamic Bayesian networks. In: *Canadian Conference on Artificial Intelligence*. Springer, pp. 287–290.

Manhaeve, Robin; Dumancic, Sebastijan; Kimmig, Angelika; Demeester, Thomas; De Raedt, Luc (2018). Deepproblog: Neural probabilistic logic programming. In: *Advances in Neural Information Processing Systems*, pp. 3749–3759.

Meshgi, Kourosh; Ishii, Shin (2015). The state-of-the-art in handling occlusions for visual object tracking. In: *IEICE TRANSACTIONS on Information and Systems*.

Michels, Steffen; Hommersom, Arjen; Lucas, Peter J. F. (2016). Approximate Probabilistic Inference with Bounded Error for Hybrid Probabilistic Logic Programming. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Milch, Brian; Marthi, Bhaskara; Russell, Stuart; Sontag, David; Ong, Daniel L; Kolobov, Andrey (2005). BLOG: Probabilistic models with unknown objects. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Milch, Brian; Russell, Stuart (2006). General-Purpose MCMC inference over relational structures. In: *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pp. 349–358.

Minka, Tom; Winn, John; Guiver, J; Webster, S; Zaykov, Y; Yangel, B; Spengler, A; Bronskill, J (2014). Infer .NET 2.6. Microsoft Research Cambridge. In: *URL http://research. microsoft. com/infernet*.

Miosic, Ivan; Zuidberg Dos Martires, Pedro (2020). Measure Theoretic Weighted Model Integration. In: *(in preparation)*.

Moldovan, Bogdan; Moreno, Plinio; Van Otterlo, Martijn; Santos-Victor, José; De Raedt, Luc (2012). Learning relational affordance models for robots in multi-object manipulation tasks. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE, pp. 4373–4378.

Molina, Alejandro; Vergari, Antonio; Di Mauro, Nicola; Natarajan, Sriraam; Esposito, Floriana; Kersting, Kristian (2018). Mixed sum-product networks: A deep architecture for hybrid domains. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Morettin, Paolo; Kolb, Samuel; Teso, Stefano; Passerini, Andrea (2020). Learning Weighted Model Integration Distributions. In: *AAAI*, pp. 5224–5231.

Morettin, Paolo; Passerini, Andrea; Sebastiani, Roberto (2017). Efficient Weighted Model Integration via SMT-Based Predicate Abstraction. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Mösenlechner, Lorenz; Beetz, Michael (2009). Using Physics-and Sensor-based Simulation for High-Fidelity Temporal Projection of Realistic Robot Behavior. In: *ICAPS*.

Muggleton, Stephen; De Raedt, Luc (1994). Inductive logic programming: Theory and methods. In: *The Journal of Logic Programming*.

Narayanan, Praveen; Carette, Jacques; Romano, Wren; Shan, Chung-chieh; Zinkov, Robert (2016). Probabilistic Inference by Program Transformation in Hakaru (System Description). In: *International Symposium on Functional and Logic Programming - 13th International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings*. Springer.

Neville, Jennifer; Jensen, David (2007). Relational dependency networks. In: *Journal of Machine Learning Research* 8.Mar, pp. 653–692.

Nilsson, Nils J (1994). Probabilistic logic revisited. In: *Artificial intelligence* 59.1-2, pp. 39–42.

Nilsson, Nils J. (1986). Probabilistic logic. In:

Nitti, Davide; De Laet, Tinne; De Raedt, Luc (2013). A particle filter for hybrid relational domains. In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, pp. 2764–2771.

Nitti, Davide; De Laet, Tinne; De Raedt, Luc (2014). Relational object tracking and learning. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, pp. 935–942.

Nitti, Davide; De Laet, Tinne; De Raedt, Luc (2016a). Probabilistic logic programming for hybrid relational domains. In: *Machine Learning*.

Nitti, Davide; Ravkic, Irma; Davis, Jesse; De Raedt, Luc (2016b). Learning the structure of dynamic hybrid relational models. In: *Proceedings of the Twenty-second European Conference on Artificial Intelligence*. IOS Press, pp. 1283–1290.

Nori, Aditya V; Hur, Chung-Kil; Rajamani, Sriram K; Samuel, Selva (2014). R2: An Efficient MCMC Sampler for Probabilistic Programs. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Oh, Songhwai; Russell, Stuart; Sastry, Shankar (2009). Markov chain Monte Carlo data association for multi-target tracking. In: *IEEE Transactions on Automatic Control*.

Oztok, Umut; Darwiche, Adnan (2018). An Exhaustive DPLL Algorithm for Model Counting. In: *Journal of Artificial Intelligence Research*.

Park, James D; Darwiche, Adnan (2004). Complexity results and approximation strategies for MAP explanations. In: *Journal of Artificial Intelligence Research* 21, pp. 101–133.

Parzen, Emanuel (1962). On estimation of a probability density function and mode. In: *The annals of mathematical statistics*.

Pearl, Judea (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier.

Peharz, Robert; Lang, Steven; Vergari, Antonio; Stelzner, Karl; Molina, Alejandro; Trapp, Martin; Van den Broeck, Guy; Kersting, Kristian; Ghahramani, Zoubin (2020). Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits. In: *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

Peng, Xue Bin; Andrychowicz, Marcin; Zaremba, Wojciech; Abbeel, Pieter (2018). Sim-to-real transfer of robotic control with dynamics randomization. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 1–8.

Persson, Andreas (2019). Studies in Semantic Modeling of Real-World Objects using Perceptual Anchoring. PhD thesis. Örebro University.

Persson, Andreas; Längkvist, Martin; Loutfi, Amy (2017). Learning actions to improve the perceptual anchoring of objects. In: *Frontiers in Robotics and AI*.

Persson, Andreas; Zuidberg Dos Martires, Pedro; De Raedt, Luc; Loutfi, Amy (2020a). ProbAnch: a Modular Probabilistic Anchoring Framework. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Persson, Andreas; Zuidberg Dos Martires, Pedro; Loutfi, Amy; De Raedt, Luc (2020b). Semantic Relational Object Tracking. In: *IEEE Transactions on Cognitive and Developmental Systems* 12.1, pp. 84–97.

Pfeffer, Avi (2009). CTPPL: A continuous time probabilistic programming language. In: *Twenty-First International Joint Conference on Artificial Intelligence*.

Phan, Quoc-Sang; Malacaria, Pasquale (2014). Abstract model counting: a novel approach for quantification of information leaks. In: *Proceedings of the 9th ACM symposium on Information, computer and communications security*.

Poole, David (1993). Probabilistic Horn abduction and Bayesian networks. In: *Artificial intelligence* 64.1, pp. 81–129.

Poole, David (1997). The independent choice logic for modelling multiple agents under uncertainty. In: *Artificial intelligence*.

Poole, David (2010). Probabilistic programming languages: Independent choices and deterministic systems. In: *Heuristics, probability and causality: A tribute to Judea Pearl*, pp. 253–269.

Przymusinski, Teodor C (1988). Perfect Model Semantics. In: *ICLP/SLP*. Vol. 88, pp. 1081–1096.

Rainforth, Tom; Cornish, Robert; Yang, Hongseok; Warrington, Andrew (2018). On Nesting Monte Carlo Estimators. In: *ICML*.

Ram, Parikshit; Gray, Alexander G (2011). Density estimation trees. In: *KDD*. ACM.

Ravkic, Irma; Ramon, Jan; Davis, Jesse (2015). Learning relational dependency networks in hybrid domains. In: *Machine Learning* 100.2-3, pp. 217–254.

Reid, Donald (1979). An algorithm for tracking multiple targets. In: *IEEE transactions on Automatic Control*.

Richardson, Matthew; Domingos, Pedro (2006). Markov logic networks. In: *Machine learning* 62.1-2, pp. 107–136.

Riguzzi, Fabrizio (2018). *Foundations of Probabilistic Logic Programming*. River Publishers.

Riguzzi, Fabrizio; Swift, Terrance (2013). Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics. In: *Theory and practice of logic programming* 13.2, p. 279.

Rosenblatt, Murray (1956). Remarks on some nonparametric estimates of a density function. In: *The Annals of Mathematical Statistics*.

Ruiz-Sarmiento, Jose-Raul; Günther, Martin; Galindo, Cipriano; González-Jiménez, Javier; Hertzberg, Joachim (2017). Online context-based object recognition for mobile robots. In: *2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE.

Russakovsky, Olga; Deng, Jia; Su, Hao; Krause, Jonathan; Satheesh, Sanjeev; Ma, Sean; Huang, Zhiheng; Karpathy, Andrej; Khosla, Aditya; Bernstein, Michael, et al. (2015). Imagenet large scale visual recognition challenge. In: *International journal of computer vision* 115.3, pp. 211–252.

Russell, Stuart (2015). Unifying logic and probability. In: *Communications of the ACM* 58.7, pp. 88–97.

Rusu, Radu Bogdan; Cousins, Steve (2011). 3d is here: Point cloud library (pcl). In: *2011 IEEE international conference on robotics and automation*. IEEE, pp. 1–4.

Salvatier, John; Wiecki, Thomas V; Fonnesbeck, Christopher (2016). Probabilistic programming in Python using PyMC3. In: *PeerJ Computer Science* 2, e55.

Sanner, Scott; Abbasnejad, Ehsan (2012). Symbolic variable elimination for discrete and continuous graphical models. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Sanner, Scott; Delgado, Karina Valdivia; Barros, Leliane Nunes de (2011). Symbolic dynamic programming for discrete and continuous state MDPs. In: *Proceedings of the Uncertainty in Artificial Intelligence (UAI) Conference*.

Sato, Taisuke (1995). A Statistical Learning Method for Logic Programs with Distribution Semantics. In: *In proceedings of the 12TH international conference on logic programming (ICLP'95*. Citeseer.

Sato, Taisuke; Kameya, Yoshitaka (1997). PRISM: a language for symbolic-statistical modeling. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Sato, Taisuke; Kameya, Yoshitaka (2001). Parameter learning of logic programs for symbolic-statistical modeling. In: *Journal of Artificial Intelligence Research* 15, pp. 391–454.

Schrijvers, Tom; Costa, Vítor Santos; Wielemaker, Jan; Demoen, Bart (2008). Towards Typed Prolog. In: *ICLP*.

Shah, Nimish; Olascoaga, Laura I Galindez; Meert, Wannes; Verhelst, Marian (2020). Acceleration of probabilistic reasoning through custom processor architecture. In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 322–325.

Shan, Chung-chieh; Ramsey, Norman (2017). Exact Bayesian inference by symbolic disintegration. In: *ACM SIGPLAN Notices*. ACM.

Silver, David; Huang, Aja; Maddison, Chris J; Guez, Arthur; Sifre, Laurent; Van Den Driessche, George; Schrittwieser, Julian; Antonoglou, Ioannis; Panneershelvam, Veda; Lanctot, Marc, et al. (2016). Mastering the game of Go with deep neural networks and tree search. In: *nature*.

Smeulders, Arnold WM; Chu, Dung M; Cucchiara, Rita; Calderara, Simone; Dehghan, Afshin; Shah, Mubarak (2013). Visual tracking: An experimental survey. In: *IEEE transactions on pattern analysis and machine intelligence*.

Sommer, Lukas; Oppermann, Julian; Molina, Alejandro; Binnig, Carsten; Kersting, Kristian; Koch, Andreas (2018). Automatic mapping of the sum-product network

inference problem to fpga-based accelerators. In: *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE, pp. 350–357.

Sorensson, Niklas; Een, Niklas (2005). Minisat v1. 13-a sat solver with conflict-clause minimization. In: *SAT*.

Speichert, Stefanie; Belle, Vaishak (2019). Learning Probabilistic Logic Programs over Continuous Data. In: *International Conference on Inductive Logic Programming*.

Staton, Sam; Wood, Frank; Yang, Hongseok; Heunen, Chris; Kammar, Ohad (2016). Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In: *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE.

Sterling, Leon; Shapiro, Ehud Y (1994). *The art of Prolog: advanced programming techniques*. MIT press.

Szegedy, Christian; Liu, Wei; Jia, Yangqing; Sermanet, Pierre; Reed, Scott; Anguelov, Dragomir; Erhan, Dumitru; Vanhoucke, Vincent; Rabinovich, Andrew (2015). Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Taskar, Ben; Abbeel, Pieter; Koller, Daphne (2002). Discriminative probabilistic models for relational data. In: *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., pp. 485–492.

Tenorth, Moritz; Beetz, Michael (2013). KnowRob: A knowledge processing infrastructure for cognition-enabled robots. In: *The International Journal of Robotics Research* 32.5, pp. 566–590.

Tobin, Josh; Fong, Rachel; Ray, Alex; Schneider, Jonas; Zaremba, Wojciech; Abbeel, Pieter (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 23–30.

Tran, Dustin; Kucukelbir, Alp; Dieng, Adji B.; Rudolph, Maja; Liang, Dawen; Blei, David M. (2016). Edward: A Library for Probabilistic Modeling, Inference, and Criticism. In: *arXiv preprint arXiv:1610.09787*.

Trevor, Alexander JB; Gedikli, Suat; Rusu, Radu B; Christensen, Henrik I (2013). Efficient organized point cloud segmentation with connected components. In: *Semantic Perception Mapping and Exploration (SPME)*.

Valiant, Leslie G (1979). The complexity of computing the permanent. In: *Theoretical Computer Science*.

Van Gelder, Allen; Ross, Kenneth A; Schlipf, John S (1991). The well-founded semantics for general logic programs. In: *Journal of the ACM (JACM)* 38.3, pp. 619–649.

Vennekens, Joost; Verbaeten, Sofie; Bruynooghe, Maurice (2004). Logic programs with annotated disjunctions. In: *International Conference on Logic Programming*. Springer.

Vezzani, Roberto; Grana, Costantino; Cucchiara, Rita (2011). Probabilistic people tracking with appearance models and occlusion classification: The ad-hoc system. In: *Pattern Recognition Letters*.

Vlasselaer, Jonas; Meert, Wannes; Van den Broeck, Guy; De Raedt, Luc (2016). Exploiting local and repeated structure in dynamic Bayesian networks. In: *Artificial Intelligence* 232, pp. 43–53.

Wiedemeyer, Thiemo (2015). IAI kinect2. In: *Institute for artificial intelligence, University Bremen*, pp. 2014–2015.

Wong, Lawson LS; Kaelbling, Leslie Pack; Lozano-Pérez, Tomás (2015). Data association for semantic world modeling from partial views. In: *The International Journal of Robotics Research*.

Wood, Frank; Meent, Jan Willem van de; Mansinghka, Vikash (2014). A New Approach to Probabilistic Programming Inference. In: *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*.

Wu, Yi; Lim, Jongwoo; Yang, Ming-Hsuan (2013). Online object tracking: A benchmark. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Wu, Yi; Srivastava, Siddharth; Hay, Nicholas; Du, Simon; Russell, Stuart (2018). Discrete-Continuous Mixtures in Probabilistic Programming: Generalized Semantics and Inference Algorithms. In: *International Conference on Machine Learning*.

Xie, Christopher; Xiang, Yu; Mousavian, Arsalan; Fox, Dieter (2019). The Best of Both Modes: Separately Leveraging RGB and Depth for Unseen Object Instance Segmentation. In: *Conference on Robot Learning (CoRL)*.

Zeng, Zhe; Morettin, Paolo; Yan, Fanqi; Vergari, Antonio; Van den Broeck, Guy (2020). Scaling up Hybrid Probabilistic Inference with Logical and Arithmetic Constraints via Message Passing. In: *ICML*.

Zeng, Zhe; Van den Broeck, Guy (2019). Efficient Search-Based Weighted Model Integration. In: *Proceedings of the Uncertainty in Artificial Intelligence (UAI) Conference*.

Zhou, Min; He, Fei; Song, Xiaoyu; He, Shi; Chen, Gangyi; Gu, Ming (2015). Estimating the volume of solution space for satisfiability modulo linear real arithmetic. In: *Theory of Computing Systems*.

Zuidberg Dos Martires, Pedro (2019). Differentiation and Weighted Model Integration. In: *The 1st Workshop on Deep Continuous-Discrete Machine Learning@ ECML, Location: Würzburg*.

Zuidberg Dos Martires, Pedro; Derkinderen, Vincent; Manhaeve, Robin; Meert, Wannes; Kimmig, Angelika; De Raedt, Luc (2019a). Transforming Probabilistic Programs into Algebraic Circuits for Inference and Learning. In: *Program Transformations for Machine Learning @ NeurIPS 2019*.

Zuidberg Dos Martires, Pedro; Dries, Anton; De Raedt, Luc (2018). Knowledge Compilation with Continuous Random Variables and its Application in Hybrid Probabilistic Logic Programming. In: *Eigth International Workshop on Statistical Relational AI @ IJCAI*.

Zuidberg Dos Martires, Pedro; Dries, Anton; De Raedt, Luc (2019b). Exact and Approximate Weighted Model Integration with Probability Density Functions Using Knowledge Compilation. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zuidberg Dos Martires, Pedro; Dumancic, Sebastijan (2018). Reactive Probabilistic Programming. In: *The International Conference on Probabilistic Programming, Location: Boston, United States of America*.

Zuidberg Dos Martires, Pedro; Kimmig, Angelika; De Raedt, Luc (2020a). Extending ProbLog with Random Function Symbols. In: *(in preparation)*.

Zuidberg Dos Martires, Pedro; Kolb, Samuel (2020). Monte Carlo Anti-Differentiation for Approximate Weighted Model Integration. In: *Ninth International Workshop on Statistical Relational AI @ AAAI*.

Zuidberg Dos Martires, Pedro; Kumar, Nitesh; Persson, Andreas; Loutfi, Amy; De Raedt, Luc (2020b). Symbolic Learning and Reasoning with Noisy Data for Probabilistic Anchoring. In: *Frontiers in Robotics and AI* 7, p. 100.

List of Publications

Journal Papers

Persson, Andreas; Zuidberg Dos Martires, Pedro; Loutfi, Amy; De Raedt, Luc [2020b]. Semantic Relational Object Tracking. In: *IEEE Transactions on Cognitive and Developmental Systems* 12.1, pp. 84–97.

Zuidberg Dos Martires, Pedro; Kumar, Nitesh; Persson, Andreas; Loutfi, Amy; De Raedt, Luc [2020b]. Symbolic Learning and Reasoning with Noisy Data for Probabilistic Anchoring. In: *Frontiers in Robotics and AI* 7, p. 100.

Conference Papers

Zuidberg Dos Martires, Pedro; Dries, Anton; De Raedt, Luc [2019b]. Exact and Approximate Weighted Model Integration with Probability Density Functions Using Knowledge Compilation. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Kolb, Samuel; Zuidberg Dos Martires, Pedro; De Raedt, Luc [2019b]. How to Exploit Structure while Solving Weighted Model Integration Problems. In: *Proceedings of the Uncertainty in Artificial Intelligence (UAI) Conference*.

Derkinderen, Vincent; Heylen, Evert; Pedro, Zuidberg Dos Martires; Kolb, Samuel; De Raedt, Luc [2020]. Ordering Variables for Weighted Model Integration. In: *Proceedings of the Uncertainty in Artificial Intelligence (UAI) Conference*.

Demo Papers

Kolb, Samuel; Morettin, Paolo; Zuidberg Dos Martires, Pedro; Sommavilla, Francesco; Passerini, Andrea; Sebastiani, Roberto; De Raedt, Luc [2019a]. The pywmi Framework and Toolbox for Probabilistic Inference using Weighted Model Integration. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Persson, Andreas; Zuidberg Dos Martires, Pedro; De Raedt, Luc; Loutfi, Amy [2020a]. ProbAnch: a Modular Probabilistic Anchoring Framework. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Workshop Papers

Antanas, Laura; Can, Ozan Arkan; Davis, Jesse; De Raedt, Luc; Loutfi, Amy; Persson, Andreas; Saffiotti, Alessandro; Unal, Emre; Yuret, Deniz; Martires, Pedro Zuidberg dos [2017]. Relational Symbol Grounding through Affordance Learning: an Overview of the ReGround Project. In: *International Workshop on Grounding Language Understanding @ INTERSPEECH*.

Zuidberg Dos Martires, Pedro; Dries, Anton; De Raedt, Luc [2018]. Knowledge Compilation with Continuous Random Variables and its Application in Hybrid Probabilistic Logic Programming. In: *Eigth International Workshop on Statistical Relational AI @ IJCAI*.

Can, Ozan Arkan; Zuidberg Dos Martires, Pedro; Persson, Andreas; Gaal, Julian; Loutfi, Amy; De Raedt, Luc; Yuret, Deniz; Saffiotti, Alessandro [2019]. Learning from Implicit Information in Natural Language Instructions for Robotic Manipulations. In: *Combined Workshop on Spatial Language Understanding and Grounded Communication for Robotics @ NAACL*.

Zuidberg Dos Martires, Pedro; Kolb, Samuel [2020]. Monte Carlo Anti-Differentiation for Approximate Weighted Model Integration. In: *Ninth International Workshop on Statistical Relational AI @ AAAI*.

Abstracts

Zuidberg Dos Martires, Pedro; Dumancic, Sebastijan [2018]. Reactive Probabilistic Programming. In: *The International Conference on Probabilistic Programming, Location: Boston, United States of America*.

Zuidberg Dos Martires, Pedro [2019]. Differentiation and Weighted Model Integration. In: *The 1st Workshop on Deep Continuous-Discrete Machine Learning@ ECML, Location: Würzburg*.

Zuidberg Dos Martires, Pedro; Derkinderen, Vincent; Manhaeve, Robin; Meert, Wannes; Kimmig, Angelika; De Raedt, Luc [2019a]. Transforming Probabilistic Programs into Algebraic Circuits for Inference and Learning. In: *Program Transformations for Machine Learning @ NeurIPS 2019*.

In Preparation

Miosic, Ivan; Zuidberg Dos Martires, Pedro [2020]. Measure Theoretic Weighted Model Integration. In: *(in preparation)*.

Zuidberg Dos Martires, Pedro; Kimmig, Angelika; De Raedt, Luc [2020a]. Extending ProbLog with Random Function Symbols. In: *(in preparation)*.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
DTAI
Celestijnenlaan 200A box 2402
B-3001 Leuven
pedro.zudo@kuleuven.be
https://pedrozudo.github.io/